## 1 Basic functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Programming</td>
<td>25</td>
</tr>
<tr>
<td>1.1.1 abort interrupt evaluation.</td>
<td>25</td>
</tr>
<tr>
<td>1.1.2 ans answer</td>
<td>26</td>
</tr>
<tr>
<td>1.1.3 apropos searches keywords in Scilab help</td>
<td>26</td>
</tr>
<tr>
<td>1.1.4 backslash - left matrix division.</td>
<td>26</td>
</tr>
<tr>
<td>1.1.5 bool2s convert boolean matrix to a zero one matrix.</td>
<td>27</td>
</tr>
<tr>
<td>1.1.6 boolean Scilab Objects, boolean variables and operators &amp; !</td>
<td>27</td>
</tr>
<tr>
<td>1.1.7 brackets - left and right brackets</td>
<td>28</td>
</tr>
<tr>
<td>1.1.8 break keyword to interrupt loops</td>
<td>28</td>
</tr>
<tr>
<td>1.1.9 call Fortran or C user routines call</td>
<td>28</td>
</tr>
<tr>
<td>1.1.10 case keyword used in select</td>
<td>30</td>
</tr>
<tr>
<td>1.1.11 clear kills variables</td>
<td>30</td>
</tr>
<tr>
<td>1.1.12 clearglobal kills global variables</td>
<td>31</td>
</tr>
<tr>
<td>1.1.13 colon - colon operator</td>
<td>31</td>
</tr>
<tr>
<td>1.1.14 comma - column, instruction, argument separator</td>
<td>32</td>
</tr>
<tr>
<td>1.1.15 comments comments</td>
<td>32</td>
</tr>
<tr>
<td>1.1.16 debug debugging level</td>
<td>32</td>
</tr>
<tr>
<td>1.1.17 dot - symbol</td>
<td>32</td>
</tr>
<tr>
<td>1.1.18 else keyword in if-then-else</td>
<td>33</td>
</tr>
<tr>
<td>1.1.19 elseif keyword in if-then-else</td>
<td>33</td>
</tr>
<tr>
<td>1.1.20 empty - empty matrix</td>
<td>33</td>
</tr>
<tr>
<td>1.1.21 end end keyword</td>
<td>34</td>
</tr>
<tr>
<td>1.1.22 equal - affectation, comparison equal sign</td>
<td>34</td>
</tr>
<tr>
<td>1.1.23 errcatch error trapping</td>
<td>34</td>
</tr>
<tr>
<td>1.1.24 ercclear error clearing</td>
<td>35</td>
</tr>
<tr>
<td>1.1.25 error error messages</td>
<td>35</td>
</tr>
<tr>
<td>1.1.26 evstr evaluation of expressions</td>
<td>35</td>
</tr>
<tr>
<td>1.1.27 exec script file execution</td>
<td>36</td>
</tr>
<tr>
<td>1.1.28 execstr execute Scilab code in strings</td>
<td>37</td>
</tr>
<tr>
<td>1.1.29 exists checks variable existence</td>
<td>37</td>
</tr>
<tr>
<td>1.1.30 exit Ends the current Scilab session</td>
<td>38</td>
</tr>
<tr>
<td>1.1.31 external Scilab Object, external function or routine</td>
<td>38</td>
</tr>
<tr>
<td>1.1.32 extraction matrix and list entry extraction</td>
<td>39</td>
</tr>
<tr>
<td>1.1.33 feval multiple evaluation</td>
<td>41</td>
</tr>
<tr>
<td>1.1.34 find find indices of boolean vector or matrix true elements</td>
<td>41</td>
</tr>
<tr>
<td>1.1.35 for language keyword for loops</td>
<td>42</td>
</tr>
<tr>
<td>1.1.36 format number printing and display format</td>
<td>42</td>
</tr>
<tr>
<td>1.1.37 fort Fortran or C user routines call</td>
<td>43</td>
</tr>
<tr>
<td>1.1.38 funptr coding of primitives ( wizard stuff )</td>
<td>45</td>
</tr>
<tr>
<td>1.1.39 getenv get the value of an environment variable</td>
<td>45</td>
</tr>
<tr>
<td>1.1.40 getfield list field extraction</td>
<td>45</td>
</tr>
<tr>
<td>1.1.41 getpid get Scilab process identificator</td>
<td>46</td>
</tr>
</tbody>
</table>
1.1.42 getversion  get Scilab version name ........................................ 46
1.1.43 global Define global variable .................................................. 46
1.1.44 gstacksize set/get scilab global stack size .................................. 47
1.1.45 hat - exponentiation ............................................................... 47
1.1.46 host shell (sh) command execution ............................................ 48
1.1.47 hypermat initialize an N dimensional matrices ............................ 48
1.1.48 hypermatrices Scilab object, N dimensional matrices in Scilab ......... 49
1.1.49 iconvert conversion to 1 or 4 byte integer representation ............... 49
1.1.50 ieee set floating point exception mode ....................................... 50
1.1.51 if else - conditional execution ............................................... 50
1.1.52 insertion matrix and list insertion or modification ....................... 51
1.1.53 intppty set interface argument passing properties ........................ 53
1.1.54 inttype type integers used in integer data types .......................... 54
1.1.55 invcoeff build a polynomial matrix from its coefficients ................ 54
1.1.56 iserror error test ..................................................................... 55
1.1.57 isglobal check if a variable is global ........................................ 55
1.1.58 lasterror get last recorded error message ................................... 55
1.1.59 left - left bracket .................................................................. 56
1.1.60 less - lower than comparison ...................................................... 56
1.1.61 list Scilab object and list function definition ................................ 57
1.1.62 lsslist Scilab linear state space function definition ....................... 57
1.1.63 lstcat list concatenation ................................................................ 57
1.1.64 matrices Scilab object, matrices in Scilab ...................................... 58
1.1.65 matrix reshape a vector or a matrix to a different size matrix .......... 58
1.1.66 mlist Scilab object, matrix oriented typed list definition. ................. 59
1.1.67 mode select a mode in exec file .................................................. 59
1.1.68 mtlb_mode switch Matlab like operations ..................................... 60
1.1.69 names scilab names syntax .......................................................... 61
1.1.70 null delete an element in a list .................................................... 61
1.1.71 overloading display, functions and operators overloading capabilities ... 61
1.1.72 parents ) - left and right parenthesis .......................................... 63
1.1.73 pause pause mode, invoke keyboard ............................................ 64
1.1.74 percent - special character .......................................................... 64
1.1.75 plus - addition operator .............................................................. 64
1.1.76 poly polynomial definition .......................................................... 65
1.1.77 power power operation ("^," ) ...................................................... 66
1.1.78 predef variable protection ........................................................... 66
1.1.79 pwd print Scilab current directory ................................................ 67
1.1.80 quit decrease the pause level or exit ............................................ 67
1.1.81 quote - transpose operator, string delimiter ................................... 67
1.1.82 rational Scilab objects, rational in Scilab ....................................... 67
1.1.83 resume return or resume execution and copy some local variables ....... 68
1.1.84 return return or resume execution and copy some local variables ...... 68
1.1.85 rlist Scilab rational fraction function definition ............................. 69
1.1.86 sciargs scilab command line arguments ....................................... 69
1.1.87 select select keyword .................................................................. 69
1.1.88 semi - instruction and row separator ............................................. 70
1.1.89 semicolon - ending expression and row separator .......................... 70
1.1.90 setfield list field insertion ........................................................... 70
1.1.91 slash - right division and feed back ............................................. 71
1.1.92 stacksize set scilab stack size ...................................................... 71
1.1.93 star - multiplication operator ....................................................... 72
1.1.94 symbols scilab operator names ..................................................... 72
1.1.95 testmatrix generate some particular matrices ................................ 73
1.1.96 then keyword in if-then-else ...................................................... 73
1.2.42 getsymbol dialog to select a symbol and its size .................................. 107
1.2.43 gr_menu simple interactive graphic editor ............................................. 107
1.2.44 graduate pretty axis graduations .......................................................... 108
1.2.45 graycolorormap linear gray colormap .................................................... 108
1.2.46 grayplot 2D plot of a surface using colors .......................................... 109
1.2.47 graypolarplot Polar 2D plot of a surface using colors ......................... 109
1.2.48 hist3d 3D representation of a histogram ............................................. 110
1.2.49 histplot plot a histogram ...................................................................... 110
1.2.50 hotcolorormap red to yellow colorormap .............................................. 111
1.2.51 isoview set scales for isometric plot (do not change the size of the window) 111
1.2.52 legends draw graph legend ................................................................... 112
1.2.53 locate mouse selection of a set of points .............................................. 112
1.2.54 m_circle M-circle plot ........................................................................... 113
1.2.55 milkdrop milk drop 3D function ................................................................ 114
1.2.56 nf3d rectangular facets to plot3d parameters ......................................... 114
1.2.57 nyquist nyquist plot ............................................................................... 115
1.2.58 param3d 3D plot of a curve .................................................................... 115
1.2.59 param3d1 3D plot of curves .................................................................... 116
1.2.60 paramfplot2d animated 2D plot, curve defined by a function ............... 117
1.2.61 plot simple plot ....................................................................................... 118
1.2.62 plot2d 2D plot ......................................................................................... 118
1.2.63 plot2d1 2D plot (logarithmic axes) (obsolete) ....................................... 121
1.2.64 plot2d2 2D plot (step function) .............................................................. 122
1.2.65 plot2d3 2D plot (vertical bars) ............................................................... 122
1.2.66 plot2d4 2D plot (arrows style) ............................................................... 123
1.2.67 plot3d 3D plot of a surface ...................................................................... 123
1.2.68 plot3d1 3D gray or color level plot of a surface ................................... 125
1.2.69 plot3d2 plot surface defined by rectangular facets ............................... 126
1.2.70 plot3d3 mesh plot surface defined by rectangular facets ....................... 127
1.2.71 plotframe plot a frame with scaling and grids ........................................ 127
1.2.72 plzr pole-zero plot ............................................................................... 128
1.2.73 polarplot Plot polar coordinates ............................................................ 128
1.2.74 printing printing scilab graphics ............................................................. 129
1.2.75 replot redraw the current graphics window with new boundaries ........... 130
1.2.76 rotate rotation of a set of points ............................................................. 131
1.2.77 scaling affine transformation of a set of points ....................................... 131
1.2.78 sd2sci gr_menu structure to scilab instruction convertor ...................... 131
1.2.79 secto3d 3D surfaces conversion ............................................................. 132
1.2.80 sgrid s-plane grid lines. ......................................................................... 132
1.2.81 square set scales for isometric plot (change the size of the window) ........ 133
1.2.82 subplot divide a graphics window into a matrix of sub-windows ............ 133
1.2.83 titlepage add a title in the middle of a graphics window ...................... 134
1.2.84 winsid return the list of graphics windows ............................................ 134
1.2.85 xarc draw a part of an ellipse .................................................................. 134
1.2.86 xarcs draw parts of a set of ellipses ....................................................... 135
1.2.87 xarrows draw a set of arrows .................................................................. 135
1.2.88 xaxis draw an axis .................................................................................... 136
1.2.89 xbase clear a graphics window and erase the associated recorded graphics 137
1.2.90 xbasimp send graphics to a Postscript printer or in a file ....................... 137
1.2.91 xbasr redraw a graphics window ............................................................ 137
1.2.92 xchange transform real to pixel coordinates ......................................... 137
1.2.93 xclea erase a rectangle ........................................................................... 138
1.2.94 xclear clear a graphics window ............................................................. 138
1.2.95 xclick wait for a mouse click ................................................................. 138
1.2.96 xclip set a clipping zone ........................................................................ 140
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdel</td>
<td>delete a graphics window</td>
</tr>
<tr>
<td>xend</td>
<td>close a graphics session</td>
</tr>
<tr>
<td>xfarc</td>
<td>fill a part of an ellipse</td>
</tr>
<tr>
<td>xfarc5</td>
<td>fill parts of a set of ellipses</td>
</tr>
<tr>
<td>xfpoly</td>
<td>fill a polygon</td>
</tr>
<tr>
<td>xfpolys</td>
<td>fill a set of polygons</td>
</tr>
<tr>
<td>xfrect</td>
<td>fill a rectangle</td>
</tr>
<tr>
<td>xget</td>
<td>get current values of the graphics context</td>
</tr>
<tr>
<td>xgetech</td>
<td>get the current graphics scale</td>
</tr>
<tr>
<td>xgetmouse</td>
<td>get the current position of the mouse</td>
</tr>
<tr>
<td>xgraduate</td>
<td>axis graduation</td>
</tr>
<tr>
<td>xgrid</td>
<td>add a grid on a 2D plot</td>
</tr>
<tr>
<td>xinfo</td>
<td>draw an info string in the message subwindow</td>
</tr>
<tr>
<td>xinit</td>
<td>initialisation of a graphics driver</td>
</tr>
<tr>
<td>xlfont</td>
<td>load a font in the graphics context or query loaded font</td>
</tr>
<tr>
<td>xname</td>
<td>change the name of the current graphics window</td>
</tr>
<tr>
<td>xnumb</td>
<td>draw numbers</td>
</tr>
<tr>
<td>xpause</td>
<td>suspend Scilab</td>
</tr>
<tr>
<td>xpoly</td>
<td>draw a polyline or a polygon</td>
</tr>
<tr>
<td>xpolys</td>
<td>draw a set of polylines or polygons</td>
</tr>
<tr>
<td>xrect</td>
<td>draw a rectangle</td>
</tr>
<tr>
<td>xrects</td>
<td>draw or fill a set of rectangles</td>
</tr>
<tr>
<td>xrpoly</td>
<td>draw a regular polygon</td>
</tr>
<tr>
<td>xs2fig</td>
<td>send graphics to a file in Xfig syntax</td>
</tr>
<tr>
<td>xsave</td>
<td>save graphics into a file</td>
</tr>
<tr>
<td>xsegs</td>
<td>draw unconnected segments</td>
</tr>
<tr>
<td>xselect</td>
<td>raise the current graphics window</td>
</tr>
<tr>
<td>xset</td>
<td>set values of the graphics context</td>
</tr>
<tr>
<td>xsetech</td>
<td>set the sub-window of a graphics window for plotting</td>
</tr>
<tr>
<td>xsetm</td>
<td>dialog to set values of the graphics context</td>
</tr>
<tr>
<td>xstring</td>
<td>draw strings</td>
</tr>
<tr>
<td>xstringb</td>
<td>draw strings into a box</td>
</tr>
<tr>
<td>xstringl</td>
<td>compute a box which surrounds strings</td>
</tr>
<tr>
<td>xtape</td>
<td>set up the record process of graphics</td>
</tr>
<tr>
<td>xtitle</td>
<td>add titles on a graphics window</td>
</tr>
<tr>
<td>zgrid</td>
<td>zgrid plot</td>
</tr>
</tbody>
</table>

### 1.3 Utilities and Elementary Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td>absolute value, magnitude</td>
</tr>
<tr>
<td>acos</td>
<td>element wise cosine inverse</td>
</tr>
<tr>
<td>acosh</td>
<td>hyperbolic cosine inverse</td>
</tr>
<tr>
<td>acoshm</td>
<td>matrix hyperbolic inverse cosine</td>
</tr>
<tr>
<td>acosm</td>
<td>matrix wise cosine inverse</td>
</tr>
<tr>
<td>addf</td>
<td>symbolic addition</td>
</tr>
<tr>
<td>adj2sp</td>
<td>converts adjacency form into sparse matrix</td>
</tr>
<tr>
<td>amell</td>
<td>Jacobi’s am function</td>
</tr>
<tr>
<td>and</td>
<td>logical and</td>
</tr>
<tr>
<td>asin</td>
<td>sine inverse</td>
</tr>
<tr>
<td>asinh</td>
<td>hyperbolic sine inverse</td>
</tr>
<tr>
<td>asinhm</td>
<td>matrix hyperbolic sine inverse</td>
</tr>
<tr>
<td>asinm</td>
<td>matrix wise sine inverse</td>
</tr>
<tr>
<td>atan</td>
<td>2-quadrant and 4-quadrant inverse tangent</td>
</tr>
<tr>
<td>atanhm</td>
<td>matrix hyperbolic tangent inverse</td>
</tr>
<tr>
<td>atanm</td>
<td>square matrix tangent inverse</td>
</tr>
</tbody>
</table>
1.3.18 besseli  Modified I sub ALPHA Bessel functions of the first kind.   ......... 169
1.3.19 besselj Modified J sub ALPHA Bessel functions of the first kind.   ......... 170
1.3.20 besselk Modified K sub ALPHA Bessel functions of the second kind. ......... 170
1.3.21 bessely Modified Y sub ALPHA Bessel functions of the second kind. ......... 171
1.3.22 binomial binomial distribution                                  ......... 171
1.3.23 bloc2exp block-diagram to symbolic expression ..................... 172
1.3.24 bloc2ss block-diagram to state-space conversion ................... 173
1.3.25 calerf computes error functions.                                ...... 175
1.3.26 ceil rounding up                                           .......... 176
1.3.27 cmb lin symbolic linear combination                           ......... 176
1.3.28 conj conjugate                                             ......... 176
1.3.29 cos cosine function                                        ......... 177
1.3.30 cosh hyperbolic cosine                                      ......... 177
1.3.31 coshm matrix hyperbolic cosine                               ......... 178
1.3.32 cosm matrix cosine function                                 ......... 178
1.3.33 cotg cotangent                                             ......... 178
1.3.34 coth hyperbolic cotangent                                   ......... 179
1.3.35 cothm matrix hyperbolic cotangent                           ......... 179
1.3.36 cumpd cumulative product                                   ......... 179
1.3.37 cumsum cumulative sum                                       ......... 180
1.3.38 delip elliptic integral                                     ......... 180
1.3.39 diag diagonal including or extracting                      ......... 181
1.3.40 dgamma derivative of gammaln function.                    ......... 182
1.3.41 double conversion from integer to double precision representation ...... 182
1.3.42 erf The error function.                                    ......... 182
1.3.43 erfc The complementary error function.                     ......... 183
1.3.44 erfcx scaled complementary error function.                 ......... 183
1.3.45 eval evaluation of a matrix of strings                     ......... 184
1.3.46 eye identity matrix                                        ......... 184
1.3.47 fix rounding towards zero                                   ......... 184
1.3.48 floor rounding down                                        ......... 185
1.3.49 frexp dissect floating-point numbers into base 2 exponent and mantissa ...... 185
1.3.50 full sparse to full matrix conversion                       ......... 186
1.3.51 gamma The gamma function.                                   ......... 186
1.3.52 gammaln The logarithm of gamma function.                   ......... 186
1.3.53 gsort decreasing order sorting                             ......... 187
1.3.54 imag imaginary part                                        ......... 188
1.3.55 int integer part                                            ......... 188
1.3.56 int8 conversion to one byte integer representation          ......... 188
1.3.57 integrate integration by quadrature                        ......... 189
1.3.58 interp interpolation                                        ......... 189
1.3.59 interpln linear interpolation                              ......... 189
1.3.60 intersect returns the vector of common values of two vectors ...... 190
1.3.61 intsplin integration of experimental data by spline interpolation ...... 191
1.3.62 inttrap integration of experimental data by trapezoidal interpolation ...... 192
1.3.63 isdef check variable existence                             ......... 192
1.3.64 isinf check for infinite entries                          ......... 193
1.3.65 isnan check for “Not a Number” entries                    ......... 193
1.3.66 isreal check if a variable as real or complex entries      ......... 193
1.3.67 kron Kronecker product (.*.)                                ......... 194
1.3.68 ldif left symbolic division                                 ......... 194
1.3.69 lexsort lexicographic matrix rows sorting                  ......... 194
1.3.70 linspace linearly spaced vector                             ......... 195
1.3.71 log natural logarithm                                      ......... 195
1.3.72 log10 logarithm                                           ......... 196
1.3.73 log2 base 2 logarithm .................................. 196
1.3.74 logm square matrix logarithm .......................... 196
1.3.75 logspace logarithmically spaced vector ................. 197
1.3.76 max maximum .......................................... 197
1.3.77 maxi maximum .......................................... 198
1.3.78 mean mean (row mean, column mean) of vector/matrix entries ... 198
1.3.79 median median (row median, column median) of vector/matrix entries ... 199
1.3.80 min minimum ........................................... 199
1.3.81 mini minimum ........................................... 200
1.3.82 minus - substraction operator, sign changes .......... 201
1.3.83 modulo symmetric arithmetic remainder modulo m .......... 201
1.3.84 mps2linpro convert lp problem given in MPS format to linpro format .... 202
1.3.85 mtlb_sparse convert sparse matrix ....................... 202
1.3.86 mult symbolic multiplication ........................... 203
1.3.87 nnz number of non zero entries in a matrix ............. 203
1.3.88 norm matrix norms ..................................... 203
1.3.89 not - logical not ........................................ 204
1.3.90 ones matrix made of ones ................................ 204
1.3.91 or - logical or ......................................... 205
1.3.92 pen2ea pencil to E,A conversion .......................... 205
1.3.93 pertrans pertranspose .................................. 206
1.3.94 prod product ............................................ 206
1.3.95 rand random number generator .......................... 207
1.3.96 rat Floating point rational approximation ............... 207
1.3.97 rdifv right symbolic division ........................... 208
1.3.98 real real part ........................................... 208
1.3.99 round rounding .......................................... 209
1.3.100 sign sign function ..................................... 209
1.3.101 signm matrix sign function ............................ 209
1.3.102 sin sine function ...................................... 210
1.3.103 sinh hyperbolic sine ................................... 210
1.3.104 sinhm matrix hyperbolic sine .......................... 210
1.3.105 sinm matrix sine function .............................. 211
1.3.106 size size of objects .................................... 211
1.3.107 smooth smoothing by spline functions .................. 212
1.3.108 solve symbolic linear system solver .................... 212
1.3.109 sort decreasing order sorting .......................... 213
1.3.110 sp2adj converts sparse matrix into adjacency form ...... 213
1.3.111 sparse sparse matrix definition ........................ 214
1.3.112 spcompack converts a compressed adjacency representation .... 215
1.3.113 speye sparse identity matrix ............................ 216
1.3.114 spget retrieves entries of sparse matrix ................ 216
1.3.115 splin spline function ................................... 217
1.3.116 spones sparse matrix ................................... 218
1.3.117 sprand sparse random matrix ........................... 218
1.3.118 spzeros sparse zero matrix ............................ 218
1.3.119 sqrt square root ....................................... 219
1.3.120 sqrtm matrix square root .............................. 219
1.3.121 squarewave generates a square wave with period 2*%pi .......... 220
1.3.122 ssprint pretty print for linear system .................. 220
1.3.123 ssrand random system generator ......................... 220
1.3.124 std deviation standard deviation (row or column-wise) of vector/matrix entries ... 221
1.3.125 subf symbolic subtraction ............................... 222
1.3.126 sum sum (row sum, column sum) of vector/matrix entries .......... 222
1.3.127 sysconv system conversion ............................... 223
1.3.128 sysdiag  block diagonal system connection .................................. 224
1.3.129 syslin linear system definition .............................................. 224
1.3.130 tan  tangent .............................................................................. 225
1.3.131 tanh  hyperbolic tangent .......................................................... 226
1.3.132 tanhm  matrix hyperbolic tangent ............................................. 226
1.3.133 tanm  matrix tangent ............................................................... 227
1.3.134 toeplitz  toeplitz matrix .......................................................... 227
1.3.135 trfmod  poles and zeros display ................................................ 228
1.3.136 trianfml  symbolic triangularization ........................................ 228
1.3.137 tril  lower triangular part of matrix .......................................... 228
1.3.138 trisolve  symbolic linear system solver ...................................... 229
1.3.139 triu  upper triangle .................................................................. 229
1.3.140 typeof  object type .................................................................. 229
1.3.141 union  extract union components of a vector .............................. 230
1.3.142 unique  extract unique components of a vector ............................ 230
1.3.143 zeros  matrix made of zeros ...................................................... 231
1.4 Input/Output functions .................................................................... 232
1.4.1 diary  diary of session .................................................................... 233
1.4.2 disp  displays variables .................................................................. 233
1.4.3 dispfile  display opened files properties ........................................ 233
1.4.4 file  file management .................................................................... 234
1.4.5 fileinfo  Provides information about a file ................................. 235
1.4.6 fprintf  Emulator of C language fprintf function ....................... 235
1.4.7 fprintfMat  print a matrix in a file. .............................................. 236
1.4.8 fscanf  Converts formatted input read on a file ........................... 237
1.4.9 fscanfMat  Reads a Matrix from a text file. ................................. 237
1.4.10 getio  get Scilab input/output logical units .................................. 238
1.4.11 input  prompt for user input .......................................................... 238
1.4.12 lines  rows and columns used for display .................................... 238
1.4.13 load  load saved variable ............................................................ 239
1.4.14 manedit  editing a manual item ................................................... 239
1.4.15 mclearerr  reset binary file access errors .................................... 240
1.4.16 mclose  close an opened file ....................................................... 240
1.4.17 meof  check if end of file has been reached .................................. 241
1.4.18 mscanf  interface to the C scanf function .................................... 241
1.4.19 mget  reads byte or word in a given binary format and convert to double .................................................. 242
1.4.20 mgetl  read lines from an ascii file ............................................ 243
1.4.21 mgetstr  read a character string ................................................ 244
1.4.22 mopen  open a file ...................................................................... 244
1.4.23 mfprintf  converts, formats, and writes data to a file ................. 245
1.4.24 mput  writes byte or word in a given binary format ...................... 246
1.4.25 mputl  writes strings in an ascii file .......................................... 247
1.4.26 mputstr  write a character string in a file .................................. 247
1.4.27 mfscanf  scan data from file ........................................................ 248
1.4.28 mseek  set current position in binary file. .................................. 248
1.4.29 mtell  binary file management ................................................... 249
1.4.30 newest  returns newest file of a set of files ............................... 250
1.4.31 oldload  load saved variable in 2.4.1 and previous formats ........ 250
1.4.32 oldsave  saving variables in 2.4.1 and previous format ............... 251
1.4.33 print  prints variables in a file ..................................................... 251
1.4.34 printf  Emulator of C language printf function ....................... 252
1.4.35 printf conversion  printf, sprintf, fprintf conversion specifications .... 252
1.4.36 read  matrices read .................................................................. 254
1.4.37 read4b  fortran file binary read ................................................ 255
1.4.38 readb  fortran file binary read ................................................... 255
1.4.39 readc read a character string
1.4.40 readmps reads a file in MPS format
1.4.41 save saving variables in binary files
1.4.42 scanf Converts formatted input on standard input
1.4.43 scanf Conversion scanf, sscanf, fscanf Conversion specifications
1.4.44 sprintf Emulator of C language sprintf function
1.4.45 sscanf Converts formatted input given by a string
1.4.46 startup startup file
1.4.47 warning warning messages
1.4.48 write fortran file binary write
1.4.49 write write in a formatted file
1.4.50 write4b fortran file binary write
1.4.51 xgetfile dialog to get a file path

1.5 Handling of functions and libraries
1.5.1 addinter new functions interface incremental linking at run time
1.5.2 argn number of arguments in a function call
1.5.3 clearfun remove primitive
1.5.4 comp scilab function compilation
1.5.5 deff on-line definition of function
1.5.6 delbpt delete breakpoint
1.5.7 dispbpt display breakpoints
1.5.8 edit function editing
1.5.9 funcprot switch scilab functions protection mode
1.5.10 function opens a function definition
1.5.11 functions Scilab procedures and Scilab objects
1.5.12 genlib build library from all functions in given directory
1.5.13 getfunction get source file path of a library function
1.5.14 getd getting all functions defined in a directory
1.5.15 getf defining a function from a file
1.5.16 lib library definition
1.5.17 macr2lst function to list conversion
1.5.18 macro Scilab procedure and Scilab object
1.5.19 macrovar variables of function
1.5.20 newfun add a name in the table of functions
1.5.21 plotprofile extracts and displays execution profiles of a Scilab function
1.5.22 profile extract execution profiles of a Scilab function
1.5.23 setbpt setting breakpoints
1.5.24 showprofile extracts and displays execution profiles of a Scilab function
1.5.25 varargin variable numbers of arguments in an input argument list
1.5.26 varargout variable numbers of arguments in an output argument list

1.6 Strings manipulations
1.6.1 code2str returns character string associated with Scilab integer codes
1.6.2 convstr case conversion
1.6.3 emptystr zero length string
1.6.4 grep find matches of a string in a vector of strings
1.6.5 length length of object
1.6.6 part extraction of strings
1.6.7 str2code return scilab integer codes associated with a character string
1.6.8 strcat catenate character strings
1.6.9 strindex search position of a character string in an other string
1.6.10 string conversion to string
1.6.11 strings Scilab Object, character strings
1.6.12 stripblanks strips leading and trailing blanks of strings
1.6.13 strsubst substitute a character string by another in a character string

1.7 Dialogs
1.7.1 addmenu interactive button or menu definition .......................... 287
1.7.2 delmenu interactive button or menu deletion ............................ 288
1.7.3 getvalue xwindow dialog for data acquisition ........................... 288
1.7.4 halt stop execution ................................. 289
1.7.5 havewindow return scilab window mode ............................... 289
1.7.6 keyboard keyboard commands ...................................... 290
1.7.7 setmenu interactive button or menu activation ........................... 290
1.7.8 unsetmenu interactive button or menu or submenu de-activation .......... 290
1.7.9 x_choices interactive Xwindow choices through toggle buttons .......... 291
1.7.10 x_choose interactive Xwindow choice .................................. 291
1.7.11 x_dialog Xwindow dialog ........................................... 292
1.7.12 x_matrix Xwindow editing of matrix .............................. 292
1.7.13 x_mndialog Xwindow dialog ......................................... 293
1.7.14 x_message X window message ..................................... 293
1.7.15 x_message_modeless X window modeless message ......................... 294

1.8 Utilities ............................................. 295
1.8.1 %helps Variable defining the path of help directories ....................... 296
1.8.2 G_make call make or nmake ........................................... 296
1.8.3 c_link check dynamic link ............................................ 297
1.8.4 chdir changes Scilab current directory ................................ 297
1.8.5 dec2hex hexadecimal representation of integers .......................... 298
1.8.6 demos guide for scilab demos ........................................ 298
1.8.7 help on-line help command ......................................... 298
1.8.8 hex2dec converts hexadecimal representation of integers to numbers .... 299
1.8.9 ilib_build utility for shared library management ........................ 299
1.8.10 ilib_compile ilib_build utility: executes the makefile produced by ilib_genMake . 300
1.8.11 ilib_for_link utility for shared library management with link .......... 301
1.8.12 ilib_genMake utility for ilib_build: produces a makefile for building shared libraries 302
1.8.13 ilib_gen_gateway utility for ilib_build, generates a gateway file ........ 302
1.8.14 ilib_gen_loader utility for ilib_build: generates a loader file ........ 302
1.8.15 intersci scilab tool to interface C of Fortran functions with scilab .... 303
1.8.16 link dynamic link ............................................... 303
1.8.17 man on line help source file description format .......................... 305
1.8.18 sci2exp converts variable to expression ................................ 307
1.8.19 sci2map Scilab to Maple variable conversion ........................... 308
1.8.20 scilab Major unix script to execute Scilab and miscellaneous tools .... 308
1.8.21 scilink Unix script to relink Scilab ................................ 309
1.8.22 ulink unlink a dynamically linked shared object ......................... 309
1.8.23 unix shell (sh) command execution .................................. 310
1.8.24 unix_g shell (sh) command execution, output redirected to a variable .... 310
1.8.25 unix_s shell (sh) command execution, no output ........................ 311
1.8.26 unix_w shell (sh) command execution, output redirected to scilab window ... 311
1.8.27 unix_x shell (sh) command execution, output redirected to a window .... 312

1.9 Time and date ........................................... 313
1.9.1 date Current date as date string .................................... 314
1.9.2 getdate get date and time information ................................ 314
1.9.3 timer cpu time ................................................. 315

2 Specialized Toolboxes 317
2.1 General System and Control macros ....................................... 317
2.1.1 abcd state-space matrices ......................................... 318
2.1.2 abinv AB invariant subspace ....................................... 318
2.1.3 arhank Hankel norm approximant .................................. 321
2.1.4 arl2 SISO model realization by L2 transfer approximation ............. 321
2.1.5 balreal balanced realization ...................................... 322
2.1.6 bilin general bilinear transform ........................................... 323
2.1.7 cainv Dual of abinv .......................................................... 323
2.1.8 calfrq frequency response discretization ............................... 324
2.1.9 canon canonical controllable form ........................................ 325
2.1.10 cls2dls bilinear transform ................................................ 326
2.1.11 colregul removing poles and zeros at infinity ....................... 326
2.1.12 cont_frm transfer to controllable state-space ....................... 327
2.1.13 cont_mat controllability matrix ....................................... 327
2.1.14 contr controllability, controllable subspace ......................... 328
2.1.15 contrss controllable part ................................................ 328
2.1.16 csim simulation (time response) of linear system .................... 329
2.1.17 ctr_gram controllability gramian .................................... 330
2.1.18 dbphi frequency response to phase and magnitude representation .... 330
2.1.19 ddp disturbance decoupling ............................................. 331
2.1.20 des2tf descriptor to transfer function conversion ................... 332
2.1.21 dscr discretization of linear system .................................. 333
2.1.22 dsimul state space discrete time simulation ......................... 333
2.1.23 dtility detectability test .................................................. 334
2.1.24 equil balancing of pair of symmetric matrices ....................... 334
2.1.25 equil1 balancing (nonnegative) pair of matrices .................... 335
2.1.26 feedback feedback operation .......................................... 335
2.1.27 flts time response (discrete time, sampled system) ................. 336
2.1.28 freq time response (discrete time, sampled system) .................. 338
2.1.29 freq_freq frequency response .......................................... 339
2.1.30 freson peak frequencies ................................................ 339
2.1.31 g_margin gain margin .................................................. 340
2.1.32 gfrancis Francis equations for tracking ................................ 340
2.1.33 imrep2ss state-space realization of an impulse response .......... 341
2.1.34 invsyslin system inversion ........................................... 342
2.1.35 kpure continuous SISO system limit feedback gain .................. 342
2.1.36 krac2 continuous SISO system limit feedback gain .................. 343
2.1.37 lin linearization .......................................................... 343
2.1.38 lqe linear quadratic estimator (Kalman Filter) ....................... 344
2.1.39 lqr LQG to standard problem .......................................... 344
2.1.40 lqr LQG to standard problem .......................................... 345
2.1.41 lqg LQG compensator .................................................... 345
2.1.42 lqg2stan LQG to standard problem .................................... 345
2.1.43 lqg2stan LQG to standard problem .................................... 347
2.1.44 linreml minimal balanced realization .................................. 347
2.1.45 minreal minimal balanced realization .................................. 348
2.1.46 minss minimal realization .............................................. 348
2.1.47 observer based controller .............................................. 349
2.1.48 observer design ........................................................ 349
2.1.49 obsvmat observability matrix .......................................... 350
2.1.50 obsvss observable part ................................................ 351
2.1.51 p_margin phase margin ................................................. 352
2.1.52 pfss partial fraction decomposition ................................... 353
2.1.53 phasemag phase and magnitude computation .......................... 353
2.1.54 phasemag phase and magnitude computation .......................... 354
2.1.55 projsl linear system projection ...................................... 354
2.1.56 repfreq frequency response ........................................... 355
2.1.57 ricc Riccati equation .................................................. 356
2.1.58 rowregul removing poles and zeros at infinity ...................... 357
2.1.59 rtitr discrete time response (transfer matrix) ....................... 358
2.1.60 sm2des system matrix to descriptor .................................. 360
2.1.61 sm2ss system matrix to state-space ........................................ 360
2.1.62 specfact spectral factor ..................................................... 360
2.1.63 ss2des (polynomial) state-space to descriptor form .................... 361
2.1.64 ss2ss state-space to state-space conversion, feedback, injection ....... 362
2.1.65 ss2tf conversion from state-space to transfer function ................. 363
2.1.66 stablity stabilizability test ............................................... 364
2.1.67 svplot singular-value sigma-plot ........................................ 365
2.1.68 sysfact- factorization ...................................................... 366
2.1.70 syssize size of state-space system ..................................... 367
2.1.71 tf2ss transfer to state-space .............................................. 367
2.1.72 timeid SISO least square identification .................................. 367
2.1.73 trzeros transmission zeros and normal rank ................................ 368
2.1.74 uiobserver unknown input observer ..................................... 369
2.1.75 unobs unobservable subspace ............................................. 370
2.1.76 zeropen zero pencil ......................................................... 371
2.2 Robust control toolbox ................................................................... 372
2.2.1 augment augmented plant ..................................................... 373
2.2.2 bstap hankel approximant .................................................... 374
2.2.3 ccontrg central H-infinity controller ..................................... 374
2.2.4 colinout inner-outer factorization ....................................... 375
2.2.5 copfac right coprime factorization ....................................... 375
2.2.6 dcf double coprime factorization ......................................... 375
2.2.7 des2ss descriptor to state-space ........................................... 376
2.2.8 dnorm discrete H-infinity norm ........................................... 376
2.2.9 dtsi stable anti-stable decomposition ................................... 377
2.2.10 fourplan augmented plant to four plants ................................ 377
2.2.11 fspecg stable factorization ................................................ 377
2.2.12 fstabst Youla’s parametrization ........................................ 378
2.2.13 gamitg H-infinity gamma iterations .................................... 378
2.2.14 gcare control Riccati equation .......................................... 379
2.2.15 gfare filter Riccati equation ............................................... 379
2.2.16 gtild tilde operation .......................................................... 380
2.2.17 h2norm H2 norm ................................................................ 381
2.2.18 hcl closed loop matrix .......................................................... 381
2.2.19 hinf H-infinity (central) controller ......................................... 381
2.2.20 hinf_st static H_infinity problem ......................................... 382
2.2.21 hnorm H-infinity norm ........................................................ 382
2.2.22 hankelsv Hankel singular values ........................................ 383
2.2.23 lcf normalized coprime factorization .................................... 383
2.2.24 lqgr H-infinity LQ gain (full state) ...................................... 383
2.2.25 lift linear fractional transformation ...................................... 384
2.2.26 linf infinity norm ................................................................. 385
2.2.27 linfn infinity norm ............................................................... 385
2.2.28 lqg Ltr LQG with loop transform recovery ............................. 386
2.2.29 macglov Mac Farlane Glover problem .................................. 387
2.2.30 nehari Nehari approximant .................................................. 387
2.2.31 parrot Parrot’s problem ....................................................... 387
2.2.32 ric_desc Riccati equation .................................................... 388
2.2.33 riccati Riccati equation ....................................................... 389
2.2.34 rowinout inner-outer factorization ...................................... 389
2.2.35 sensi sensitivity functions .................................................. 390
2.2.36 tf2des transfer function to descriptor .................................. 390
2.3 Tools for dynamical systems ...................................................... 392
2.3.1 arma Scilab arma library ....................................................... 393
2.3.2 arma2p extract polynomial matrices from ar representation .......................... 393
2.3.3 armac Scilab description of an armax process ........................................... 394
2.3.4 armax armax identification ............................................................................. 395
2.3.5 armax1 armax identification .......................................................................... 396
2.3.6 arsimul armax simulation .............................................................................. 396
2.3.7 narsimul armax simulation (using rtitr) ....................................................... 397
2.3.8 noisegen noise generation ............................................................................. 397
2.3.9 odedi test of ode ......................................................................................... 397
2.3.10 prbs a pseudo random binary sequences generation ................................... 398
2.3.11 reglin Linear regression .............................................................................. 398

2.4 Examples ............................................................................................................ 399
2.4.1 artest arnold dynamical system ................................................................. 400
2.4.2 bifish shows a bifurcation diagram in a fish population discrete time model ... 400
2.4.3 boucle phase portrait of a dynamical system with observer ......................... 401
2.4.4 chaintest a three-species food chain model ................................................. 401
2.4.5 gpeche a fishing program .............................................................................. 402
2.4.6 fusee a set of Scilab macro for a landing rocket problem ............................ 402
2.4.7 lotest demo of the Lorenz attractor .............................................................. 403
2.4.8 mine a mining problem ................................................................................ 404
2.4.9 obscnt1 a controlled-observed system ....................................................... 405
2.4.10 port3d 3 dimensional phase portrait .......................................................... 405
2.4.11 portrait 2 dimensional phase portrait ......................................................... 406
2.4.12 recur a bilinear recurrent equation ............................................................. 406
2.4.13 systems a collection of dynamical system .................................................. 407
2.4.14 tangent linearization of a dynamical system at an equilibrium point .......... 408
2.4.15 tdinit interactive initialisation of the tdes dynamical systems .................... 409

2.5 Non-linear tools (optimization and simulation) .................................................. 410
2.5.1 bvode boundary value problems for ODE ................................................... 411
2.5.2 colnew boundary value problems for ODE ............................................... 414
2.5.3 dasrt DAE solver with zero crossing ............................................................ 414
2.5.4 dassl differential algebraic equation ............................................................ 416
2.5.5 datafit Parameter identification based on measured data ......................... 417
2.5.6 derivative derivative .................................................................................. 419
2.5.7 fitdat Parameter identification based on measured data ............................ 419
2.5.8 lsove find a zero of a system of n nonlinear functions ............................... 420
2.5.9 impl differential algebraic equation .............................................................. 421
2.5.10 int2d definite 2D integral by quadrature and cubature method .................. 422
2.5.11 int3d definite 3D integral by quadrature and cubature method .................. 423
2.5.12 intc Cauchy integral .................................................................................. 425
2.5.13 intg definite integral ................................................................................... 425
2.5.14 intl Cauchy integral ................................................................................... 426
2.5.15 karmarkar karmarkar algorithm ................................................................. 426
2.5.16 leastsq Solves non-linear least squares problems ..................................... 427
2.5.17 linpro linear programming solver .............................................................. 429
2.5.18 lmisolver linear matrix inequation solver ............................................... 430
2.5.19 lmitool tool for solving linear matrix inequalities ..................................... 431
2.5.20 ode ordinary differential equation solver .................................................. 431
2.5.21 ode discrete ordinary differential equation solver, discrete time simulation . 434
2.5.22 ode_root ordinary differential equation solver with root finding .............. 435
2.5.23 odedc discrete/continuous ode solver ...................................................... 436
2.5.24 odeoptions set options for ode solvers ...................................................... 437
2.5.25 optim non-linear optimization routine ...................................................... 438
2.5.26 quapro linear quadratic programming solver .......................................... 440
2.5.27 semidef semidefinite programming ............................................................ 441

2.6 Signal Processing toolbox .................................................................................. 444
2.6.1 %asn elliptic integral ................................ 445
2.6.2 %k Jacobi's complete elliptic integral ..................... 445
2.6.3 %sn Jacobi's elliptic function ................................... 446
2.6.4 analpf create analog low-pass filter ............................ 446
2.6.5 buttmag response of Butterworth filter ............................. 447
2.6.6 casc cascade realization of filter from coefficients .................. 447
2.6.7 cepstrum cepstrum calculation .................................... 448
2.6.8 cheb1mag response of Chebyshev type 1 filter .................... 448
2.6.9 cheb2mag response of type 2 Chebyshev filter .................... 449
2.6.10 chepol Chebychev polynomial ..................................... 449
2.6.11 convol convolution ................................................ 450
2.6.12 corr correlation, covariance ...................................... 450
2.6.13 cspect spectral estimation (correlation method) .................. 452
2.6.14 czt chirp z-transform algorithm ................................... 453
2.6.15 dft discrete Fourier transform ................................... 454
2.6.16 ell1mag magnitude of elliptic filter .............................. 455
2.6.17 eqfir minimax approximation of FIR filter ....................... 455
2.6.18 eqfir Design of iir filters ..................................... 456
2.6.19 faurre filter computation by simple Faurre algorithm ............. 456
2.6.20 flipt coefficients of FIR low-pass ................................ 457
2.6.21 fitt fast Fourier transform. ..................................... 457
2.6.22 filter modelling filter ............................................ 458
2.6.23 findfreq parameter compatibility for elliptic filter design .......... 458
2.6.24 findm for elliptic filter design .................................. 459
2.6.25 frfit frequency response fit ..................................... 459
2.6.26 frmag magnitude of FIR and IIR filters ....................... 460
2.6.27 fsfirlin design of FIR, linear phase filters, frequency sampling technique ...... 460
2.6.28 group group delay for digital filter .............................. 461
2.6.29 hank covariance to hankel matrix ................................ 461
2.6.30 hilb Hilbert transform ............................................ 462
2.6.31 iir iir digital filter .............................................. 463
2.6.32 iirgroup group delay Lp IIR filter optimization .................. 463
2.6.33 iirlp Lp IIR filter optimization .................................. 464
2.6.34 intdec Changes sampling rate of a signal ........................ 464
2.6.35 jmat row or column block permutation ............................ 464
2.6.36 kalm Kalman update .............................................. 465
2.6.37 lattn recursive solution of normal equations .................... 465
2.6.38 lattp lattp ..................................................... 466
2.6.39 lev Yule-Walker equations (Levinson's algorithm) ............... 466
2.6.40 levin Toeplitz system solver by Levinson algorithm (multidimensional) ...... 466
2.6.41 lgfft utility for fft ................................................ 468
2.6.42 lindquist Lindquist's algorithm .................................. 469
2.6.43 mese maximum entropy spectral estimation ........................ 469
2.6.44 mfft multi-dimensional fft ...................................... 470
2.6.45 mrfft frequency response fit ..................................... 470
2.6.46 phc Markovian representation ..................................... 471
2.6.47 pspect cross-spectral estimate between 2 series .................... 472
2.6.48 remez Remes's algorithm ........................................... 473
2.6.49 remezb Minimax approximation of magnitude response ............ 473
2.6.50 rpem RPEM estimation .............................................. 474
2.6.51 sinc samples of sinc function .................................... 476
2.6.52 sindc digital sinc function or Direchlet kernel ................. 476
2.6.53 srfaur square-root algorithm ..................................... 476
2.6.54 srfur square root Kalman filter ................................... 477
2.6.55 ssrf steady-state Kalman filter .................................... 477
| 2.8.7 | classmarkov | recurrent and transient classes of Markov matrix | 504 |
| 2.8.8 | coff | resolvent (cofactor method) | 505 |
| 2.8.9 | colcomp | column compression, kernel, nullspace | 505 |
| 2.8.10 | companion | companion matrix | 506 |
| 2.8.11 | cond | condition number | 506 |
| 2.8.12 | det | determinant | 507 |
| 2.8.13 | eigenmarkov | normalized left and right Markov eigenvectors | 507 |
| 2.8.14 | ereduc | computes matrix column echelon form by qz transformations | 508 |
| 2.8.15 | exp | element-wise exponential | 508 |
| 2.8.16 | expm | square matrix exponential | 509 |
| 2.8.17 | fstair | computes pencil column echelon form by qz transformations | 509 |
| 2.8.18 | fullrf | full rank factorization | 510 |
| 2.8.19 | fullrfk | full rank factorization of A^k | 510 |
| 2.8.20 | genmarkov | generates random markov matrix with recurrent and transient classes | 511 |
| 2.8.21 | givens | Givens transformation | 511 |
| 2.8.22 | glever | inverse of matrix pencil | 512 |
| 2.8.23 | gschur | generalized Schur form (matrix pencils). | 513 |
| 2.8.24 | gspec | eigenvalues of matrix pencil | 514 |
| 2.8.25 | hess | Hessenberg form | 514 |
| 2.8.26 | householder | Householder orthogonal reflexion matrix | 515 |
| 2.8.27 | iminv | inverse image | 515 |
| 2.8.28 | inv | matrix inverse | 516 |
| 2.8.29 | kernel | kernel, nullspace | 516 |
| 2.8.30 | kroneck | Kronecker form of matrix pencil | 517 |
| 2.8.31 | linsolve | linear equation solver | 518 |
| 2.8.32 | lu | LU factors of Gaussian elimination | 519 |
| 2.8.33 | luded | utility function used with lufact | 519 |
| 2.8.34 | lufact | sparse lu factorization | 520 |
| 2.8.35 | luget | sparse lu factorization | 520 |
| 2.8.36 | lusolve | sparse linear system solver | 521 |
| 2.8.37 | lyap | Lyapunov equation | 522 |
| 2.8.38 | nlev | Leverrier’s algorithm | 522 |
| 2.8.39 | orth | orthogonal basis | 523 |
| 2.8.40 | pbig | eigen-projection | 523 |
| 2.8.41 | pencan | canonical form of matrix pencil | 524 |
| 2.8.42 | penlaur | Laurent coefficients of matrix pencil | 524 |
| 2.8.43 | pinv | pseudoinverse | 525 |
| 2.8.44 | polar | polar form | 525 |
| 2.8.45 | proj | projection | 526 |
| 2.8.46 | prospec | spectral operators | 526 |
| 2.8.47 | psmall | spectral projection | 527 |
| 2.8.48 | qr | QR decomposition | 528 |
| 2.8.49 | quaskro | quasi-Kronecker form | 528 |
| 2.8.50 | randpencil | random pencil | 529 |
| 2.8.51 | range | range (span) of A^k | 530 |
| 2.8.52 | rank | rank | 530 |
| 2.8.53 | rcond | inverse condition number | 531 |
| 2.8.54 | rowcomp | row compression, range | 531 |
| 2.8.55 | rowshuff | shuffle algorithm | 532 |
| 2.8.56 | rref | computes matrix row echelon form by lu transformations | 532 |
| 2.8.57 | schur | [ordered] Schur decomposition | 533 |
| 2.8.58 | spaninter | subspace intersection | 534 |
| 2.8.59 | spanplus | sum of subspaces | 535 |
| 2.8.60 | spantwo | sum and intersection of subspaces | 535 |
| 2.8.61 | spchol | sparse cholesky factorization | 536 |
2.8.62 spec eigenvalues ........................................ 537
2.8.63 sroot W*W' hermitian factorization ..................... 537
2.8.64 sva singular value approximation ........................ 538
2.8.65 svd singular value decomposition ........................ 538
2.8.66 sylv Sylvester equation ................................. 539
2.8.67 trace trace ........................................... 539

2.9 Metanet ................................................. 540
2.9.1 add_edge adds an edge or an arc between two nodes .......................... 541
2.9.2 add_node adds a disconnected node to a graph ......................... 541
2.9.3 adj_lists computes adjacency lists .......................... 542
2.9.4 arc_graph graph with nodes corresponding to arcs .................... 543
2.9.5 arc_number number of arcs of a graph ....................... 543
2.9.6 articul finds one or more articulation points ..................... 544
2.9.7 bandwidth bandwidth reduction for a sparse matrix .................. 544
2.9.8 best_match best matching of a graph ....................... 545
2.9.9 chain_graph chained structure from adjacency lists of a graph ....... 546
2.9.10 check_graph checks a Scilab graph list ..................... 547
2.9.11 circuit finds a circuit or the rank function in a directed graph ........ 548
2.9.12 conn_nodes set of nodes of a connected component .................. 548
2.9.13 connex connected components ................................ 549
2.9.14 contract_edge contracts edges between two nodes .................. 549
2.9.15 convex_hull convex hull of a set of points in the plane ............ 550
2.9.16 cycle_basis basis of cycle of a simple undirected graph .......... 551
2.9.17 delete_arcs deletes all the arcs or edges between a set of nodes .... 551
2.9.18 delete_nodes deletes nodes ................................ 552
2.9.19 edge_number number of edges of a graph ..................... 553
2.9.20 find_path finds a path between two nodes ..................... 553
2.9.21 gen_net generation of a network ............................ 554
2.9.22 girth girth of a directed graph ............................ 555
2.9.23 glist graph list creation ................................ 555
2.9.24 graph_list description of graph list ........................ 555
2.9.25 graph_mat node-arc or node-node incidence matrix of a graph ........... 558
2.9.26 graph_center center of a graph ............................ 558
2.9.27 graph_complement complement of a graph ........................ 559
2.9.28 graph_diameter diameter of a graph .......................... 560
2.9.29 graph_power kth power of a directed 1-graph .................... 560
2.9.30 graph_simp converts a graph to a simple undirected graph .......... 561
2.9.31 graph_sum sum of two graphs ............................... 561
2.9.32 graph_union union of two graphs ............................ 562
2.9.33 hamilton hamiltonian circuit of a graph ........................ 563
2.9.34 is_connex connectivity test ................................ 563
2.9.35 knapsack solves a 0-1 multiple knapsack problem .................. 564
2.9.36 line_graph graph with nodes corresponding to edges ................. 564
2.9.37 load_graph loads a graph ................................ 565
2.9.38 make_graph makes a graph list ................................ 566
2.9.39 mat_graph graph from node-arc or node-node incidence matrix ....... 566
2.9.40 max_flow_path maximum capacity path ......................... 567
2.9.41 max_clique maximum clique of a graph ........................ 568
2.9.42 max_flow maximum flow between two nodes ...................... 568
2.9.43 mesh2d triangulation of n points in the plane ..................... 569
2.9.44 metanet opens a Metanet window ................................ 571
2.9.45 metanet_sync asynchronous or synchronous mode in Metanet .......... 572
2.9.46 min_flow minimum linear cost constrained flow .................... 572
2.9.47 min_flow1 minimum linear cost flow .......................... 573
2.9.48 min_flow2 minimum linear cost flow .......................... 574
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.12.9</td>
<td>cdfnor cumulative distribution function normal distribution</td>
<td>642</td>
</tr>
<tr>
<td>2.12.10</td>
<td>cdfpoi cumulative distribution function poisson distribution</td>
<td>643</td>
</tr>
<tr>
<td>2.12.11</td>
<td>cdft cumulative distribution function Student’s T distribution</td>
<td>643</td>
</tr>
<tr>
<td>2.12.12</td>
<td>grand Random number generator</td>
<td>644</td>
</tr>
<tr>
<td>2.13</td>
<td>TCL/Tk interface</td>
<td>648</td>
</tr>
<tr>
<td>2.13.1</td>
<td>ScilabEval tcl instruction : Evaluate a string with scilab interpreter</td>
<td>649</td>
</tr>
<tr>
<td>2.13.2</td>
<td>TK_EvalFile Reads and evaluate a tcl/tk file</td>
<td>649</td>
</tr>
<tr>
<td>2.13.3</td>
<td>TK_EvalStr Evaluate a string within the tcl/tk interpreter</td>
<td>650</td>
</tr>
<tr>
<td>2.13.4</td>
<td>TK_GetVar Get a tcl/tk variable value</td>
<td>651</td>
</tr>
<tr>
<td>2.13.5</td>
<td>TK_SetVar Set a tcl/tk variable value</td>
<td>651</td>
</tr>
<tr>
<td>2.13.6</td>
<td>close close a figure</td>
<td>652</td>
</tr>
<tr>
<td>2.13.7</td>
<td>figure create a figure</td>
<td>652</td>
</tr>
<tr>
<td>2.13.8</td>
<td>findobj find an object with specified property</td>
<td>653</td>
</tr>
<tr>
<td>2.13.9</td>
<td>gcf gets the current active tksci figure</td>
<td>654</td>
</tr>
<tr>
<td>2.13.10</td>
<td>get Retrieve a property value from an User Interface object</td>
<td>654</td>
</tr>
<tr>
<td>2.13.11</td>
<td>set set a property value of a User Interface object</td>
<td>655</td>
</tr>
<tr>
<td>2.13.12</td>
<td>uicontrol create a Graphic User Interface object</td>
<td>655</td>
</tr>
<tr>
<td>2.13.13</td>
<td>uimenu Create a menu or a submenu in a figure</td>
<td>657</td>
</tr>
<tr>
<td>2.14</td>
<td>Language and data translation tools</td>
<td>659</td>
</tr>
<tr>
<td>2.14.1</td>
<td>ascii string ascii conversions</td>
<td>660</td>
</tr>
<tr>
<td>2.14.2</td>
<td>excel2sci reads ascii Excel files</td>
<td>660</td>
</tr>
<tr>
<td>2.14.3</td>
<td>formatman formats all help files in a directory in ascii, text or html</td>
<td>660</td>
</tr>
<tr>
<td>2.14.4</td>
<td>fun2string generates ascii definition of a scilab function</td>
<td>661</td>
</tr>
<tr>
<td>2.14.5</td>
<td>mfile2sci Matlab M file to scilab translation function</td>
<td>661</td>
</tr>
<tr>
<td>2.14.6</td>
<td>mtlb_load load variables from file with matlab4 format</td>
<td>663</td>
</tr>
<tr>
<td>2.14.7</td>
<td>mtlb_save save variables on file with matlab4 format</td>
<td>663</td>
</tr>
<tr>
<td>2.14.8</td>
<td>pol2tex convert polynomial to TeX format</td>
<td>664</td>
</tr>
<tr>
<td>2.14.9</td>
<td>sci2for scilab function to Fortran routine conversion</td>
<td>664</td>
</tr>
<tr>
<td>2.14.10</td>
<td>texprint TeX output of Scilab object</td>
<td>665</td>
</tr>
<tr>
<td>2.14.11</td>
<td>translatepaths translate a set of Matlab M file directories to scilab</td>
<td>665</td>
</tr>
<tr>
<td>2.15</td>
<td>Interprocess communication toolbox</td>
<td>667</td>
</tr>
<tr>
<td>2.15.1</td>
<td>AdCommunications advanced communication toolbox for parallel programming</td>
<td>668</td>
</tr>
<tr>
<td>2.15.2</td>
<td>Example just to test the environment</td>
<td>668</td>
</tr>
<tr>
<td>2.15.3</td>
<td>pvm communications with other applications using Parallel Virtual Machine</td>
<td>668</td>
</tr>
<tr>
<td>2.15.4</td>
<td>pvm_addhosts add hosts to the virtual machine</td>
<td>669</td>
</tr>
<tr>
<td>2.15.5</td>
<td>pvm_bcast brocasts a message to all members of a group</td>
<td>669</td>
</tr>
<tr>
<td>2.15.6</td>
<td>pvm_bufinfo Returns information about a message buffer</td>
<td>670</td>
</tr>
<tr>
<td>2.15.7</td>
<td>pvm_config sends a message</td>
<td>670</td>
</tr>
<tr>
<td>2.15.8</td>
<td>pvm_delhosts deletes hosts from the virtual machine</td>
<td>671</td>
</tr>
<tr>
<td>2.15.9</td>
<td>pvm_error Prints message describing an error returned by a PVM call</td>
<td>671</td>
</tr>
<tr>
<td>2.15.10</td>
<td>pvm_exit tells the local pvmd that this process is leaving PVM</td>
<td>672</td>
</tr>
<tr>
<td>2.15.11</td>
<td>pvm_sci2f77 Convert a F77 complex into a complex scalar</td>
<td>672</td>
</tr>
<tr>
<td>2.15.12</td>
<td>pvm_get_timer Gets the system’s notion of the current time</td>
<td>673</td>
</tr>
<tr>
<td>2.15.13</td>
<td>pvm_getinst returns the instance number in a group of a PVM process</td>
<td>673</td>
</tr>
<tr>
<td>2.15.14</td>
<td>pvm_gsize returns the number of members presently in the named group</td>
<td>674</td>
</tr>
<tr>
<td>2.15.15</td>
<td>pvm_halt stops the PVM daemon</td>
<td>674</td>
</tr>
<tr>
<td>2.15.16</td>
<td>pvm_joiningroup enrolls the calling process in a named group</td>
<td>675</td>
</tr>
<tr>
<td>2.15.17</td>
<td>pvm_kill Terminates a specified PVM process</td>
<td>675</td>
</tr>
<tr>
<td>2.15.18</td>
<td>pvm_lgroup Unenrolls the calling process from a named group</td>
<td>676</td>
</tr>
<tr>
<td>2.15.19</td>
<td>pvm_mytid returns the tid of the calling process</td>
<td>676</td>
</tr>
<tr>
<td>2.15.20</td>
<td>pvm_probe Check if message has arrived</td>
<td>677</td>
</tr>
<tr>
<td>2.15.21</td>
<td>pvm_recv receive a message</td>
<td>677</td>
</tr>
<tr>
<td>2.15.22</td>
<td>pvm_reduce Performs a reduce operation over members of the specified group</td>
<td>678</td>
</tr>
<tr>
<td>2.15.23</td>
<td>pvm_sci2f77 Convert complex scalar into F77</td>
<td>679</td>
</tr>
<tr>
<td>2.15.24</td>
<td>pvm_send immediately sends (or multicast) data</td>
<td>679</td>
</tr>
</tbody>
</table>
2.15.25 pvm_set_timer  Sets the system's notion of the current time. ............ 680
2.15.26 pvm_spawn  Starts new Scilab processes. .............................. 681
2.15.27 pvm_spawnindependent  Starts new PVM processes. .................... 681
2.15.28 pvm_start  Start the PVM daemon .............................. 682
2.15.29 pvm_tidtohost  returns the host of the specified PVM process. ........... 683
2.15.30 pvmd3  PVM daemon ........................................ 683
2.15.31 Communications  communications with other applications using GeCi ........ 686
2.15.32 CreateLink  creates a link between two applications ........................ 686
2.15.33 DestroyLink  destroys the link between two applications .................... 687
2.15.34 ExecAppli  executes an application ................................... 687
2.15.35 ExecScilab  executes another local Scilab .............................. 687
2.15.36 ExeclScilab  executes another linked local Scilab ...................... 688
2.15.37 GetMsg  gets a pending message .................................... 688
2.15.38 SendMsg  sends a message ........................................ 688
2.15.39 WaitMsg  waits for a message ...................................... 689
Chapter 1

Basic functions

1.1 Programming
1.1.1 abort ................................................................................ interrupt evaluation.

DESCRIPTION :
abort interrupts current evaluation and gives the prompt. Within a pause level abort return to level 0 prompt.
SEE ALSO: quit 67, pause 64, break 28, abort 26, quit 67

1.1.2 ans ................................................................................ answer

DESCRIPTION :
ans means “answer”. Variable ans is created automatically when expressions are not assigned. ans contains the last unassigned evaluated expression.

1.1.3 apropos ................................................................. searches keywords in Scilab help

CALLING SEQUENCE :
apropos word
apropos ‘string’

DESCRIPTION :
Looks for keywords in man/*/whatis files.
EXAMPLE :
apropos ‘+’
apropos ode
apropos ‘list of’

SEE ALSO: help 298

1.1.4 backslash .......................................................... - left matrix division.

CALLING SEQUENCE :
x=A\b

DESCRIPTION :
Backslash denotes left matrix division. x=A\b is a solution to A*x=b.
If A is nonsingular x=A\b (uniquely defined) is equivalent to x=inv(A)*b.
If A is singular, x is a least square solution. i.e. norm (A*x-b) is minimal. If A is full column rank, the least square solution, x=A\b, is uniquely defined (there is a unique x which minimizes norm (A*x-b)). If A is not full column rank, then the least square solution is not unique, and x=A\b, in general, is not the solution with minimum norm (the minimum norm solution is x=pinv(A)*b).
A\B is the matrix with (i,j) entry A(i,j)\B(i,j). If A (or B) is a scalar A\B is equivalent to A*ones(B)\B (or A\(B*ones(A))
A\B is an operator with no predefined meaning. It may be used to define a new operator (see overloading) with the same precedence as * or /.
EXAMPLE :
brackets Scilab Function

A=rand(3,2);b=[1;1;1]; x=A\b; y=pinv(A)*b; x-y
A=rand(2,3);b=[1;1]; x=A\b; y=pinv(A)*b; x-y, A*x-b, A*y-b
A=rand(3,1)*rand(1,2); b=[1;1;1]; x=A\b; y=pinv(A)*b; A*x-b, A*y-b
A=rand(2,1)*rand(1,3); b=[1;1]; x=A\b; y=pinv(A)*b; A*x-b, A*y-b

See Also: slash 71, inv 516, pinv 525, percent 64, ieee 50

1.1.5  **bool2s** ___________________ convert boolean matrix to a zero one matrix.

**CALLING SEQUENCE:**

bool2s(x)

**PARAMETERS:**

x : a boolean vector or a boolean matrix or a constant matrix

**DESCRIPTION:**

If x is a boolean matrix, bool2s (x) returns the matrix where "true" values are replaced by 1 and "false" value by 0.
If x is a "standard" matrix, bool2s (x) returns the matrix where non-zero values are replaced by 1.

**EXAMPLE:**

bool2s([%t %t %f %t])
bool2s([2.3 0 10 -1])

See Also: boolean 27, find 41

1.1.6  **boolean** __________ Scilab Objects, boolean variables and operators & | ~

**DESCRIPTION:**

A boolean variable is %T (for "true") or %F (for "false"). These variables can be used to define matrices of booleans, with the usual syntax. Boolean matrices can be manipulated as ordinary matrices for elements extraction/insertion and concatenation. Note that other usual operations (+, *, -, ^, etc) are undefined for booleans matrices, three special operators are defined for boolean matrices:

~b : is the element wise negation of boolean b (matrix).
b1&b2 : is the element wise logical and of b1 and b2 (matrices).
b1|b2 : is the element wise logical or of b1 and b2 (matrices).

Boolean variables can be used for indexing matrices or vectors. For instance a([%T,%F,%T],:) returns the submatrix made of rows 1 and 3 of a. Boolean sparse matrices are supported.

**EXAMPLE:**

[1,2]==[1,3]
[1,2]==1
a=1:5; a(a>2)

See Also: matrices 58, or 205, and 165, not 204

Scilab Group

April 1993

27
1.1.7 brackets ____________________________ - left and right brackets

CALLING SEQUENCE:

```
[a11,a12,...;a21,a22,...;...]
[s1,s2,...]=func(...)
```

PARAMETERS:

- `a11,a12,...`: any matrix (real, polynomial, rational, syslin list ...) with appropriate dimensions
- `s1,s2,...`: any possible variable name

DESCRIPTION:

Left and right brackets are used to note vector and matrix concatenation. These symbols are also used to denote a multiple left-hand-side for a function call.

Inside concatenation brackets, blank or comma characters mean "column concatenation", semicolon and carriage-return mean "row concatenation".

Note: to avoid confusions it is safer to use commas instead of blank to separate columns.

Within multiple lhs brackets variable names must be separated by comma.

EXAMPLES:

```
[6.9,9.64; sqrt(-1) 0]
[1 +%i 2 -%i 3]
[]
['this is';'a string';'vector']
```

```
s=poly(0,'s');[1/s,2/s]
```

```
[u,s]=schur(rand(3,3))
```

SEE ALSO: comma 32, semicolon 70

1.1.8 break ____________________________ keyword to interrupt loops

DESCRIPTION:

Inside a `for` or `while` loop, the command `break` forces the end of the loop.

EXAMPLE:

```
k=0; while 1==1, k=k+1; if k > 100 then break,end; end
```

SEE ALSO: while 77, if 50, for 42, abort 26, return 68

1.1.9 call ____________________________ Fortran or C user routines call

CALLING SEQUENCE:

```
// long form 'out' is present
[y1,...,yk]=call("ident",x1,pxl,"tx1","txn",...
, "out",[ny1,my1],pyl,"ty1",...,[nyl,myl],pyl,"tyl")
```

```
// short form : no 'out' parameter
[y1,...,yk]=call("ident",x1,...,xn)
```
PARAMETERS :
"ident" : string.
xi : real matrix or string
pxi, pyi : integers
txi, tyi : character string "d", "r", "i" or "c".

DESCRIPTION :
Interactive call of Fortran (or C) user program from Scilab. The routine must be previously linked with
Scilab. This link may be done:
- with Scilab "link" command (incremental "soft" linking) during the Scilab session.(see link)
- by "hard" re-linking. Writing the routine call within Scilab routine default/Ex-fort.f, adding
the entry point in the file default/Flist and then re-linking Scilab with the command make
bin/scilex in main Scilab directory.

There are two forms of calling syntax, a short one and a long one. The short one will give faster code
and an easier calling syntax but one has to write a small (C or Fortran) interface in order to make
the short form possible. The long one make it possible to call a Fortran routine (or a C one) whitout
modification of the code but the syntax is more complex and the interpreted code slower.

The meaning of each parameter is described now:

"ident " is the name of the called subroutine.
x1,...,xn are input variables (real matrices or strings) sent to the routine,
p1x1,...,pxn are the respective positions of these variables in the calling sequence of the routine "ident"
and
tx1,...,txn are their types ("r", "i", "d" and "c" for real (float), integer, double precision and
strings)
"out " is a keyword used to separate input variables from output variables. when this key word is present
it assumes that the long form will be used and when it is not present, the short form is used.
[ny1, my1] are the size (# of rows and columns. For 'c' arguments,ml+n1 is the number of charaters
) of output variables and
py1, ... are the positions of output variables (possibly equal to px1 ) in the calling sequence of the
routine. The pyt's integers must be in increasing order.
"ty1", ... are the Fortran types of output variables. The ky first output variables are put in y1,..., yk.

If an output variable coincides with an input variable (i.e. pyi=pxj ) one can pass only its position pyt.
The size and type of yi are then the same as those of xi. If an output variable coincides with an input
variable and one specify the dimensions of the output variable [my1,ny1] must follow the compatibility
condition mxk*nk >= myl*nyl.
In the case of short syntax, [y1,...,yk]=call("ident",x1,...,xn), the input parameters
xi's and the name "ident " are sent to the interface routine Ex-fort. This interface routine is then
very similar to an interface (see the source code in the directory SCIDIR/default/Ex-fort.f).
For example the following program:

subroutine foof(c,a,b,n,m)
integer n,m
double precision a(*),b,c(*)
do 10 i=1,m*n
   c(i) = sin(a(i))+b
10 continue
end

link("foof.o","foof")
a=[1,2,3;4,5,6];b= %pi;

Scilab Group April 1993 29
[m,n]=size(a);
// Inputs:
// a is in position 2 and double
// b 3 double
// n 4 integer
// m 5 integer
// Outputs:
// c is in position 1 and double with size [m,n]
c=call("foof",a,2,"d",b,3,"d",n,4,"i",m,5,"i","out",[m,n],1,"d");

returns the matrix c=2*a+b.

If your machine is a DEC Alpha, SUN Solaris or SGI you may have to change the previous command
line link("foo.o","foo") by one of the followings:

link('foof.o -lfor -lm -lc','foof').
link('foof.o -lftn -lm -lc','foof').
link('foof.o -L/opt/SUNWspro/SC3.0/lib/lib77 -lm -lc','foof').

The same example coded in C:

void fooc(c,a,b,m,n)
double a[],*b,c[];
int *m,*n;
{ double sin();
int i;
for ( i =0 ; i < (*m)*(*n) ; i++)
    c[i] = sin(a[i]) + *b;
}

link("fooc.o","fooc","C") // note the third argument
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
c=call("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");

SEE ALSO:  link 303,  c_link 297,  intersci 303,  addinter 265

1.1.10  case  _______________________________  keyword used in select

DESCRIPTION:
Keyword used in select ... case Use it in the following way:

select expr0,
    case expr1 then instructions1,
    case expr2 then instructions2,
    ...
    case exprn then instructionsn,
    [else instructions],
end

SEE ALSO:  select 69,  while 77,  end 34,  for 42

1.1.11  clear  _______________________________  kills variables

CALLING SEQUENCE:

Scilab Group  April 1993  30
clear a

DESCRIPTION:
This command kills variables which are not protected. It removes the named variables from the environment. By itself clear kills all the variables except the variables protected by predef. Thus the two commands predef (0) and clear remove all the variables.

Normally, protected variables are standard libraries and variables with the percent prefix.

Note the particular syntax clear a and not clear(a). Note also that a=[] does not kill a but sets a to an empty matrix.

SEE ALSO: predef 66, who 77

1.1.12 clear global ___________________________________ kills global variables

CALLING SEQUENCE:
clear global()
clear global nam1 .. namn
clear global ('nam1', .., 'namn')

PARAMETERS:
nam1, ..., namn: valid variable names

DESCRIPTION:
clear global() kills all the global variables.
clear global nam1 .. namn kills the global variables given by their names

Note that clear global() only clears the global variables, the local variables pointing on these global variables are not destroyed.

SEE ALSO: global 46, who 77

EXAMPLE:
global a b c
a=1;b=2;c=3;
who('global')
clear global b
who('global')

1.1.13 colon ___________________________________ - colon operator

DESCRIPTION:
: : Colon. Used in subscripts and loops.

j:k is the vector [j, j+1, ..., k] (empty if J>K).
j:d:k is the vector [j, j+d, ..., j+m*d]

The colon notation can also be used to pick out selected rows, columns and elements of vectors and matrices.

A(:) is the vector of all the elements of A regarded as a single column.
A(:,j) is the j-th column of A
A(:,j:k) is [A(:,j), A(:,j+1), ..., A(:,k)]
A(:,j:k) is [A(:,j), A(:,j+1), ..., A(:,k)]
A(:)=w fills the matrix A with entries of w (taken column by column if w is a matrix).

SEE ALSO: matrix 58
1.1.14 comma ---------------------- - column, instruction, argument separator

DESCRIPTION:
Commats are used to separate parameters in functions or to separate entries of row vectors. Blanks can also be used to separate entries in a row vector but use preferably commas. Also used to separate Scilab instructions. (Use ; to have the result not displayed on the screen).
EXAMPLES:
\[
a=[1,2,3;4,5,6];
a=1,b=1;c=2
\]
SEE ALSO:  semi 70, brackets 28

1.1.15 comments---------------------------------------- comments

DESCRIPTION:
Command lines which begin by // are not interpreted by Scilab. Comments must not begin with //end !

1.1.16 debug ---------------------------------------- debugging level

CALLING SEQUENCE:
debug(level-int)
level-int=debug()

PARAMETERS:
level-int : integer (0 to 4)

DESCRIPTION:
For the values 0,1,2,3,4 of level-int , debug defines various levels of debugging. (For Scilab experts only).

1.1.17 dot ---------------------------------------- - symbol

CALLING SEQUENCE:
\[
123.33
a.*b
[123,..
456]
\]
SH DESCRIPTION
TP 6
.
Dot is used to mark decimal point for numbers : 3.25 and 0.001
TP
.<op>
used in conjunction with other operator symbols (* / \ \ ^ ') to form other operators. Element-by-element multiplicative operations are
obtained using .*, .^, ./, .\ or .’. For example, C = A ./ B is the matrix with elements c(i,j) = a(i,j)/b(i,j). Kronecker product is noted .:*.

Note that when dot follows a number it is always part of the number so 2.*x is evaluated as 2.0*x and 2 .*x is evaluated as (2).*x.

Continuation. Two or more decimal points at the end of a line causes the following line to be a continuation.

**EXAMPLE**

```matlab
x=[1 2 3]; x.^2 .*x // a space is required between 2 and dot
```

**SEE ALSO:** star 72, hat 47, slash 71, backslash 26

### 1.1.18 else

** DESCRIPTION:**

Used with if.

**SEE ALSO:** if 50

### 1.1.19 elseif

** DESCRIPTION:**

See if, then, else.

### 1.1.20 empty

** DESCRIPTION:**

[] denotes the empty matrix. It is uniquely defined and has 0 row and 0 column, i.e. `size([])` = [0,0]. The following convenient conventions are made:

- `[] * A = A * [] = []`
- `[] + A = A + [] = A`
- `[[], A] = [A, []] = A`
- `inv([]) = rcond([]) = cond([]) = 1, rank([]) = 0`

Matrix functions return [] or an error message when there is no obvious answer. Empty linear systems (syslin lists) may have several rows or columns.

**EXAMPLE:**

```matlab
s=poly(0,'s'); A = [s, s+1];
A+[], A*[]
A=rand(2,2); AA=A([],1), size(AA)
svd([])
w=ssrand(2,2,2); wr=[], w1=ss2tf(wr), size(w1)
```

**SEE ALSO:** matrices 58, poly 65, string 284, boolean 27, rational 68, syslin 224
1.1.21 end __________________________________________ end keyword

DESCRIPTION:
Used at end of loops or conditionals. for, while, if, select must be terminated by end.

SEE ALSO: for 42, while 77, if 50, select 69

1.1.22 equal ___________________________ - affectation, comparison equal sign

DESCRIPTION:
Equal sign is used to denote a value affectation to a variable.
== denote equality comparison between two expressions and returns a boolean matrix.

EXAMPLES:

a=sin(3.2)
[u,s]=schur(rand(3,3))
[1:10]==4
1˜=2

SEE ALSO: less 56, boolean 27

1.1.23 errcatch _____________________________ error trapping

CALLING SEQUENCE:

errcatch(n [,,'action'] [,,'option'])

PARAMETERS:

n : integer
action, option : strings

DESCRIPTION:
errcatch gives an "action" (error-handler) to be performed when an error of type n occurs. n has the following meaning:
if n>0, n is the error number to trap
if n<0 all errors are to be trapped

action is one of the following character strings:
"pause" : a pause is executed when trapping the error. This option is useful for debugging purposes. Use whereami() to get informations on the current context.
"continue" : next instruction in the function or exec files is executed, current instruction is ignored.
It is possible to check if an error has occurred using the iserror function. Do not forget to clear the error using the errclear function as soon as possible This option is useful for error recovery. In many cases, usage of errcatch(n,"continue",...) can be replaced by the use of execstr function.
"kill" : default mode, all intermediate functions are killed, scilab goes back to the level 0 prompt.
"stop" : interrupts the current Scilab session (useful when Scilab is called from an external program).

option is the character string 'nomessage' for killing error message.

To set back default mode, enter errcatch(-1,"kill") or similarly errcatch(-1).

Then called in a scilab function the errcatch is automatically reset to the default mode when the function returns.

SEE ALSO: errclear 35, iserror 55, whereami 76, execstr 37
1.1.24 **errclear** error clearing

**CALLING SEQUENCE:**

```
errclear([n])
```

**DESCRIPTION:**

Clears the action (error-handler) connected to error of type `n`. If `n` is positive, it is the number of the cleared error; otherwise all errors are cleared (default case).

**SEE ALSO:** `errcatch 34`, `iserror 55`, `lasterror 55`

1.1.25 **error** error messages

**CALLING SEQUENCE:**

```
error('string' [,n])
error(m)
```

**DESCRIPTION:**

Prints the character string `'string'` in an error message and stops the current instruction. If `n` is given, it is associated to the number of the error. `n` should be larger than `10000` (default value). `error(m)` prints the message associated with the error number `m`.

**SEE ALSO:** `warning 261`

1.1.26 **evstr** evaluation of expressions

**CALLING SEQUENCE:**

```
H=evstr(Z)
[H,ierr]=evstr(Z)
```

**PARAMETERS:**

```
Z : matrix of character strings
M or list(M,Subexp)
M : matrix of character strings
Subexp : vector of character strings
H : matrix
ierr : integer, error indicator
```

**DESCRIPTION:**

Returns the result of the evaluation of the matrix of character strings `M`. Each element of the matrix must define a valid Scilab expression. If the evaluation of `M` expression leads to an error, the single return value version, `H=evstr(M)`, raises the error as usual. The two return values version, `[H,ierr]=evstr(M)`, on the other hand, produces no error, but returns the error number in `ierr`. If `Z` is a list, `Subexp` is a vector of character strings, that defines sub-expressions which are evaluated before evaluating `M`. These sub-expressions must be referred to as `%(k)` in `M`, where `k` is the sub-expression’s index in `Subexp`. `evstr('a=1')` is not valid (use `execstr` instead).

**EXAMPLES:**

```
a=1; b=2; Z=['a','b'] ; evstr(Z)
a=1; b=2; Z=list(['%(1)','%(1)-%(2)'],['a+1','b+1']);
evstr(Z)
```

**SEE ALSO:** `execstr 37`
1.1.27 exec 

script file execution

CALLING SEQUENCE :

exec(path [,mode])
exec(fun [,mode])
ierr=exec(path,‘errcatch’ [,mode])
ierr=exec(fun,‘errcatch’ [,mode])

PARAMETERS :

path  : a string, the path of the script file
mode  : an integer scalar, the execution mode (see below)
fun   : a scilab function
ierr  : integer, 0 or error number

DESCRIPTION :

exec(path [,mode]) executes sequentialy the scilab instructions contained in the file given by path
with an optional execution mode mode .
The different cases for mode  are :

0 : the default value
-1 : nothing is printed
1 : echo of each command line
2 : prompt --> is printed
3 : echoes + prompts
4 : stops before each prompt
7 : stops + prompts + echoes : useful mode for demos.

exec(fun [,mode]) executes function fun as a script: no input nor output argument nor specific
variable environment. This form is more efficient, because script code may be pre-compiled (see getf,
comp). This method for script evaluation allows to store scripts as function in libraries.
If an error is encountered while executing , if ‘errcatch’ flag is present exec issues an error message,
aborts execution of the instructions and resume with ierr equal to the error number,if ‘errcatch’ flag is
not present, standard error handling works.

REMARK :

exec files may now be used to define functions using the inline function definition syntax (see function).

EXAMPLES :

// create a script file
write(TMPDIR+'/'+myscript,’a=1;b=2’)
// execute it
exec(TMPDIR+'/'+myscript')
who

//create a function
deff(’y=foo(x)’,’a=x+1;y=a^2’)clear a b
//execute the function
foo(1)
// a is a variable created in the environment of the function foo
// it is destroyed when foo returns
who

x=1 //create x to make it known by the script foo
exec(foo)
// a and y are created in the current environment
who

SEE ALSO: getf 272, execstr 37, evstr 35, comp 266, mode 59, chdir 297, getcwd 67

1.1.28 execstr ___________________________ execute Scilab code in strings

CALLING SEQUENCE:
execstr(instr)
ierr=execstr(instr,'errcatch' [,msg])

PARAMETERS:
instr : vector of character strings, Scilab instruction to be executed.
ierr : integer, 0 or error number.
msg : character string with values 'm' or 'n'. Default value is 'n'.

DESCRIPTION:
Executes the Scilab instructions given in argument instr.
If the 'errcatch' flag is not present, error handling works as usual.
If the 'errcatch' flag is set, and an error is encountered while executing the instructions defined in instr, execstr issues no error message, but aborts execution of the instr instructions (at the point where the error occurred), and resumes with ierr equal to the error number. In this case the display of the error message is controlled by the msg option:
"m" : error message is displayed and recorded.
"n" : no error message is displayed, but the error message is recorded (see lasterror). This is the default.

EXAMPLE:
execstr('a=1') // sets a=1.
execstr('1+1') // does nothing (while evstr('1+1') returns 2)
execstr(['if %t then;
    a=1;
    b=a+1;
else
    b=0;
end'])

execstr('a=zzzzzzz','errcatch')
execstr('a=zzzzzzz','errcatch','m')

SEE ALSO: evstr 35, lasterror 55, error 35

1.1.29 exists _____________________________ checks variable existence

CALLING SEQUENCE:
exists(name [,where])
PARAMETERS:
name: a character string
where: an optional character string with default value 'all'

DESCRIPTION:
exists(name) returns 1 if the variable named name exists and 0 otherwise.
   Caveats: a function which uses exists may return a result which depends on the environment!
exists(name,'local') returns 1 if the variable named name exists in the local environment
   of the current function and 0 otherwise.

EXAMPLE:
def('foo(x)',...
   ['disp([exists(''a12''),exists(''a12''',''local''))]
   'disp([exists(''x''),exists(''x''',''local''))])
foo(1)
a12=[];foo(1)

SEE ALSO:  isdef 192,  whereis 77, type 74, typeof 229, macrovar 274

1.1.30 exit ____________________________ Ends the current Scilab session

DESCRIPTION:
   Ends the current Scilab session.
SEE ALSO: quit 67, abort 26, break 28, return 68, resume 68

1.1.31 external _____________ Scilab Object, external function or routine

DESCRIPTION:
   External function or routine for use with specific commands.
An "external" is a function or routine which is used as an argument of some high-level primitives (such as
ode, optim, schur...).
The calling sequence of the external (function or routine) is imposed by the high-level primitive which sets
the arguments of the external.
For example the external function costfunc is an argument of the optim primitive. Its calling se-
quence must be: [f,g,ind]=costfunc(x,ind) and optim (the high-level optimization primitive)
is invoked as follows:
optim(costfunc,...)
   Here costfunc (the cost function to be minimized by the primitive optim) evaluates f=f(x) and
g= gradient of f at x (ind is an integer which is not useful here).
If other values are needed by the external function these variables can be defined in the environment. Also,
they can be put in a list. For example, the external function
[f,g,ind]=costfunc(x,ind,a,b,c)
   is valid for optim if the external is list(costfunc,a,b,c) and the call to optim is then:
optim(list(costfunc,a1,b1,c1),...).
   An external can also be a Fortran routine : this is convenient to speed up the computations.
The name of the routine is given to the high-level primitive as a character string. The calling sequence of
the routine is also imposed. Examples are given in the routines/default directory (see the README
file).
External Fortran routines can also be dynamically linked (see link)
SEE ALSO:  ode 431,  optim 438,  impl 421, dassl 416, intg 425, schur 533, gschur 513
1.1.32 extraction

CALLING SEQUENCE:

\[ x(i,j) \]
\[ x(i) \]
\[ \ldots = l(i) \]
\[ \ldots = l(k1) \ldots (kn)(i) \text{ or } \ldots = l(\text{list}(k1,\ldots,\text{kn},i)) \]
\[ l(k1) \ldots (kn)(i,j) \text{ or } l(\text{list}(k1,\ldots,\text{kn},\text{list}(i,j))) \]

PARAMETERS:

- \( x \): matrix of any possible types
- \( l \): list variable
- \( i, j \): indices
- \( k1, \ldots \text{kn} \): indices

DESCRIPTION:

MATRIX CASE: \( i \) and \( j \), can be:

- real scalars or vectors or matrices with positive elements.
  * \( r = x(i,j) \) designs the matrix \( r \) such as \( r(l,k) = x(\text{int}(i(l)), \text{int}(j(k))) \) for \( l \) from 1 to \( \text{size}(i,'*') \) and \( k \) from 1 to \( \text{size}(j,'*') \).
  * \( i (j) \) Maximum value must be less or equal to \( \text{size}(x,1) \) (\( \text{size}(x,2) \)).
  * \( r = x(i) \) with \( x \) a 1x1 matrix designs the matrix \( r \) such as \( r(l,k) = x(\text{int}(i(l)), \text{int}(i(k))) \) for \( l \) from 1 to \( \text{size}(i,1) \) and \( k \) from 1 to \( \text{size}(i,2) \).
  * \( i \) Maximum value must be less or equal to \( \text{size}(x,'*') \).
  - the \( : \) symbol which stands for "all elements".
  * \( r = x(i,:) \) designs the matrix \( r \) such as \( r(l,k) = x(\text{int}(i(l)),k) \) for \( l \) from 1 to \( \text{size}(i,'*') \) and \( k \) from 1 to \( \text{size}(x,2) \).
  * \( r = x(:,j) \) designs the matrix \( r \) such as \( r(l,k) = x(l, \text{int}(j(k))) \) for \( l \) from 1 to \( \text{size}(r,1) \) and \( k \) from 1 to \( \text{size}(j,'*') \).
  * \( r = x(:,:) \) designs the column vector \( r \) formed by the column concatenations of \( x \) columns. It is equivalent to \( \text{matrix}(x, \text{size}(x,'*'),1) \).
  - vector of boolean. If an index \( i \) or \( j \) is a vector of booleans it is interpreted as \( \text{find}(i) \) or respectively \( \text{find}(j) \).
  - a polynomial. If an index \( i \) or \( j \) is a vector of polynomials or implicit polynomial vector it is interpreted as \( \text{horner}(i,m) \) or respectively \( \text{horner}(j,n) \) where \( m \) and \( n \) are associated \( x \) dimensions.

Even if this feature works for all polynomials, it is recommended to use polynomials in \( \$ \) for readability.

LIST OR TLIST CASE: If they are present the \( ki \) give the path to a sub-list entry of \( l \) data structure. They allow a recursive extraction without intermediate copies.

The \( \ldots = l(k1) \ldots (kn)(i) \text{ and } \ldots = l(\text{list}(k1,\ldots,\text{kn},i)) \) instructions are interpreted as:

\( l(k1) = l(k1) \ldots = \ldots l(kn) = l(kn-1(kn) \ldots) = l(kn(i)) \text{ And the } l(k1) \ldots (kn)(i,j) \)
and \( l(\text{list}(k1,\ldots,\text{kn},\text{list}(i,j))) \) instructions are interpreted as:
When path points on more than one list component the instruction must have as many left hand side arguments as selected components. But if the extraction syntax is used within a function input calling sequence each returned list component is added to the function calling sequence.

Note that, \$l(\text{list}())\$ is the same as \$l\$.

- real scalar or vector or matrix with positive elements.
  \[ [r1,...rn]=l(i) \] extracts the \$i(k)\$ elements from the list \$l\$ and store them in \$rk\$ variable for \$k\$ from 1 to \$\text{size}(i,\'**\')\$.
- the \$: \$ symbol which stands for ”all elements”.
- a vector of booleans. If \$i\$ is a vector of booleans it is interpreted as \$\text{find}(i)\$.
- a polynomial. If \$i\$ is a vector of polynomials or implicit polynomial vector it is interpreted as \$\text{horner}(i,m)\$ where \$m=\text{size}(l)\$.

Even if this feature works for all polynomials, it is recommended to use polynomials in \$\$ for readability.

\$k1,..kn\$ may be:
- real positive scalar.
- a polynomial, interpreted as \$\text{horner}(ki,m)\$ where \$m\$ is the corresponding sub-list size.

**REMARKS:**

For soft coded matrix types such as rational functions and state space linear systems, \$x(i)\$ syntax may not be used for vector element extraction due to confusion with list element extraction. \$x(1,j)\$ or \$x(i,1)\$ syntax must be used.

**EXAMPLE:**

- a character string associated with a sub-list entry name. // MATRIX CASE
  \[ a=[1 2 3;4 5 6] \]
  \$a(1,2)\$
  \$a([1 1],2)\$
  \$a(:,1)\$
  \$a(:,3:-1:1)\$
  \$a(1)\$
  \$a(6)\$
  \$a(:)\$
  \$a([\%\text{t} \%\text{f} \%\text{f} \%\text{t}])\$
  \$a([\%\text{t} \%\text{f}], [2 3])\$
  \$a(1:2,\%\text{s}-1)\$
  \$a(\%\text{s}:-1:1,2)\$
  \$a(\%\text{s})\$

  //
  \$x=\text{\'test\'}\$
  \$x([1 1;1 1;1 1])\$

  //
  \$b=[1/\%s,(\%s+1)/(\%s-1)]\$
  \$b(1,\text{1})\$
  \$b(1,\%\text{s})\$
  \$b(2)\$ // the numerator

  // LIST OR TLIST CASE
  \$l=\text{list}(1,\'qwerw',\%s)\$
  \$l(1)\$
  \$[a,b]=l([3 2])\$
  \$l(\%\text{s})\$
  \$x=\text{tlist}(l(2:3))\$ // form a tlist with the last 2 components of \$l\$
find Scilab Function

1.1.33  **feval**  multiple evaluation

**CALLING SEQUENCE:**

\[ \text{[z]} = \text{feval}(x, y, f) \]

**PARAMETERS:**

\(x, y\) : two vectors

\(f\) : function or character string (for Fortran call)

**DESCRIPTION:**

Multiple evaluation of a function for one or two arguments of vector type:

\[ z = \text{feval}(x, f) \]

returns the vector \(z\) defined by \(z(i) = f(x(i))\)

\[ z = \text{feval}(x, y, f) \]

returns the matrix \(z\), \(z(i, j) = f(x(i), y(j))\)

\(f\) is an external (function or routine) depending on one or two arguments which are supposed to be real. The result returned by \(f\) can be real or complex. In case of a Fortran call, the function ‘\(f\)’ must be defined in the subroutine \(ffeval.f\) (in directory \(SCIDIR/routines/default\)).

**EXAMPLE:**

```matlab
deff('[z]=f(x,y)','z=x^2+y^2');
feval(1:10,1:5,f)
```

**SEE ALSO:**  find 41, horner 490, parents 63

1.1.34  **find**  find indices of boolean vector or matrix true elements

**CALLING SEQUENCE:**

\[ \text{[ii]} = \text{find}(x) \]

\[ \text{[ir, ic]} = \text{find}(x) \]

**PARAMETERS:**

\(x\) : a boolean vector or a boolean matrix or a "standard" matrix

\(ii, ir, ic\) : integer vectors of indices or empty matrices
DESCRIPTION:
If \( x \) is a boolean matrix,
\[ ii = \text{find}(x) \]
returns the vector of indices \( i \) for which \( x(i) \) is "true". If no true element found \text{find} \ returns an empty matrix.
\[ [ir,ic]=\text{find}(x) \]
returns two vectors of indices \( ir \) (for rows) and \( ic \) (for columns) such that \( x(ir(n),ic(n)) \) is "true". If no true element found \text{find} \ returns empty matrices in \( ir \) and \( ic \).
if \( x \) is standard matrix \text{find}(x) \ is interpreted as \text{find}(x<>0)
\text{find}([]) \ returns []

EXAMPLE:
\[
\begin{align*}
A &= \text{rand}(1,20); \\
w &= \text{find}(A<0.5); \\
A(w) \\
w &= \text{find}(A>100);
\end{align*}
\]

SEE ALSO: boolean 27, extraction 39, insertion 51

1.1.35 for language keyword for loops

DESCRIPTION:
Used to define loops. Its syntax is:
\[
\begin{align*}
\text{for variable}=\text{expression} & , \text{instruction}, \cdots, \text{instruction}, \text{end} \\
\text{for variable}=\text{expression} & \text{do instruction, } \cdots, \text{instruction}, \text{end}
\end{align*}
\]
If \( \text{expression} \) is a matrix or a row vector, \text{variable} takes as values the values of each column of the matrix.
Useful example: \( \text{for variable}=n1:\text{step:}n2, \ldots, \text{end} \)
If \( \text{expression} \) is a list \text{variable} takes as values the successive entries of the list.
Warning: the number of characters used to define the body of any conditionnal instruction (if while for or select/case) must be limited to 16k.

EXAMPLE:
\[
\begin{align*}
n &= 5; \\
\text{for } i = 1:n, \text{for } j = 1:n, a(i,j) &= 1/(i+j-1); \text{end}; \text{end} \\
\text{for } j = 2:n-1, a(j,j) &= j; \text{end}; a \\
\text{for } e=\text{eye}(3,3), e, \text{end} \\
\text{for } v=a, \text{write}(6,v), \text{end} \\
\text{for } j=1:n, v=a(:,j), \text{write}(6,v), \text{end} \\
\text{for } l=\text{list}(1,2,'\text{example'}); l, \text{end}
\end{align*}
\]

1.1.36 format number printing and display format

CALLING SEQUENCE:
\[
\text{format(
[\text{type}],\text{[long]})}
\]

PARAMETERS:
\[
\begin{align*}
\text{type} & : \text{character string} \\
\text{long} & : \text{integer (max number of digits (default 10))}
\end{align*}
\]

DESCRIPTION:
Sets the current printing format with the parameter \text{type}; it is one of the following:
"v" : for a variable format (default)
"e" : for the e-format.

long defines the max number of digits (default 10). format() returns a vector for the current format:
first component is the type of format (0 if v ; 1 if e ); second component is the number of digits.
In "variable format" mode, vectors entries which are less than %eps times the maximum absolute value of
the entries were displayed as "0" in the previous Scilab versions. It is no more the case, the clean function
can be used to set neglitible entries to zeros.

EXAMPLE :

x=rand(1,5);
format('v',10);x
format(20);x
format('e',10);x
format(20);x

x=[100 %eps];
format('e',10);x
format('v',10);x
format()

SEE ALSO: write 262, disp 233, print 251

1.1.37 fort ------------------------------------ Fortran or C user routines call

CALLING SEQUENCE :

// long form 'out' is present
[y1,...,yk]=fort("ident",xi,pxi,"txi"...,xn,pxn,"txn",
    "out",[ny1,my1],py1,"ty1"...,[nyl,myl],pyl,"tyl")

// short form : no 'out' parameter
[y1,...,yk]=fort("ident",x1,...,xn)

PARAMETERS :
"ident" : string.
xi : real matrix or string
pxi, pyi : integers
txi, tyi : character string "d", "r", "i" or "c".

DESCRIPTION :
Interactive call of Fortran (or C) user program from Scilab. The routine must be previously linked with
Scilab. This link may be done:

- with Scilab "link" command (incremental "soft" linking) during the Scilab session.(see link)
- by "hard" re-linking. Writing the routine call within Scilab routine default/Ex-fort.f, adding
  the entry point in the file default/Flist and then re-linking Scilab with the command make
  bin/scilex in main Scilab directory.

There are two forms of calling syntax, a short one and a long one. The short one will give faster code
and an easier calling syntax but one has to write a small (C or Fortran) interface in order to make
the short form possible. The long one make it possible to call a Fortran routine (or a C one) whitout
modification of the code but the syntax is more complex and the interpreted code slower.

The meaning of each parameter is described now:
"ident" is the name of the called subroutine.
x1, ..., xn are input variables (real matrices or strings) sent to the routine,
px1, ..., pxn are the respective positions of these variables in the calling sequence of the routine "ident" and
tx1, ...,txn are their types ("r", "i", "d" and "c" for real (float), integer, double precision and strings)
"out" is a keyword used to separate input variables from output variables. when this key word is present it is assumed that the long form will be used and when it is not present, the short form is used.
[ny1, my1] are the size (# of rows and columns. For 'c' arguments,m1*n1 is the number of charaters ) of output variables and
py1, ... are the positions of output variables (possibly equal to pxi ) in the calling sequence of the routine. The pyi's integers must be in increasing order.
"ty1", ... are the Fortran types of output variables. The k first output variables are put in y1,...,
yk.

If an output variable coincides with an input variable (i.e. pyi=pxj ) one can pass only its position pyi. The size and type of yi are then the same as those of xi. If an output variable coincides with an input variable and one specify the dimensions of the output variable [my1,ny1] must follow the compatibility condition mxk*nxk >= my1*ny1.
In the case of short syntax, [y1, ..., yk]=fort("ident",x1,...,xn), the input parameters xi's and the name "ident" are sent to the interface routine Ex-fort. This interface routine is then very similar to an interface (see the source code in the directory SCIDIR/default/Ex-fort.f).
For example the following program:

```fortran
subroutine foof(c,a,b,n,m)
   integer n,m
   double precision a(*),b,c(*)
   do 10 i=1,m*n
      c(i) = sin(a(i))+b
   10 continue
end

link("foof.o","foof")
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
// Inputs:
// a is in position 2 and double
// b 3 double
// n 4 integer
// m 5 integer
// Outputs:
// c is in position 1 and double with size [m,n]
c=fort("foof",a,2,"d",b,3,"d",n,4,"i",m,5,"i","out",[m,n],1,"d");
returns the matrix c=2*a+b.
```

If your machine is a DEC Alpha, SUN Solaris or SGI you may have to change the previous command line link("foo.o","foo") by one of the followings:

```bash
link('foof.o -lfor -lm -lc','foof').
link('foof.o -lftn -lm -lc','foof').
link('foof.o -L/opt/SUNWspro/SC3.0/lib/lib77 -lm -lc','foof').
```

The same example coded in C:

```c
void fooc(c,a,b,m,n)
double a[],*b,c[];
int *m,*n;
```
( double sin();
int i;
for ( i =0 ; i < (*m)*(*n) ; i++)
    c[i] = sin(a[i]) + *b;
)

link("fooc.o","fooc","C") // note the third argument
a=[1,2,3;4,5,6];b= %pi;
[m,n]=size(a);
c=fort("fooc",a,2,"d",b,3,"d",m,4,"i",n,5,"i","out",[m,n],1,"d");

SEE ALSO: link 303, c_link 297, intersci 303, addinter 265

1.1.38 funptr ____________________________ coding of primitives ( wizard stuff )

DESCRIPTION:
Utility function (for experts only) funptr('name') returns 100*fun + fin where (fun,fin) is the internal coding of the primitive 'name'. fun is the interface number and fin the routine number.

SEE ALSO: clearfun 265, newfun 274

1.1.39 getenv ____________________________ get the value of an environment variable

CALLING SEQUENCE:
env=getenv(str [, rep] )

PARAMETERS:
str : character string specifying environment variable name rep : an optional character string. When this optional value is used, the function getenv returns the value rep when the environment variable str is not found.
env : character string which contain the environment variable value

DESCRIPTION:
Return the value of an environment variable if it exists.

EXAMPLE:
getenv('SCI')
getenv('FOO','foo')

1.1.40 getfield ____________________________ list field extraction

CALLING SEQUENCE:
[x,...]=getfield(i,l)

PARAMETERS:
x : matrix of any possible types
l : list, tlist or mlist variable
i : field index, see extraction for more details.
**DESCRIPTION:**
This function is an equivalent of \([x, ...] = l(i)\) syntax for field extraction with the only difference that it also applies to \(m\text{list}\) objects.

**EXAMPLE:**

```plaintext
l=list(1,'qwerw',%s)
[a,b]=getfield([3 2],l)
```

\(a=\text{hypermat}([2,2,2],\text{rand}(1:2^3));\) // hypermatrices are coded using \(m\text{lists}\)

\(a(1)\) // the \(a(1,1,1)\) entry

```plaintext
getfield(1,a) // the mlist first field
```

**SEE ALSO:** extraction 39

### 1.1.41 getpid

**CALLING SEQUENCE:**

```plaintext
id=getpid()
```

**DESCRIPTION:**
Return an the scilab process identificator integer

**EXAMPLE:**

```plaintext
d='SD_'+\text{string}(\text{getpid}())+'__'
```

### 1.1.42 getversion

**CALLING SEQUENCE:**

```plaintext
version=getversion()
```

**PARAMETERS:**

version : a string

**DESCRIPTION:**
return in \(version\) the Scilab version name

**EXAMPLE:**

```plaintext
getversion()
```

### 1.1.43 global

**CALLING SEQUENCE:**

```plaintext
global('nam1',...,'namn')
global \(\text{nam1 ... namn}\)
```

**PARAMETERS:**

\(n\text{am1},..., \text{namn}:\) valid variable names

Scilab Group Mai 1999 46
**DESCRIPTION:**
Ordinarily, each Scilab function has its own local variables and can "read" all variables created in the base workspace or by the calling functions. The `global` allow to make variables read/write across functions. Any assignment to that variable, in any function, is available to all the other functions declaring it `global`.

If the global variable doesn’t exist the first time you issue the `global` statement, it will be initialized to the empty matrix.

**EXAMPLE:**

```plaintext
//first: calling environment and a function share a variable
global a
a=1
deff('y=f1(x)', 'global a,a=x^2,y=a^2')
f1(2)
a
//second: three functions share variables
deff('initdata()', 'global A C ;A=10,C=30')
deff('letsgo()', 'global A C ;disp(A) ;C=70')
deff('letsgo1()', 'global C ;disp(C)')
initdata()
letsgo()
letsgo1()
```

**SEE ALSO:** `who`, `ismglobal`, `clearglobal`, `gstacksize`, `resume`

1.1.44 **gstacksize**

**CALLING SEQUENCE:**

```plaintext
gstacksize(n)
sz=gstacksize()
```

**PARAMETERS:**

- `n`: integer, the required global stack size given in number of double precision words
- `sz`: 2-vector [total used]

**DESCRIPTION:**
Scilab stores global variables in a stack.

`gstacksize(n)` allows the user to increase or decrease the size of this stack. The maximum allowed size depends on the amount of free memory and swap space available at the time. Note that Scilab can increase automatically the global stacksize when needed.

`sz=gstacksize()` returns a 2-vector which contains the current total and used global stack size.

**SEE ALSO:** `who`, `stacksize`  

1.1.45 **hat**

**CALLING SEQUENCE:**

```plaintext
A^b
```
DESCRIPTION:
Exponentiation of matrices or vectors by a constant vector.
If $A$ is a vector or a rectangular matrix the exponentiation is done element-wise, with the usual meaning.
For square $A$ matrices the exponentiation is done in the matrix sense.
For boolean, polynomial and rational matrices, the exponent must be an integer.
Remark that $123.\hat{b}$ is interpreted as $(123).\hat{b}$. In such cases dot is part of the operator, not of the number.

EXAMPLES:

$2^4$
$[1 2; 2 4]^{(1+\%i)}$
$s=poly(0,"s");$
$[1 2 s]^4$
$[s 1; 1 s]^{-1}$

SEE ALSO: exp 508, inv 516

1.1.46 host ____________________________ shell (sh) command execution

CALLING SEQUENCE:

$\text{stat} = \text{host}(\text{command-name})$

PARAMETERS:

command-name : A character string containing Unix sh instruction
stat : An integer flag

DESCRIPTION:
Sends a string command-name to Unix for execution by the sh shell. Standard output and standard errors of the shell command are written in the calling shell. stat gives -1 if host can’t be called (Not enough system memory available) or the sh return code.

EXAMPLE:

host("ls $SCI/demos");
host("emacs $SCI/demos/wheel2/Makefile");
def('wd=getdir()','if MSDOS then host(''cd>''+TMPDIR+''\path'');.. else host(''pwd>''+TMPDIR+''/path'');end..
wd=read(TMPDIR+''/path''+1,1,''(a)''+1)
wd=getdir()

SEE ALSO: edit 267, manedit 239, unix_g 310, unix_s 311, unix_w 311, unix_x 312

1.1.47 hypermat ___________________________ initialize an N dimensional matrices

CALLING SEQUENCE:

$M=\text{hypermat}(\text{dims} [,v])$

PARAMETERS:

dims : vector of hypermatrix dimensions
v : vector of hypermatrix entries (default value zeros(prod(dims),1))
**DESCRIPTION:**
Initialize an hypermatrix whose dimensions are given in the vector dims and entries are given in optional argument v. M data structure contains the vector of matrix dimensions M('dims') and the vector of entries M('entries') such as the leftmost subscripts vary first [M(1,1,..);..;M(n1,1,..);...;M(1,n2,..);..;M(n1,n2,..)].

**EXAMPLES:**
M=hypermat([2 3 2 2],1:24)

1.1.48 hypermatrices ________ Scilab object, N dimensional matrices in Scilab

**DESCRIPTION:**
Hypermatrix type allows to manipulate multidimensional arrays.

They can be defined by extension of 2D matrices as follows a=[1 2;3 4];a(:,:,2)=rand(2,2)
or directly using hypermat function

Entries can be real or complex numbers, polynomials, rationals, strings, booleans.

Hypermatices are mlists: mlist(['hm','dims','entries'],sz,v) where sz is the row vector of dimensions and v the column vector of entries (first dimension are stored first).

**EXAMPLES:**
a(1,1,1,1:2)=[1 2]
a=[1 2;3 4];a(:,:,2)=rand(2,2)
a(1,1,:)
[a a]

SEE ALSO: hypermat 48

1.1.49 iconvert ____________ conversion to 1 or 4 byte integer representation

**CALLING SEQUENCE:**
y=iconvert(X,itype)

**PARAMETERS:**

X : matrix of floats or integers
y : matrix of integers coded on one, two or four bytes.

**DESCRIPTION:**
converts and stores data two one, two or four bytes integers.

itype=0 : return floating point numbers
itype=1 : return int8 numbers in the range [-128,127]
itype=11 : return uint8 numbers in the range [0,255]
itype=2 : return int16 numbers in the range [-32768,32767]
itype=12 : return uint16 numbers in the range [0, 65535]
itype=4 : return int32 numbers in the range [-2147483648,2147483647]
itype=14 : return uint32 numbers in the range [0, 4294967295]

**EXAMPLE:**
b=int32([1 -120 127 312])
y=iconvert(b,8)

SEE ALSO: double 182, inttype 54
1.1.50  ieee __________________ set floating point exception mode

CALLING SEQUENCE :

mod=ieee()
ieee(mod)

PARAMETERS :

mod : integer scalar whose possible values are 0,1,or 2

DESCRIPTION :

ieee() returns the current floating point exception mode.

0 : floating point exception produce an error
1 : floating point exception produce a warning
2 : floating point exception produces Inf or Nan

ieee(mod) sets the current floating point exception mode.

The initial mode value is 0.

REMARKS :

Floating point exception arizing inside some library algorithms are not yet handled by ieee modes.

EXAMPLE :

ieee(1);1/0
ieee(2);1/0,log(0)

SEE ALSO:  errcatch 34

1.1.51  if ____________________________ else - conditional execution

SYNTAX :

if expr1 then statements
elseif expri then statements
....
else statements
end

DESCRIPTION :

The if statement evaluates a logical expression and executes a group of statements when the expression is true.

The expri are expressions with numeric or boolean values. If expri are matrix valued the condition is true only if all matrix entries are true.

The optional elseif and else provide for the execution of alternate groups of statements. An end keyword, which matches the if, terminates the last group of statements. The line structure given above is not significant, the only constraint is that each then keyword must be on the same line line as its corresponding if elseif keyword.

The keyword then can be replaced by a carriage return or a comma.

Warning: the number of characters used to define the body of any conditionnal instruction (if while for or select/case) must be limited to 16k.

EXAMPLE :
i=2
for j = 1:3,
    if i == j then
        a(i,j) = 2;
    elseif abs(i-j) == 1 then
        a(i,j) = -1;
    else a(i,j) = 0;
end,
end

SEE ALSO: while 77, select 69, boolean 27, end 34, then 73, else 33

1.1.52 insertion _______________ matrix and list insertion or modification

CALLING SEQUENCE :

x(i,j)=a
x(i)=a
l(i)=a
l(k1)...(kn)(i)=a or l(list(k1,...,kn,i))=a
l(k1)...(kn)(i,j)=a or l(list(k1,...,kn,list(i,j))=a

PARAMETERS :

x : matrix of any kind (constant, sparse, polynomial,...)
l : list
i,j : indices
k1,...kn : indices with integer value
a : new entry value

DESCRIPTION :

MATRIX CASE i and j, may be:
- real scalars or vectors or matrices with positive elements.
  * if a is a matrix with dimensions (size(i,"*"),size(j,"*")) x(i,j)=a returns a new x
    matrix such as x(int(i(l)),int(j(k)))=a(l,k) for l from 1 to size(i,"*") and k
    from 1 to size(j,"*"), other initial entries of x are unchanged.
  if a is a scalar x(i,j)=a returns a new x matrix such as x(int(i(l)),int(j(k)))=a for l
    from 1 to size(i,"*") and k from 1 to size(j,"*"), other initial entries of x are unchanged.
  If i or j maximum value exceed corresponding x matrix dimension x is previously extended to the
  required dimensions with zeros entries for standard matrices, 0 length character string for string matrices
  and false values for boolean matrices.

  * x(i,j)=[] kills rows specified by i if j matches all columns of x or kills columns specified by j
  if i matches all rows of x. In other cases x(i,j)=[] produce an error.
  * x(i)=a with a a vector returns a new x matrix such as x(int(i(l)))=a(l) for l from 1 to size(i,"*"),
    other initial entries of x are unchanged.

x(i)=a with a a scalar returns a new x matrix such as x(int(i(l)))=a for l from 1 to size(i,"*")
, other initial entries of x are unchanged.

If i maximum value exceed size(x,1), x is previously extended to the required dimension with zeros
entries for standard matrices, 0 length character string for string matrices and false values for boolean
matrices.
if \( x \) is a 1x1 matrix \( a \) may be a row (respectively a column) vector with dimension \( \text{size}(i,'*') \).

Resulting \( x \) matrix is a row (respectively a column) vector

if \( x \) is a row vector \( a \) must be a row vector with dimension \( \text{size}(i,'*') \)

if \( x \) is a column vector \( a \) must be a column vector with dimension \( \text{size}(i,'*') \)

if \( x \) is a general matrix \( a \) must be a row or column vector with dimension \( \text{size}(i,'*') \) and \( i \)

maximum value cannot exceed \( \text{size}(x,'*') \).

\* \( x(i)=[] \) kills entries specified by \( i \).

- the : symbol which stands for "all elements".

\* \( x(i,:)=a \) is interpreted as \( x(i,1:\text{size}(x,2))=a \)

\* \( x(:,j)=a \) is interpreted as \( x(1:\text{size}(x,1),j)=a \)

\* \( x(:,:,)=a \) returns in \( x \) the \( a \) matrix reshaped according to \( x \) dimensions. \( \text{size}(x,'*') \) must be

equal to \( \text{size}(a,'*') \)

- vector of boolean. If an index \( (i \text{ or } j) \) is a vector of booleans it is interpreted as \( \text{find}(i) \) or respectively \( \text{find}(j) \)

- a polynomial. If an index \( (i \text{ or } j) \) is a vector of polynomials or implicit polynomial vector it is interpreted as \( \text{horner}(i,m) \) or respectively \( \text{horner}(j,n) \) where \( m \) and \( n \) are associated \( x \) dimensions.

Even if this feature works for all polynomials, it is recommended to use polynomials in \$ \) for readability.

LIST OR TLIST CASE If they are present the \( k_i \) give the path to a sub-list entry of \( l \) data structure. They allow a recursive extraction without intermediate copies.

The \( l(k_1)\ldots(k_n)(i)=a \) and \( l(\text{list}(k_1,\ldots,k_n,i)=a) \) instructions are interpreted as:

\[
\begin{align*}
\text{l(k1) .. } & = \ldots \text{lkn} = \text{lkn-1(kn)} & \text{lkn(i)} = a \text{lkn-1(kn)} = \text{lkn} \ldots \\
& = \ldots \text{l(k1)} & \text{lkn(i, j)} = a \text{lkn-1(kn)} = \text{lkn} \ldots \\
& = \ldots \text{l(k1)} & \text{lkn(i, j)} = a \text{lkn-1(kn)} = \text{lkn} \ldots \\
& = \ldots \text{l(k1)} & \text{lkn(i, j)} = a \text{lkn-1(kn)} = \text{lkn} \ldots
\end{align*}
\]

\( i \) may be real non negative scalar. \( l(0)=a \) adds an entry on the "left" of the list

\( l(0)=a \) sets the \( i \) entry of the list \( l \) to \( a \). if \( i>\text{size}(l) \), \( l \) is previously extended with zero length entries (undefined).

\( l(i)=null() \) suppress the \( i \)th list entry.

- a polynomial. If \( i \) is a polynomial it is interpreted as \( \text{horner}(i,m) \) where \( m=\text{size}(l) \).

Even if this feature works for all polynomials, it is recommended to use polynomials in \$ \) for readability.

\( k_1,\ldots,k_n \) may be:

- real positive scalar.
- a polynomial, interpreted as \( \text{horner}(k_i,m) \) where \( m \) is the corresponding sub-list size.

REMARKS:

For soft coded matrix types such as rational functions and state space linear systems, \( x(i) \) syntax may not be used for vector entry insertion due to confusion with list entry insertion. \( x(1,j) \) or \( x(i,1) \) syntax must be used.

EXAMPLE:

- a character string associated with a sub-list entry name. // MATRIX CASE
  
a=[1 2 3;4 5 6]
  \a(1,2)=10
  \a([1 1],2)=[-1;-2]
  \a(:,1)=[8;5]
  \a(1,3:-1:1)=[77 44 99]

Scilab Group September 1996 52
a(1)=%s
a(6)=%s+1
a(:,)=1:6
a([%t %f],1)=33
a(1:2,:-1)=[2;4]
a(:,1:1,1)=[8;7]
a($) = 123
```
//
x='test'
x([4 5])=['4','5']
```
//
b=[1/%s, (%s+1)/(%s-1)]
b(1,1)=0
b(1,$)=b(1,$)+1
b(2)=[1 2] // the numerator
// LIST OR TLIST CASE
l=list(1,'qwerw',%s)
l(1)='Changed'
l(0)='Added'
l(6)=['one more';'added']
//
//
dts=list(1,tlist(['x';'a';'b'],10,[2 3]));
dts(2)('a')=33
dts(2)('b')(1,2)=-100
```
SEE ALSO: find 41, horner 490, parents 63, extraction 39

1.1.53 intppty __________________________ set interface argument passing properties

CALLING SEQUENCE :

```
funs=intppty()
intppty(fun)
```

PARAMETERS :

```
fun : integer an interface number (see funptr)
funs : integer vector, vector of interface number (see funptr)
```

DESCRIPTION :

The interface programs may be written in 2 different ways for the mode of function argument passing. In the first and default way, the arguments are passed by value. With the following syntax:

```
foo(A,1+2)
```

the argument associated with A will be passed by value (a copy of A is made before foo is called, and the argument associated with 1+2 will be passed by value.

In the second way arguments may be passed be reference if there are "named arguments" (no copy of the variable value is done). intppty(fun) with fun>0 tells Scilab that the interface with number fun can handle arguments passed by reference. With the following syntax:

```
foo(A,1+2)
```
the argument associated with A will be passed by reference, and the argument associated with 1+2 will be passed by value.

Warning, declaring that the interface with number fun can handle arguments passed by reference if it is not the case should produce unpredictable results.

intppty (fun) with fun<0 suppress this property for the interface -fun.

intppty () returns the vector of interfaces which handle arguments passed by reference.

This function may be useful for dynamically loaded interface (see addinter).

SEE ALSO: funptr 45, addinter 265

1.1.54 inttype ________________ type integers used in integer data types

CALLING SEQUENCE:

[i]=inttype (x)

PARAMETERS:

x : an matrix of integers (see int8,..)
i : integer

DESCRIPTION:

inttype (x) returns an integer which is the type of the entries of x as following:

1 : one byte integer representation
2 : two bytes integer representation
4 : four bytes integer representation
11 : one byte unsigned integer representation
12 : two bytes unsigned integer representation
14 : four bytes unsigned integer representation

EXAMPLE:

x=uint16 (1:10);
inttype (x)

SEE ALSO: int8 188

1.1.55 inv_coeff _____________ build a polynomial matrix from its coefficients

CALLING SEQUENCE:

[P]=inv_coeff (C,[],d,[name])

PARAMETERS:

C : big matrix of the coefficients
d : Polynomial matrix degree. optional parameter with default value d=-1+size(C,'c')/size(C,'r')
name : string giving the polynomial variable name (default value 'x').

DESCRIPTION:

P=inv_coeff (Mp,k). When k is compatible with Mp size it returns a polynomial matrix of degree k.
C=[C0,C1,...,Ck] and P= C0 + C1*x +... +Ck*x^k.

EXAMPLE:
A=int(10*rand(2,6))
// Building a degree 1 polynomial matrix
P=inv_coeff(A,1)
norm(coeff(P)-A)
// Using default value for degree
P1=inv_coeff(A)
norm(coeff(P1)-A)

SEE ALSO: poly 65, degree 486, coeff 485

1.1.56 iserror .............................. error test

CALLING SEQUENCE:

iserror([n])

DESCRIPTION:
tests if error number n has occurred (after a call to errcatch). iserror returns 1 if the error occurred and 0 otherwise
n>0 is the error number; all errors are tested with n<0.

SEE ALSO: error 35, errcatch 34

1.1.57 isglobal .............................. check if a variable is global

CALLING SEQUENCE:

t=isglobal(x)

PARAMETERS:

x : any variable
t : a boolean

DESCRIPTION:
isglobal(x) returns true if x has been declared to be a global variable and false otherwise.

EXAMPLE:

isglobal(1)
global a
isglobal(a)

SEE ALSO: global 46, clearglobal 31, who 77

1.1.58 lasterror .............................. get last recorded error message

CALLING SEQUENCE:

str=lasterror([opt])
[str,n]=lasterror([opt])

PARAMETERS:
### lasterror Function

**Description:**
Each time an error occurs, the Scilab error handler records it in internal tables (only the last one is retained). The `lasterror` function allows to get the message and the error number out of these tables. This function is useful while using `errcatch` or `execstr`. The recorded error message may be retained for a further call to `lasterror` using `lasterror(%f)`.

**Example:**

```plaintext
 ierr=execstr('a=zzzzzzz','errcatch')
 if ierr>0 then disp(lasterror()),end
```

**See Also:** `errcatch`, `execstr`, `error`, `errclear`

### Calling Sequence

```
[a11,a12,...;a21,a22,...;...]
[s1,s2,...]=func(...)```

**Parameters:**

- `a11,a12,...`: matrix of any compatible types with compatible dimensions
- `s1,s2,...`: any possible variable names

**Description:**
Left and right brackets are used for vector and matrix concatenation. These symbols are also used to denote a multiple left-hand-side for a function call. Inside concatenation brackets blank or comma characters mean "column concatenation", semicolon and carriage-return mean "row concatenation".

**Examples:**

- `[6.9,9.64; sqrt(-1) 0]`
- `[1 +%i 2 -%i 3]`
- `['this is';'a string';'vector']`
- `[u,s]=schur(rand(3,3))`

### less Operator

**Description:**
Logical comparison symbol.

- `< >` means "different" (same as `~=`)
- `<` means "lower than"
- `>` means "larger than"
- `<=` means lower than or equal to.
- `>=` means larger than or equal to

**See Also:** `if`
1.1.61 list __________________ Scilab object and list function definition

CALLING SEQUENCE:

list(a1,....an)

DESCRIPTION:

Creates a list with elements a1's which are arbitrary Scilab objects (matrix, list,...). Type of list objects is 15.

list() is the empty list (0 element).

Operations on lists:

extraction : [x,y,z...] = l(v) where v is a vector of indices; [x,y,z] = l(:) extracts all the elements.

insertion : l(i) = a

deletion : l(i) = null() removes the i-th element of the list l.

EXAMPLE:

x = list(1,2,3);
x(4) = 10;
x(10) = 'a'

SEE ALSO: null 61, tlist 73, insertion 51, extraction 39, size 211, length 281

1.1.62 lsslist __________________ Scilab linear state space function definition

CALLING SEQUENCE:

lsslist()
lsslist(a1,....an)

DESCRIPTION:

lsslist(a1,....an) is a shortcut to tlist(['lss','A';'B';'C';'X0','dt'], a1,....an)

Creates a list with ['lss','A';'B';'C';'X0','dt'] as first entry and a1's as next entries if any. No type nor size checking is done on a1's.

SEE ALSO: tlist 73, syslin 224

1.1.63 lstcat _________________________________ list concatenation

CALLING SEQUENCE:

lc = lstcat(l1,..ln)

PARAMETERS:

l1 : list or any other type of variable
lc : a list

DESCRIPTION:

lc = lstcat(l1,..ln) concatenates components of l1 lists in a single list. If l1 are other type of variables they are simply added to the resulting list.

EXAMPLE:

lstcat(list(1,2,3),33,list('foo',%s))
lstcat(1,2,3)

SEE ALSO: list 57
1.1.64  matrices  Scilab object, matrices in Scilab

DESCRIPTION :
Matrices are basic objects defined in Scilab. They can be defined as follows:

\[
E = \begin{bmatrix}
e_{11}, e_{12}, \ldots, e_{1n}; \\
e_{21}, e_{22}, \ldots, e_{2n}; \\
\vdots \\
e_{m1}, e_{m2}, \ldots, e_{mn};
\end{bmatrix}
\]

Entries \( e_{ij} \) can be real or complex numbers, polynomials, rationals, strings, booleans. Vectors are seen as matrices with one row or one column.
syslin lists in state-space form or transfer matrices can also be defined as above.

EXAMPLES :

\[
E = \begin{bmatrix} 1, 2; 3, 4 \end{bmatrix}
\]
\[
E = [%T, %F; 1==1, 1\neq 1]
\]
\[
s = \text{poly}(0, 's'); E = [s, s^2; 1, 1+s]
\]
\[
E = [1/s, 0; s, 1/(s+1)]
\]
\[
E = ['A11', 'A12'; 'A21', 'A22']
\]

SEE ALSO : poly 65 , string 284 , boolean 27 , rational 68 , syslin 224 , empty 33 , hypermatrices 49

1.1.65  matrix  reshape a vector or a matrix to a different size matrix

CALLING SEQUENCE :

\[
y = \text{matrix}(v, n, m)
\]
\[
y = \text{matrix}(v, [\text{sizes}])
\]

PARAMETERS :

\( v \) : a vector, a matrix or an hypermatrix
\( n, m \) : integers
\( \text{sizes} \) : vector of integers
\( y \) : a vector matrix or hypermatrix

DESCRIPTION :
For a vector or a matrix with \( n \times m \) entries \( y = \text{matrix}(v, n, m) \) or similarly \( y = \text{matrix}(v, [n, m]) \).
transfers the \( v \) vector (or matrix) into an \( n \times m \) matrix by stacking columnwise the entries of \( v \).
if one of the dimension \( m \) or \( n \) is equal to -1 it is automatically assigned to the quotient of \( \text{size}(v, '*') \) by the other dimension,
For an hypermatrix such as \( \text{prod}(\text{size}(v)) = \text{prod}((\text{sizes})) \), \( y = \text{matrix}(v, \text{sizes}) \) (or equivalently \( y = \text{matrix}(v, n1, n2, \ldots, nm) \)) transforms \( v \) into an matrix or hypermatrix by stacking columnwise the entries of \( v \).
\( y = \text{matrix}(v, \text{sizes}) \) results in a regular matrix if \( \text{sizes} \) is a scalar or a 2-vector.

SEE ALSO : matrices 58 , hypermatrices 49 , ones 204 , zeros 231 , rand 207 , poly 65 , empty 33

EXAMPLES :

\[
a = [1 2 3; 4 5 6]
\]
\[
\text{matrix}(a, 1, 6)
\]
\[
\text{matrix}(a, 1, -1)
\]
\[
\text{matrix}(a, 3, 2)
\]

Scilab Group  September 1995  58
1.1.66 mlist Definition

CALLING SEQUENCE:

\[ \text{mlist}(\text{typ}, a_1, \ldots, a_n) \]

PARAMETERS:

- typ : vector of character strings
- ai : any Scilab object (matrix, list, string...).

DESCRIPTION:
mlist object are very similar to tlist objects. But if \( M \) is an mlist, for any index \( i \) which is not a field name, \( M(i) \) is not the \( i \)th field of the list but is interpreted as the \( i \)th entry of \( M \) seen as a vector. This is the only difference between mlist and tlist.

mlist fields must then be designed by their names. They can also be handled using the getfield and setfield functions.

EXAMPLE:

```scilab
M = mlist(['V','name','value'], ['a','b','c'], [1 2 3]);
// define display
def('\%V_p(M)', 'disp(M.name+'''''+string(M.value))')
// define extraction operation
def('r=%V_e(i,M)', ..
   'r=mlist([''V'',''name'',''value''],M.name(i),M.value(i))')
M(2) // the second entry of the vector M
M(2).value
// define M as a tlist
M = tlist(['V','name','value'], ['a','b','c'], [1 2 3]);
M(2)
M('name')
// with two indices
M = mlist(['V','name','value'], ['a','b';'c' 'd'], [1 2;3 4]);
def('r=%V_e(varargin)', [M=varargin($);
   'H=[''V'',''name'',''value'']
   'r=mlist(H,M.name(varargin(1:$-1)),M.value(varargin(1:$-1)))'])
M(:,2)
// multidimensionnal array
str=['a','b','c','d','e','f','g','h'];
n=hypermat([2,2,2],str);
v=hypermat([2,2,2],1:8);
M=mlist(['V','name','value'],n,v);
M(1,1:2,2)
```

SEE ALSO: tlist 73, list 57, overloading 61, getfield 45, setfield 70

1.1.67 mode Select a mode in exec file

Scilab Group April 1993 59
CALLING SEQUENCE:

mode(k)
k=mode()

DESCRIPTION:
Used inside an exec-file or a scilab function \texttt{mode(k)} allows to change the information displayed during the execution, depending on the value of \texttt{k}:

\texttt{k=0} : The new variable values are displayed if required (see help on semi or comma).
\texttt{k=-1} : the exec file or scilab function executes silently. (this is the default value for scilab functions)
\texttt{k=1 or k=3} : each line of instructions is echoed preceded of the prompt. The new variable values are displayed if required. This is the default for exec files.
\texttt{k=7} : The new variable values are displayed if required, each line of instructions is echoed (if possible) and a prompt (\texttt{>>}) is issued after each line waiting for a carriage return.

line display is disable for compiled scilab function (see getf or comp)

\textbf{SEE ALSO}: \texttt{exec 36, getf 272, semi 70, comma 32}

\subsection{1.1.68 \texttt{mtlb_mode} \textbf{-----------------------------} switch Matlab like operations}

CALLING SEQUENCE:

\texttt{mmode=mtlb\_mode()}
\texttt{mtlb\_mode(mmode)}

PARAMETERS:

\texttt{mmode} : boolean

DESCRIPTION:
Scilab and Matlab additions and substractions work differently when used with empty matrices:

\textbf{Scilab}:
\begin{verbatim}
    a+[] -->a
    a-[] -->a
    []+a -->a
    []-a -->-a
\end{verbatim}

\textbf{Matlab} \begin{verbatim}
    a+[] -->[]
    a-[] -->[]
    []+a -->[]
    []-a -->[]
\end{verbatim}

\texttt{mtlb\_mode(\%t)} switches to Matlab evaluation mode for additions and substractions, \texttt{mtlb\_mode(\%f)} switches back to Scilab mode.

\texttt{mtlb\_mode()} return the current \texttt{mmode'} value

\textbf{SEE ALSO}: \texttt{empty 33}
1.1.69 names scilab names syntax

DESCRIPTION:
Names of variables and functions must begin with a letter or one of the following special characters '%%', '\', '#', '!', '\$', '?'.

Next characters may be letters or digits or any special character in '%%', '\', '#', '!', '\$', '?'. Names may be as long as you want but only the first 24 characters are taken into account. Upper and lower case letters are different.

EXAMPLES:
//Valid names
%eps
A1=123
#Color=8
My_Special_Color_Table=rand(10,3)
//Non valid names
//1A , b%, .C

1.1.70 null delete an element in a list

CALLING SEQUENCE:

l(i)=null()

DESCRIPTION:
Deletion of objects inside a list

EXAMPLE:

l=list(1,2,3);
l(2)=null() // get list(1,3)

SEE ALSO: list 57, clear 30

1.1.71 overloading display, functions and operators overloading capabilities

DESCRIPTION:
In scilab, variable display, functions and operators may be defined for new objects using functions (scilab coded or primitives).

Display: The display of new objects defined by tlist structure may be overloaded (the default display is similar to list's one). The overloading function must have no output argument a single input argument. It's name is formed as follow: %%<tlist_type>p where %%<tlist_type> stands for the first entry of the tlist type component.

Operators: Each operator which is not defined for given operands type may be defined. The overloading function must have a single output argument and one or two inputs according to the number of operands. The function name is formed as follow:

for binary operators: %<first_operand_type><op_code><second_operand_type>
for unary operators: %<operand_type><op_code>

extraction and insertion operators which are n-nary operators are described below. <operand_type>, %<first_operand_type>,<second_operand_type> are sequence of characters associated with each data type as described in the following table:
overloading Scilab keyword

<table>
<thead>
<tr>
<th>string</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>polynomial</td>
<td>p</td>
</tr>
<tr>
<td>function</td>
<td>m</td>
</tr>
<tr>
<td>constant</td>
<td>s</td>
</tr>
<tr>
<td>list</td>
<td>l</td>
</tr>
<tr>
<td>tlist</td>
<td>l</td>
</tr>
<tr>
<td>tlist_type</td>
<td>l</td>
</tr>
<tr>
<td>boolean</td>
<td>b</td>
</tr>
<tr>
<td>sparse</td>
<td>sp</td>
</tr>
<tr>
<td>boolean sparse</td>
<td>spb</td>
</tr>
</tbody>
</table>

<op_code> is a single character associated with each operator as described in the following table:

|   | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| / | + | - | * | / | \ | . | [a,b] | [a;b] |

() extraction
() insertion
==
<>
| g | h | j | 5 | 0 | 1 | 2 | 3 | 4 |

The overloading function for extraction syntax \( b = a(1, \ldots, n) \) has the following calling sequence:
\[ b = \% \langle \text{type of } a \rangle \_e(t, \ldots, n, a) \]
and the syntax \([x_1, \ldots, x_m] = a(1, \ldots, n)\) has the following calling sequence:
\[ [x_1, \ldots, x_m] = \% \langle \text{type of } a \rangle \_e(t, \ldots, n, a) \]
The overloading function associated to the insertion syntax \( a(1, \ldots, n) = b \) has the following calling sequence:
\[ a = \% \langle \text{type of } a \rangle \_i \langle \text{type of } b \rangle (t, \ldots, n, b, a) \]

Functions: Some basic primitive function may also be overloaded for new data type. When such a function is undefined for a particular data type the function \( \% \langle \text{type of an argument} \rangle \langle \text{function name} \rangle \) is called. User may add in this called function the definition associated with the input data types.

SEE ALSO: tlist 73, disp 233, symbols 72

Scilab Group

April 1993

62
EXAMPLES:

//DISPLAY
def([]=%tab_p(l)’,’disp(['’ ’;l(3)] [l(2);string(l(4))]))
tlist(’tab’,[’a’,’b’],[’x’;’y’],rand(2,2))

//OPERATOR
def(’x=%c_a_s(a,b)’,’x=a+string(b)’)
’s’+1

//FUNCTION
def(’x=%c_sin(a)’,’x=’’sin(’’+a’’)’’)
sin(’2*x’)

1.1.72 parents ____________________________ ) - left and right parenthesis

CALLING SEQUENCE:

(expression)
[...]=func(e1,e2,...)
[x1,x2,...]=(e1,e2,...)
x(i,j)
v(i)
[...]=l(i)

PARAMETERS:

x : matrix of any possible type
v : row or column vector of any possible type
l : list variable
func : any function name
e1,e2,... : any possible type expression

DESCRIPTION:

Left and right parenthesis are used to

* Specify evaluation order within expressions,
* Form right-hand-side functions argument list. Within multiple rhs arguments must be separated by comma.
* Select elements within vectors, matrices and lists. see help on extraction and insertion for more precisions
* [x1,x2,...]=(e1,e2,...) is equivalent to first performing %t_1 = e1, %t_2 = e2, ..., and then x1 = %t_1, x2 = %t_2, ..., where the variables %t_i, i = 1, 2, ... are invisible to the user.

EXAMPLE:

3^(-1)
x=poly(0,"x");
//
(x+10)/2
i3=eye(3,3)
//
a=[1 2 3;4 5 6;7 8 9],a(1,3),a([1 3],:],a(:,3)
a(:,3)=[]
a(1,$)=33
a(2,$-1)
\(a(:,\text{ }+$1$)=\left[10;11;12\right]\)

\[//\]

\(\text{w}=\text{ssrand}(2,2,2);\text{ssprint} (\text{w})\)
\(\text{ssprint}(\text{w}(:,1))\)
\(\text{ss2tf}(\text{w}(:,1))\)

\[//\]

\(\text{l}\text{=list}(1,2,3,4)\)
\(\text{[a,b,c,d]}=\text{l}(:)\)
\(\text{l}($+$1$)=’\text{new’}\)

\[//\]

\(\text{v}=\%\text{t}(\left[1 1 1 1\right])\)

\[//\]

\(\text{[x,y,z]}=(1,2,3)\)

See Also: colon 31, comma 32, brackets 28, list 57, extraction 39, insertion 51

1.1.73 pause .............................. pause mode, invoke keyboard

**DESCRIPTION:**

Switch to the pause mode; inserted in the code of a function, pause interrupts the execution of the function: one receives a prompt symbol which indicates the level of the pause (e.g. -1->). The user is then in a new session in which all the lower-level variables (and in particular all the variable of the function) are available. To return to lower session enter "return"

In this mode, [...] = return (...) returns the variables of the argument (…) to lower session with names in the output [...]. Otherwise, the lower-level variables are protected and cannot be modified. The pause is extremely useful for debugging purposes.

This mode is killed by the command "abort".

See Also: halt 289, return 68, abort 26, quit 67, whereami 76, where 76

1.1.74 percent .................................. - special character

**DESCRIPTION:**

Some predefined variables begin with \%, such as \%i (for \(\sqrt{-1}\)), \%inf (for Infinity), \%pi (for 3.14...), \%T (for the boolean variable "true")....

In addition, functions whose names begin with \% are special: they are used for coding (extensions of usual) operations.

For example the function \%rmr performs the multiplication (m) operation \(x*y\) for \(x\) and \(y\) rational matrices (r). The coding conventions are given by the readme file in directory SCIDIR/macros/percent.

**EXAMPLE:**

\(\text{x1=tlist(’x’,1,2);}\)
\(\text{x2=tlist(’x’,2,3);}\)
\(\text{deff(’x=\%mx(x1,x2)’,’x=list(’’x’’,x1(2)*x2(2),x2(3)*x2(3))’);}\)
\(\text{x1*x2}\)

1.1.75 plus ................................... - addition operator

**CALLING SEQUENCE:**
**Scilab Function**

X+Y
str1+str2

**PARAMETERS :**

X : scalar or vector or matrix of numbers, polynomials or rationals. It may also be a syslin list
Y : scalar or vector or matrix of numbers, polynomials or rationals. It may also be a syslin list
str1 : a character string, a vector or a matrix of character strings
str2 : a character string, a vector or a matrix of character strings

**DESCRIPTION :**

Addition.
For numeric operands addition as its usual meaning. If one of the operands is a matrix and the other one a scalar the scalar is added to each matrix entries. if one of the operands is an empty matrix the other operand is returned.
For character strings + means concatenation.
Addition may also be defined for other data types through "soft-coded" operations.

**EXAMPLE :**

```
[1,2]+1
[]+2
s=poly(0,"s");
s+2
1/s+2
"cat"+"enate"
```

SEE ALSO:  addf 164, mtlb_mode 60

---

**1.1.76 poly ____________________________ polynomial definition**

**CALLING SEQUENCE :**

[p]=poly(a,"x", ["flag"])

**PARAMETERS :**

a : matrix or real number
x : symbolic variable
"flag" : string ("roots", "coeff"), default value is "roots".

**DESCRIPTION :**

If a is a matrix, p is the characteristic polynomial i.e. determinant(x*eye()-a), x being the symbolic variable.
If v is a vector, poly(v,"x",["roots"]) is the polynomial with roots the entries of v and "x" as formal variable. (In this case, roots and poly are inverse functions).
poly(v,"x", "coeff") creates the polynomial with symbol "x" and with coefficients the entries of v. (Here poly and coeff are inverse functions).
s=poly(0,"s") is the seed for defining polynomials with symbol "s".

**EXAMPLE :**

```
s=poly(0,"s");p=1+s+2*s^2;
A=rand(2,2);poly(A,"x")
```

SEE ALSO:  coeff 485, matrices 58, rational 68

---

Scilab Group  
April 1993  
65
1.1.77  power  ________________________________  power operation (\(^{,}\))

CALLING SEQUENCE:

\[ t = A^b \]
\[ t = A**b \]
\[ t = A.\hat{b} \]

PARAMETERS:

- \( A, t \) : scalar, polynomial or rational matrix.
- \( b \) : a scalar, a vector or a scalar matrix.

DESCRIPTION:

\((A:\text{square})^\hat{(b:\text{scalar})} \): If \( A \) is a square matrix and \( b \) is a scalar then \( A^b \) is the matrix \( A \) to the power \( b \).

\((A:\text{matrix})^\hat{(b:\text{scalar})} \): If \( b \) is a scalar and \( A \) a matrix then \( A^b \) is the matrix formed by the element of \( A \) to the power \( b \) (elementwise power). If \( A \) is a vector and \( b \) is a scalar then \( A^b \) and \( A.\hat{b} \) performs the same operation (i.e elementwise power).

\((A:\text{scalar})^\hat{(b:\text{matrix})} \): If \( A \) is a scalar and \( b \) is a scalar matrix (or vector) then \( A^b \) and \( A.\hat{b} \) are the matrices (or vectors) formed by \( a^\hat{(b(i,j))} \).

\((A:\text{matrix})^\hat{(b:\text{matrix})} \): If \( A \) and \( b \) are vectors (matrices) with compatible dimensions then \( A^b \) is the \( A(i)\hat{b}(i) \) vector (\( A(i,j)\hat{b}(i,j) \) matrix).

Notes:

- For square matrices \( A^p \) is computed through successive matrices multiplications if \( p \) is a positive integer, and by diagonalization if not.
- ** and ^ operators are synonyms.

EXAMPLE:

\[ A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \]
\[ A^2.5, \]
\[ A.\hat{2.5} \]
\[(1:10)^\hat{2} \]
\[(1:10).\hat{2} \]

\[ s = \text{poly}(0,'s') \]
\[ s^\hat{(1:10)} \]

SEE ALSO: exp 508

1.1.78  predef  ________________________________  variable protection

CALLING SEQUENCE:

\[ n = \text{predef}() \]
\[ \text{oldnew} = \text{predef}(n) \]
\[ \text{oldnew} = \text{predef('all')} \]
\[ \text{oldnew} = \text{predef('clear')} \]
**DESCRIPTION:**
Utility function used for defining "oldest" variables as "protected". Protected variables cannot be killed. They are not saved by the 'save' command. The "oldest" are those appearing last in the fVwho('get').

`predef()` gets the number of protected variables
`predef('a[ll]')` sets all the variables protected, it also return the old and new value of protected variables number.
`predef('c[lear]')` unprotect all but the last 7 variables, it also return the old and new value of protected variables number.
`predef(n)` sets the max(n, 7) last defined variables as protected, it also return the old and new value of protected variables number.

**REMARK:**
A number of protected variables are set in the start-up file scilab.star.
User may in particular set its own predefined variables in user's startup file home/.scilab

**SEE ALSO:** clear 30, save 258

### 1.1.79 pwd ________________________________ print Scilab current directory

**getcwd** - get Scilab current directory

**CALLING SEQUENCE:**

`pwd`
`x=pwd()`
`x=getcwd()`

**DESCRIPTION:**
`pwd` returns in ans the Scilab current directory. `x=pwd()` or fVx=getcwd() returns in x the Scilab current directory.

**EXAMPLE:**

`pwd`
`x=pwd()`

**SEE ALSO:** chdir 297, unix 310

### 1.1.80 quit _______________________________ decrease the pause level or exit

**DESCRIPTION:**
`quit` terminates Scilab or decreases the pause level.

**SEE ALSO:** pause 64, break 28, abort 26, exit 38

### 1.1.81 quote ______________________________ - transpose operator, string delimiter

**DESCRIPTION:**
`quote (')` is used for (Conjugate) Transpose of matrix.
`quote (.')` is used for (non Conjugate) Transpose of matrix.
Simple (') or double (") quotes are also used to define character strings. (Character strings are defined between two quotes). A Quote within a character string is denoted by two quotes.

**EXAMPLES:**

```
[1+%i, 2']
[1+%i, 2].'  
x='This is a character string'
'He said:''Good'''
```
1.1.82 rational ________________ Scilab objects, rational in Scilab

DESCRIPTION:
A rational \( r \) is a quotient of two polynomials \( r = \frac{\text{num}}{\text{den}} \). The internal representation of a rational is a list \( r = \text{tlist}(["r","\text{num"},"\text{den"},"\text{dt"}], \text{num}, \text{den}, []) \) is the same as \( r = \frac{\text{num}}{\text{den}} \). A rational matrix can be defined with the usual syntax e.g. \( [\text{r11}, \text{r12}; \text{r21}, \text{r22}] \) is a 2x2 matrix where \( r_{ij} \) are 1x1 rationals. A rational matrix can also be defined as above as a list \( \text{tlist}(["r","\text{num"},"\text{den"},"\text{dt"}], \text{num}, \text{den}, []) \) with \( \text{num} \) and \( \text{den} \) polynomial matrices.

EXAMPLES:
\[
\begin{align*}
s &= \text{poly}(0, 's'); \\
W &= [1/s, 1/(s+1)] \\
W' &= W \\
\text{Num} &= [s, s+2; 1, s]; \\
\text{Den} &= [s*s, s; s, s*s]; \\
\text{tlist}(["r", 'num', 'den', 'dt'], \text{Num}, \text{Den}, []) \\
H &= \text{Num}./\text{Den} \\
\text{syslin('c', Num, Den)} \\
\text{syslin('c', H)} \\
[\text{Num1}, \text{Den1}] &= \text{simp} (\text{Num}, \text{Den})
\end{align*}
\]

SEE ALSO: poly 65, syslin 224, simp 497

1.1.83 resume _______ return or resume execution and copy some local variables

CALLING SEQUENCE:
\[
\text{resume} \\
[x_1, \ldots, x_n] = \text{resume}(a_1, \ldots, a_n)
\]

PARAMETERS:
\( x \)

DESCRIPTION:
In a function \( \text{resume} \) stops the execution of the function, \( [\ldots] = \text{resume}(\ldots) \) stops the execution of the function and put the local variables \( a_i \) in calling environment under names \( x_i \). In \text{pause} mode, it allows to return to lower level \( [\ldots] = \text{resume}(\ldots) \) returns to lower level and put the local variables \( a_i \) in calling environment under names \( x_i \). In an execstr called by a function \( [\ldots] = \text{resume}(\ldots) \) stops the execution of the function and put the local variables \( a_i \) in calling environment under names \( x_i \). \text{resume} is equivalent to \text{return}.

SEE ALSO: abort 26, break 28

1.1.84 return _______ return or resume execution and copy some local variables

CALLING SEQUENCE:
\[
\text{return} \\
[x_1, \ldots, x_n] = \text{return}(a_1, \ldots, a_n)
\]

PARAMETERS:

Scilab Group April 1993 68
DESCRIPTION:
In a function return stops the execution of the function, \([..]=\text{return}(..)\) stops the execution of the function and put the local variables \(ai\) in calling environment under names \(xi\). In pause mode, it allows to return to lower level \([..]=\text{return}(..)\) returns to lower level and put the local variables \(ai\) in calling environment under names \(xi\). In an execstr called by a function \([..]=\text{return}(..)\) stops the execution of the function and put the local variables \(ai\) in calling environment under names \(xi\). resume is equivalent to return.

SEE ALSO: abort 26, break 28

1.1.85 rlist ____________________ Scilab rational fraction function definition

CALLING SEQUENCE:

\[ \text{rlist}(\) \]
\[ \text{rlist}(a1, \ldots, an) \]

DESCRIPTION:
\[ \text{rlist}(a1, \ldots, an) \] is a shortcut to \(\text{tlist}(['r','num';'den','dt'], a1, \ldots, an)\)
Creates a tlist with \(['r','num';'den','dt']\) as first entry and \(ai\)'s as next entries if any. No type nor size checking is done on \(ai\)'s.

SEE ALSO: tlist 73, syslin 224

1.1.86 sciargs ______________________ scilab command line arguments

CALLING SEQUENCE:

\[ \text{args=sciargs()} \]

DESCRIPTION:
This function returns a vector of character strings containing the arguments of the Scilab command line. First args entry contains the path of the launched executable file.
This function correspond to the getarg function in C language

SEE ALSO: getenv 45

1.1.87 select _____________________________ select keyword

DESCRIPTION:

\[
\text{select expr,}
\quad \text{case expr1 then instructions1,}
\quad \text{case expr2 then instructions2,}
\quad \ldots
\quad \text{case exprn then instructionsn,}
\quad [\text{else instructions}],
\quad \text{end}
\]

Notes:
- The only constraint is that each "then" keyword must be on the same line line as corresponding "case" keyword.
- The "keyword "then" can be replaced by a carriage return or a comma.

\[
\text{instructions1 are executed if expr1=expr, etc.}
\]

Warning: the number of characters used to define the body of any conditionnal instruction (if while for or select/case) must be limited to 16k.

**EXAMPLE:**

```plaintext
while %t do
    n=round(10*rand(1,1))
    select n
    case 0 then
        disp(0)
    case 1 then
        disp(1)
        else
            break
    end
end
```

**SEE ALSO:** if 50, while 77, for 42

### 1.1.88 semi

- **semi ____________________________ - instruction and row separator**

**DESCRIPTION :**
semicolumns are used to separate rows in a matrix definition.
semicolumns may also be used at the end of an instruction. In this case it means that the result(s) are not displayed. Conversely use comma (,) to get the display

**EXAMPLES :**

```plaintext
a=[1,2,3;4,5,6];
a=1;b=1, c=2
```

**SEE ALSO:** comma 32, brackets 28

### 1.1.89 semicolon

- **semicolon ________________ - ending expression and row separator**

**DESCRIPTION :**
In a file, the line separator ";" suppresses the display of the line.
Within brackets ; denotes row separator in matrix definition.

**EXAMPLES :**

```plaintext
sin(%pi) sin(%pi); a=[1,2;3 4]
```

### 1.1.90 setfield

- **setfield ________________________________ list field insertion**

**CALLING SEQUENCE :**

```plaintext
setfield(i,x,1)
```
PARAMETERS:

- $x$ : matrix of any possible types
- $l$ : list, tlist or mlist variable
- $i$ : field index, see insertion for more details.

DESCRIPTION:

This function is an equivalent of $l(i)=x$ syntax for field extraction with the only difference that it also applies to mlist objects.

EXAMPLE:

```plaintext
l=list(1,'qwerw','%s')
l(1)='Changed'
l(0)='Added'
l(6)=[‘one more’;’added’]

a=hypermat([2,2,2],rand(1:2^3)); // hypermatrices are coded using mlists
setfield(3,1:8,a);a // set the field value to 1:8
```

SEE ALSO: insertion 51

1.1.91 slash .............................................. - right division and feed back

DESCRIPTION:

Right division. $x=A / b$ is the solution of $x*b=A$.

$\frac{b}{a} = (a' \backslash b')'$.

$a . / b$ is the matrix with entries $a(i,j) / b(i,j)$. If $b$ is scalar (1x1 matrix) this operation is the same as $a./b*ones(a)$ (Same convention if $a$ is a scalar).

Remark that $123./b$ is interpreted as $(123.)/b$. In this cases dot is part of the number not of the operator.

Backslash stands for left division.

System feed back. $S=G/.K$ evaluates $S=G*(eye()+K*G)^{-1}$ this operator avoid simplification problem.

Remark that $G/.5$ is interpreted as $G/(.5)$. In such cases dot is part of the number, not of the operator.

Comment // comments a line i.e lines which begin by // are ignored by the interpreter.

SEE ALSO: inv 516, percent 64, backslash 26, ieee 50

1.1.92 stacksize ............................................. set scilab stack size

CALLING SEQUENCE:

```plaintext
stacksize(n)
sz=stacksize()
```

PARAMETERS:

- $n$ : integer, the required stack size given in number of double precision words
- $sz$ : 2-vector [total used]
DESCRIPTION:
Scilab stores all variables in a unique stack stk.

stacksize(n) allows the user to increase or decrease the size of this stack. The maximum allowed size depends on the amount of free memory and swap space available at the time.

This function with the n argument may only be called at the main prompt; it cannot be called within a scilab function.

sz=stacksize() returns a 2-vector which contains the current total and used stack size. It can be used everywhere.

SEE ALSO: who 77

1.1.93 star ----------------------------------------- multiplication operator

DESCRIPTION:
Multiplication. Usual meaning. Valid for constant, boolean, polynomial and rational matrices.
Element-wise multiplication is denoted x.*y. If x or y is scalar (1x1 matrix) .* is the same as *.
Kronecker product is x.*.y

SEE ALSO: mulf 203

1.1.94 symbols -------------------------------------- scilab operator names

DESCRIPTION:
Use the following names to get help on a specific symbol.

<table>
<thead>
<tr>
<th>operator</th>
<th>name in Scilab help</th>
</tr>
</thead>
<tbody>
<tr>
<td>',&quot;,'</td>
<td>quote</td>
</tr>
<tr>
<td>+</td>
<td>plus</td>
</tr>
<tr>
<td>-</td>
<td>minus</td>
</tr>
<tr>
<td>'*'</td>
<td>star</td>
</tr>
<tr>
<td>'/'</td>
<td>slash</td>
</tr>
<tr>
<td>'',''</td>
<td>backslash</td>
</tr>
<tr>
<td>'.'</td>
<td>dot</td>
</tr>
<tr>
<td>'='</td>
<td>equal</td>
</tr>
<tr>
<td>'&lt;','&lt;='</td>
<td>less</td>
</tr>
<tr>
<td>'~'</td>
<td>tilda</td>
</tr>
<tr>
<td>'['</td>
<td>left</td>
</tr>
<tr>
<td>']'</td>
<td>right</td>
</tr>
<tr>
<td>'()'</td>
<td>parents</td>
</tr>
<tr>
<td>'%'</td>
<td>percent</td>
</tr>
<tr>
<td>':'</td>
<td>column</td>
</tr>
<tr>
<td>','</td>
<td>comma</td>
</tr>
<tr>
<td>';'</td>
<td>semi</td>
</tr>
<tr>
<td>'^'</td>
<td>hat</td>
</tr>
<tr>
<td>'.^'</td>
<td>power</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>'&amp;'</td>
<td>and</td>
</tr>
<tr>
<td>'.*'</td>
<td>kron</td>
</tr>
</tbody>
</table>

SEE ALSO: overloading 61
1.1.95  testmatrix  generate some particular matrices

CALLING SEQUENCE :

[y]=testmatrix(name,n)

PARAMETERS :

name : a character string
n : integers, matrix size
y : n x m matrix

DESCRIPTION :
Create some particular matrices

testmatrix('magi',n) : returns a magic square of size n.
testmatrix('frk',n) : returns the Franck matrix:
testmatrix('hilb',n) : is the inverse of the nxn Hilbert matrix (Hij= 1/(i+j-1)).

1.1.96  then  keyword in if-then-else

DESCRIPTION :
Used with if.
SEE ALSO :  if 50

1.1.97  tilda  - logical not

CALLING SEQUENCE :

˜m

PARAMETERS :

m : boolean matrix

DESCRIPTION :
˜m  is the negation of m.

1.1.98  tlist  Scilab object and typed list definition.

CALLING SEQUENCE :

tlist(typ,a1,...,an)

PARAMETERS :

typ : Character string or vector of character strings
ai : Any Scilab object (matrix, list,string...).
DESCRIPTION:
Creates a typed-list with elements ai's. The typ argument specifies the list type. Such typed-list allow the user to define new operations working on these object through scilab functions. The only difference between typed-list and list is the value of the type (16 instead of 15).
typ(1) specifies the list type (character string used to define soft coded operations)
if specified typ(i) may give the i+1th element formal name
Standard Operations on list work similarly for typed-list:

extraction : [x,y,z...] = l(v) where v is a vector of indices; [x,y,z] = l(:) extracts all the elements
insertion : l(i) = a
deletion : l(i) = null() removes the i-th element of the tlist l.
display
Moreover if typ(2:n+1) are specified, user may point elements by their names

Linear systems are represented by specific typed-list e.g. a linear system [A,B,C,D] is represented by the tlist Sys=tlist([‘lss’;’A’;’B’;’C’;’D’;’X0’;’dt’],A,B,C,D,x0,’c’)
and this specific list may be created by the function syslin.
Sys(2) or Sys(‘A’) is the state-matrix and Sys(‘td’) is the time domain
A rational matrix H is represented by the typed-list H=tlist([‘r’;’num’;’den’;’dt’],Num,Den,[0])
where Num and Den are two polynomial matrices and a (e.g. continuous time) linear system with transfer matrix H maybe created by syslin(’c’,H).
H(2) or H(’num’) is the transfer matrix numerator

SEE ALSO: null 61, percent 64, syslin 224, list 57

1.1.99 type ____________________________________________ variable type

CALLING SEQUENCE:

[i]=type(x)

PARAMETERS:

x : Scilab object
i : integer

DESCRIPTION:

type(x) returns an integer which is the type of x as following:

1 : real or complex constant matrix.
2 : polynomial matrix.
4 : boolean matrix.
5 : sparse matrix.
8 : matrix of integers stored on 1 2 or 4 bytes
10 : matrix of character strings.
11 : un-compiled function.
13 : compiled function.
14 : function library.
15 : list.
16 : typed list (tlist)
128 : pointer

SEE ALSO: typeof 229

Scilab Group April 1993 74
1.1.100  typename __________________ associates a name to variable type

CALLING SEQUENCE:

[types [,names]]=typename()
typename(name,type)

PARAMETERS:

types : integer column vector: the types codes of each defined data types.
names : column vector of strings: the names associated to type codes.
type : integer: the type code of new data type.
name : string: the name associated to the type code.

DESCRIPTION:

The function and operator overloading make use of a formal name associated to data types to form the
name of the overloading function (see overloading). The typename can be used to handle this formal
names for hard coded data types (the tlist or mlist coded data types formal names are defined in an
other way, see overloading).
Called without right hand side argument, typename returns informations on defined data types.
Called with right hand side argument, typename associates a name to a data type code.
typename(’’,type) suppress the data type given by its code type out of the table of known data
types.
SEE ALSO: type 74, typeof 229, overloading 61, tlist 73, mlist 59

1.1.101  user _____________________________ interfacing a fortran routine

CALLING SEQUENCE:

[s_1,s_2,...,s_lhs]=user(e_1,e_2,...,e_rhs)

DESCRIPTION:

With this command it is possible to use an external program as a Scilab command where (s_1,s_2,...,s_lhs)
are the output variables and (e_1,e_2,...,e_rhs) are the input variables. To insert this command in
Scilab one has to write a few lines in the user fortran subroutine of Scilab. See intersci or the Scilab
documentation for more information.
SEE ALSO: fort 43, link 303

1.1.102  varn _____________________________ symbolic variable of a polynomial

CALLING SEQUENCE:

[symb]=varn(p)
[pm]=varn(x,var)

PARAMETERS:

p : polynomial (or matrix polynomial)
symb : character string
x : polynomial or polynomial matrix
var : symbolic variable (character string)
pm : matrix or polynomial matrix

Scilab Group April 1993 75
**DESCRIPTION:**
symb=varn(p) returns in symb the symbolic variable of the polynomial p (i.e. varn(poly(0,'x')) is 'x').
varn(x,'s') returns a polynomial matrix with same coefficients as x but with 's' as symbolic variable (change of variable name).

**EXAMPLE:**

s=poly(0,'s');p=[s^2+1,s];
    varn(p) is the string 's' and varn(p,'x') is the polynomial matrix [x^2+1,x]

**SEE ALSO:** horner 490, poly 65

---

**1.1.103** what ________________________________ list the Scilab primitives

**DESCRIPTION:**
List of low level primitives and commands.

**1.1.104** where ____________________________ get current instruction calling tree

**CALLING SEQUENCE:**

[linenum,mac]=where()

**PARAMETERS:**

linenum : column vector of integer
mac : column vector of strings

**DESCRIPTION:**
returns linenum and mac such as current instruction has been called by the linenum(1) line of function mac(1), mac(1) has been called by the linenum(2) line of function mac(2) and so on
mac(i) is in general the name of a function but it may also be "exec" or "execstr" if instruction lies in ans exec file or an execstr instruction

**SEE ALSO:** whereami 76, pause 64

**1.1.105** whereami __________________________ display current instruction calling tree

**CALLING SEQUENCE:**

whereami()

**DESCRIPTION:**
Displays calling tree to instruction which contain whereami(). May be used within pause levels.

**EXAMPLE:**

deff('y=test(a)',[y=sin(a)+1; y=t1(y); y=y+1'])
deff('y=t1(y)',[y=2; whereami()])
test(1)

**SEE ALSO:** where 76, pause 64, errcatch 34
1.1.106 whereis

**Calling Sequence:**

[librname] = whereis (function-name)

**Description:**

returns as a character string the name of the library containing the function function-name. The path of the library is returned by typing "librname".

See Also: lib 272

1.1.107 while

**Description:**

while clause. Must be terminated by "end"

- The only constraint is that each "then" or "do" keyword must be on the same line line as "while" keyword.
- The "keyword "then" or "do" can be replaced by a carriage return or a comma.
- The optional ,else instructions construction allows to gives instructions which are executed when expr expression becomes false.

Warning: the number of characters used to define the body of any conditionnal instruction (if while for or select/case) must be limited to 16k.

**Example:**

e=1; a=1; k=1;
while norm(a-(a+e),1) > %eps, e=e/2; k=k+1; end

e, k

See Also: for 42, select 69, break 28, return 68, pause 64

1.1.108 who

**Calling Sequence:**

who

who()

names = who (‘local’)

(names, mem) = who (‘local’)

names = who (‘global’)

(names, mem) = who (‘global’)

**Description:**

who displays current variable names. who (‘local’) or who (‘get’) Returns current variable names and memory used in double precision worlds.

who (‘global’) returns global variable names and memory used in double precision worlds.

See Also: whos 78
1.1.109  whos ____________________________ listing of variables in long form

CALLING SEQUENCE :

whos()
whos -type typ
whos -name nam

PARAMETERS :

typ : name of selected variable type (see typeof)
nam : first characters of selected names

DESCRIPTION :

whos()  displays all current variable names, types and memory used
whos -type typ  displays all current variables with specified type
whos -name nam  displays all current variables whose names begin with nam

EXAMPLE :

whos()

whos -type boolean

whos -name %

SEE ALSO:  who 77,  typeof 229
1.2 Graphic Library
2D PLOTTING:

- plot2d: plot a curve
- plot2d2: plot a curve as step function
- plot2d3: plot a curve with vertical bars
- plot2d4: plot a curve with arrows
- fpplot2d: plot a curve defined by a function
- champ: 2D vector field
- champ1: 2D vector field with colored arrows
- fchamp: direction field of a 2D first order ODE
- contour2d: level curves of a surface on a 2D plot
- fcontour2d: level curves of a surface defined by a function on a 2D plot
- grayplot: 2D plot of a surface using colors
- fgrayplot: 2D plot of a surface defined by a function using colors
- Sgrayplot: smooth 2D plot of a surface using colors
- Sfgrayplot: smooth 2D plot of a surface defined by a function using colors
- xgrid: add a grid on a 2D plot
- errbar: add vertical error bars on a 2D plot
- histplot: plot a histogram
- Matplot: 2D plot of a matrix using colors

3D PLOTTING:

- plot3d: plot a surface
- plot3d1: plot a surface with gray or color level
- fpplot3d: plot a surface defined by a function
- fpplot3d1: plot a surface defined by a function with gray or color level
- param3d: plot one curve
- param3d1: plots curves
- contour: level curves on a 3D surface
- fcontour: level curves on a 3D surface defined by a function
- hist3d: 3D representation of a histogram
- genfac3d: compute facets of a 3D surface
- eval3dp: compute facets of a 3D surface
- geom3d: projection from 3D on 2D after a 3D plot

LINE AND POLYGON PLOTTING:

- xpoly: draw a polyline or a polygon
- xpolys: draw a set of polylines or polygons
- xrpoly: draw a regular polygon
- xsegs: draw unconnected segments
- xfpoly: fill a polygon
- xfpolys: fill a set of polygons

RECTANGLE PLOTTING:

- xrect: draw a rectangle
- xfrect: fill a rectangle
- xrects: draw or fill a set of rectangles

ARC PLOTTING:

- xarc: draw a part of an ellipse
ARROW PLOTTING:

xarrows : draw a set of arrows

STRINGS:

xstring : draw strings
xstringl : compute a box which surrounds strings
xstringb : draw strings into a box
xtile : add titles on a graphics window
titlepage : add a title in the middle of a graphics window
xinfo : draw an info string in the message subwindow

FRAMES AND AXES:

xaxis : draw an axis
graduate : pretty axis graduations
plotframe : plot a frame with scaling and grids

COORDINATES TRANSFORMATIONS:

isoview : set scales for isometric plot (do not change the size of the window)
square : set scales for isometric plot (change the size of the window)
scaling : affine transformation of a set of points
rotate : rotation of a set of points
xsetech : set the sub-window of a graphics window for plotting
subplot : divide a graphics window into a matrix of sub-windows
xgetech : get the current graphics scale
xchange : transform real to pixel coordinates

COLORS:

colormap : using colormaps
getcolor : dialog to select colors in the current colormap
addcolor : add new colors to the current colormap
graycolormap : linear gray colormap
hotcolormap : red to yellow colormap

GRAPHICS CONTEXT:

xset : set values of the graphics context
xget : get current values of the graphics context
xlfont : load a font in the graphics context or query loaded font
getsymbol : dialog to select a symbol and its size

SAVE AND LOAD:

xsave : save graphics into a file
xload : load a saved graphics
xbasimp : send graphics to a Postscript printer or in a file
xs2fig : send graphics to a file in Xfig syntax

GRAPHICS PRIMITIVES:

xbasc : clear a graphics window and erase the associated recorded graphics
xclear : clear a graphics window
driver : select a graphics driver
xinit : initialisation of a graphics driver
xend : close a graphics session
xbase : redraw a graphics window
replot : redraw the current graphics window with new boundaries
xpause : suspend Scilab
xselect : raise the current graphics window
xclea : erase a rectangle
xclip : set a clipping zone
xdel : delete a graphics window
winsid : return the list of graphics windows
xname : change the name of the current graphics window

MOUSE POSITION :

xclick : wait for a mouse click
locate : mouse selection of a set of points
xgetmouse : get the current position of the mouse

INTERACTIVE EDITOR :

edit_curv : interactive graphics curve editor
gr_menu : simple interatives graphic editor
sd2sci : gr_menu structure to scilab instruction convertor

GRAPHICS FUNCTIONS FOR AUTOMATIC CONTROL :

bode : Bode plot
gainplot : magnitude plot
nyquist : Nyquist plot
m_circle : M-circle plot
chart : Nichols chart
black : Black's diagram
evans : Evans root locu
sgrid : s-plane grid lines
plzr : pole-zero plot
zgrid : zgrid plot

1.2.2 Matplot 2D plot of a matrix using colors

CALLING SEQUENCE :

Matplot(a,[strf,rect,nax])

PARAMETERS :

a : real matrix of size (n1,n2).
strf,rect,nax : optional arguments, see plot2d.

DESCRIPTION :

The entries of matrix \( \text{int}(a) \) are used as colormap entries in the current colormap. The color associated to \( a(i,j) \) is used do draw a small square of length 1 with center at location \( (x=j, y=(n2-i+1)) \).

Enter the command Matplot() to see a demo.

EXAMPLE :

Scilab Group April 1993 82
Matplot([1 2 3;4 5 6])
// draw the current colormap
Matplot((1:xget("lastpattern")))

SEE ALSO: colormap 89, plot2d 118, Matplot1 83

1.2.3 Matplot1 2D plot of a matrix using colors

CALLING SEQUENCE:
Matplot1(a,rect)

PARAMETERS:
a : real matrix of size (n1,n2).
rect : [xmin;ymin,xmax,ymax]

DESCRIPTION:
The entries of matrix int(a) are used as colormap entries in the current colormap. rect specify a rectangle in the current scale and the matrix is drawn inside this rectangle. Each matrix entry will be rendered as a small rectangle filled with its associated color.

EXAMPLE:

//--- first example
// fix current scale
xsetech(frect=[0,0,10,10])
xrect(0,10,10,10)
a=5*ones(11,11); a(2:10,2:10)=4; a(5:7,5:7)=2;
// first matrix in rectangle [1,1,3,3]
Matplot1(a,[1,1,3,3])
a=ones(10,10); a= 3*tril(a) + 2*a;
// second matrix in rectangle [5,6,7,8]
Matplot1(a,[5,6,7,8])
xset('default')
xbasc()

//--- second example
xsetech(frect=[0,0,10,10])
xrect(0,10,10,10)
n=100;
xset(' pixmap',1)
driver('X11')
for k=-n:n,
a=ones(n,n);
a= 3*tril(a,k) + 2*a;
a= a + a';
k1= 3*(k+100)/200;
Matplot1(a,[k1,2,k1+7,9])
xset('wshow')
xset('wwpc')
end
xset(' pixmap',0)
xset('default')
xbasc()

SEE ALSO: colormap 89, plot2d 118, Matplot 82

AUTHOR: J.Ph.C.
1.2.4 **Sfgrayplot** — smooth 2D plot of a surface defined by a function using colors

**CALLING SEQUENCE:**

\[ \text{Sfgrayplot}(x,y,f,[strf,rect,nax]) \]

**DESCRIPTION:**

Sfgrayplot is the same as fgrayplot but the plot is smoothed. The function fec is used for smoothing. The surface is plotted assuming that it is linear on a set of triangles built from the grid:

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_/</td>
<td>_/</td>
</tr>
<tr>
<td>_/</td>
<td>_/</td>
</tr>
<tr>
<td>_/</td>
<td>_/</td>
</tr>
</tbody>
</table>
```

Enter the command Sfgrayplot() to see a demo.

**EXAMPLE :**

```plaintext
t=-1:0.1:1;
deff("[z]=surf(x,y)","z=x**2+y**2")
Sfgrayplot(t,t,surf,"111",[-2,-2,2,2])
```

**SEE ALSO:** fec 101, fgrayplot 102, grayplot 109, Sgrayplot 84

**AUTHOR:** J.Ph.C.

1.2.5 **Sgrayplot** — smooth 2D plot of a surface using colors

**CALLING SEQUENCE:**

\[ \text{Sgrayplot}(x,y,z,[strf,rect,nax]) \]

**DESCRIPTION:**

Sgrayplot is the same as grayplot but the plot is smoothed. The function fec is used for smoothing. The surface is plotted assuming that it is linear on a set of triangles built from the grid:

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_/</td>
<td>_/</td>
</tr>
<tr>
<td>_/</td>
<td>_/</td>
</tr>
<tr>
<td>_/</td>
<td>_/</td>
</tr>
</tbody>
</table>
```

Enter the command Sgrayplot() to see a demo.

**EXAMPLE :**

```plaintext
x=-10:10; y=-10:10; m =rand(21,21);
Sgrayplot(x,y,m,"111",[-20,-20,20,20])
t=-%pi:0.1:%pi; m=sin(t)’*cos(t);
xbasc()
Sgrayplot(t,t,m)
```

**SEE ALSO:** fec 101, fgrayplot 102, grayplot 109, Sfgrayplot 84

**AUTHOR:** J.Ph.C.
1.2.6  addcolor .......................... add new colors to the current colormap

CALLING SEQUENCE :

new=addcolor(c)

PARAMETERS :

new : ids of the colors defined in c in the new color table.
c : matrix with 3 columns, RGB color definition.

DESCRIPTION :

addcolor adds new colors given in the c argument to the current colormap. c must be a matrix with 3 columns [R G B] R is red component, G is green component, B is blue component. Each entry in c must be a non negative number less or equal to 1. The ids of the new colors are returned into new. If a color defined in c is already present in the current colormap it is not added.

SEE ALSO:  colormap 89

1.2.7  alufunctions ..................... description and number of pixel drawing modes.

DESCRIPTION :

src is the source ie the "value of the pixel" which we want to draw. dst is the destination ie "value of the pixel" which is already drawn.

0: clear ie "0"
1: and ie "src AND dst"
2: and reverse ie "src AND NOT dst"
3: copy ie "src"
4: and inverted ie "(NOT src) AND dst"
5: noop ie "dst"
6: xor ie "src XOR dst"
7: or ie "src OR dst"
8: nor ie "(NOT src) AND (NOT dst)"
9: equiv ie "((NOT src) XOR dst"
10: invert ie "NOT dst"
11: or reverse ie "src OR (NOT dst)"
12: copy inverted ie "NOT src"
13: or inverted ie "(NOT src) OR dst"
14: nand ie "(NOT src) OR (NOT dst)"
15: set ie "1"

1.2.8  black ................................ Black’s diagram (Nichols chart)

CALLING SEQUENCE :

black( sl, [fmin, fmax] [,step] [,comments] )
black( sl, frq [,comments] )
black(frq, db, phi [,comments])
black(frq, repf [,comments])

PARAMETERS :

Scilab Group  April 1993  85
**bode** Scilab Function

**sl**: list (linear system *syslin*)

**fmin, fmax**: real scalars (frequency bounds)

**frq**: row vector or matrix (frequencies)

**db, phi**: row vectors or matrices (modulus, phase)

**repf**: row vectors or matrices (complex frequency response)

**step**: real

**comments**: string

**DESCRIPTION**:
Black’s diagram (Nichols’chart) for a linear system *sl*. *sl* can be a continuous-time or discrete-time SIMO system (see *syslin*). In case of multi-output the outputs are plotted with different symbols. The frequencies are given by the bounds *fmin, fmax* (in Hz) or by a row-vector (or a matrix for multi-output) *frq*.

**step** is the (logarithmic) discretization step. (see *calfrq* for the choice of default value).  
**comments** is a vector of character strings (captions).

**db, phi** are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).

**repf** matrix of complex numbers. One row for each response.

To plot the grid of iso-gain and iso-phase of $y/(1+y)$ use *chart()*.

Default values for *fmin* and *fmax* are $1.d-3, 1.d+3$ if *sl* is continuous-time or $1.d-3, 0.5$ if *sl* is discrete-time.

**EXAMPLE**:

```plaintext
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
chart();
sstr='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
black(h,0.01,100,sstr);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
xbasc()
black([h1;h],0.01,100,['h1';'h'])
```

**SEE ALSO**: bode 86, nyquist 115, chart 88, freq 339, repfreq 355, calfrq 324, phasemag 353

---

**1.2.9  bode**  Bode plot

**CALLING SEQUENCE**:

```plaintext
bode(sl,[fmin,fmax] [,step] [,comments] )
bode(sl,frq [,comments] )
bode(frq,db,phi [,comments])
bode(frq, repf [,comments])
```

**PARAMETERS**:

- **sl**: *syslin* list (SISO or SIMO linear system) in continuous or discrete time.
- **fmin, fmax**: real (frequency bounds (in Hz))
- **step**: real (logarithmic step.)
- **comments**: vector of character strings (captions).
- **frq**: row vector or matrix (frequencies (in Hz)) (one row for each SISO subsystem).
- **db**: row vector or matrix (magnitudes (in Db)) (one row for each SISO subsystem).
- **phi**: row vector or matrix (phases (in degree)) (one row for each SISO subsystem).
- **repf**: row vector or matrix of complex numbers (complex frequency response).

---

Scilab Group  April 1993  86
DESCRIPTION:
Bode plot, i.e. magnitude and phase of the frequency response of sl.
sl can be a continuous-time or discrete-time SIMO system (see syslin). In case of multi-output the outputs are plotted with different symbols.
The frequencies are given by the bounds \( f_{\text{min}}, f_{\text{max}} \) (in Hz) or by a row-vector (or a matrix for multi-output) frq.
step is the (logarithmic) discretization step. (see calfrq for the choice of default value).
comments is a vector of character strings (captions).
db, phi are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).
repf matrix of complex numbers. One row for each response.
Default values for \( f_{\text{min}} \) and \( f_{\text{max}} \) are \( 1.d^{-3}, 1.d^{+3} \) if sl is continuous-time or \( 1.d^{-3}, 0.5 \) if sl is discrete-time. Automatic discretization of frequencies is made by calfrq.

EXAMPLE:

```scilab
s=poly(0,'s')
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
title='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
bode(h,0.01,100,title);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
xbasc()
bode([h1;h],0.01,100,['h1';'h'])
```

SEE ALSO: black 85, nyquist 115, gainplot 104, repfreq 355, g_margin 340, p_margin 352, calfrq 324, phasemag 353

1.2.10 champ ************************************ 2D vector field plot

CALLING SEQUENCE:

```scilab
champ(x, y, fx, fy, [arfact, rect, strf])
champ(x, y, fx, fy, <opt_args>)
```

PARAMETERS:

- \( x, y \) : two vectors which define the grid.
- \( fx \) : a matrix which describes the x component of the vector field. \( fx(i, j) \) is the x component of the vector field at point \( (x(i), y(j)) \).
- \( fy \) : a matrix which describes the y component of the vector field. \( fy(i, j) \) is the y component of the vector field at point \( (x(i), y(j)) \).
- \<opt_args>\ : This represents a sequence of statements \key1=value1, key2=value2,\ldots\ where \key1, key2,\ldots\ can be one of the following: arfact, rect, strf (see below).
- \arfact\ : an optional argument of type real which gives a scale factor for the display of the arrow heads on the plot (default value is 1.0).
- \rect\ : a vector \rect=[xmin, ymin, xmax, ymax] which gives the boundaries of the graphics frame to use.
- \strf\ : a string of length 3 "xyz" which has the same meaning as the \strf\ parameter of plot2d. The first character x has no effect with champ.

DESCRIPTION:
champ draws a 2D vector field. The length of the arrows is proportional to the intensity of the field.

If you want colored arrows with the color of the arrows depending on the intensity of the field, use champ1.

Enter the command champ() to see a demo.

EXAMPLE:
// using rect as plot boundaries
champ(-5:5,-5:5(rand(11,11),rand(11,11),1,[-10,-10,10,10],"011")
// using (x,y) to get boundaries
xbasc()
champ(-5:5,-5:5(rand(11,11),rand(11,11),2,[-10,-10,10,10],"021")

SEE ALSO: champ1 88, fchamp 99, plot2d 118

AUTHOR: J.Ph.C.

1.2.11 champ1 -------------------------------- 2D vector field plot with colored arrows

CALLING SEQUENCE:
champ1(x,y,fx,fy,[arfact,rect,strf])

PARAMETERS:

x,y : two vectors which define the grid.
fx : a matrix which describes the x component of the vector field. fx(i,j) is the x component of the vector field at point (x(i),y(j)).
fy : a matrix which describes the y component of the vector field. fy(i,j) is the y component of the vector field at point (x(i),y(j)).
arfact : an optional argument of type real which gives a scale factor for the display of the arrow heads on the plot (default value is 1.0).
rect : a vector rect=[xmin,ymin,xmax,ymax] which gives the boundaries of the graphics frame to use.
strf : a string of length 3 "xyz" which has the same meaning as the strf parameter of plot2d. The first character x has no effect with champ1.

DESCRIPTION:
champ1 draws a 2D vector field with colored arrows. The color of the arrows depends on the intensity of the field.

If you want arrows proportional to the intensity of the field, use champ.
Enter the command champ1() to see a demo.

EXAMPLE:

xset("use color",1)
champ1(-5:5,-5:5(rand(11,11),rand(11,11),2,[-10,-10,10,10],"021")

SEE ALSO: champ 87, fchamp 99, plot2d 118

AUTHOR: J.Ph.C.

1.2.12 chart --------------------------------- Nichols chart

CALLING SEQUENCE:
chart([flags])
chart(gain [,flags])
chart(gain,phase [,flags])

PARAMETERS:

gain : real vector (gains in DB)
**contour Function**

**phase**: real vector (phases (in degree))
**flags**: a list of at most 4 flags list(sup [,leg [,cm [,cphi]]])
**sup**: 1 indicates superposition on the previous plot 0 no superposition is done
**leg**: 1 indicates that legends are drawn, o: no legends
**cm**: color number (see plot2d) for gain curves
**cphi**: color number (see plot2d) for phase curves

**DESCRIPTION**:
plot the Nichols' chart.
The default values for **gain** and **phase** are respectively:

\[-12 -8 -6 -5 -4 -3 -2 -1.4 -1 -.5 0.25 0.5 0.7 1 1.4 2 2.3 3 4 5 6 8 12\]
\[-(1:10) , -(20:10:160)\]

**EXAMPLE**:

```scilab
s=poly(0,'s')
h=syslin('c',(sˆ2+2*0.9*10*s+100)/(sˆ2+2*0.3*10.1*s+102.01))
black(h,0.01,100,'(sˆ2+2*0.9*10*s+100)/(sˆ2+2*0.3*10.1*s+102.01)')
chart(list(1,0,2,3));
```

Another example:

```scilab
xbasc()
h1=h*syslin('c',(sˆ2+2*0.1*15.1*s+228.01)/(sˆ2+2*0.9*15*s+225))
black([h1;h],0.01,100,['h1';'h'])
chart([-8 -6 -4],80 120,list(1,0));
```

**1.2.13 colormap  using colormaps**

**DESCRIPTION**:
A colormap **cmap** is defined by a m x 3 matrix. m is the number of colors. Color number i is given as a 3-uple **cmap(i,1)**, **cmap(i,2)**, **cmap(i,3)** corresponding respectively to red, green and blue intensity between 0 and 1.

At the beginning, 32 colors are defined in the colormap. You can change the colormap by using **xset**("colormap", cmap).

Each color in the colormap has an id you have to use to specify color in most plot functions. To see the ids, use **xset()** or **xgetcolor()**.

You can come back to default colormap with **xset**("default").

**EXAMPLE**:

```scilab
m=228;
n= fix(3/8*m);
r=[(1:n)'/n; ones(m-n,1)];
g=[zeros(n,1); (1:n)'/n; ones(m-2*n,1)];
b=[zeros(2*n,1); (1:m-2*n)'/m-2*n)];
h=[r g b];
xset("colormap",h)
plot3d1()
xset("default")
```

**SEE ALSO**: addcolor 85, getcolor 106, xset 154
1.2.14  contour  ___________________________  level curves on a 3D surface

CALLING SEQUENCE :

contour(x,y,z,nz,[theta,alpha,leg,flag,ebox,zlev])
contour(x,y,z,nz,<opt_args>)

PARAMETERS :

x, y : two real row vectors of size n1 and n2.
z : real matrix of size (n1,n2), the values of the function.
nz : the level values or the number of levels.
   - If nz is an integer, its value gives the number of level curves equally spaced from zmin to zmax as follows:
     
     \[ z = z_{\text{min}} + (1:nz)*(z_{\text{max}}-z_{\text{min}})/(nz+1) \]

     Note that the zmin and zmax levels are not drawn (generically they are reduced to points) but they can be added with

     \[ [\text{im}, \text{jm}] = \text{find}(z == z_{\text{min}}); \quad \text{or zmax} \]
     \[ \text{plot2d}(x(\text{im})',y(\text{jm})',-9,"000") \]
   - If nz is a vector, nz(i) gives the value of the ith level curve. Note that it can be useful in order to see zmin and zmax level curves to add an epsilon tolerance: nz=[zmin+%eps,...,zmax-%eps].

<opt_args> : a sequence of statements key1=value1, key2=value2,... where keys may be theta, alpha, leg, flag, ebox, zlev (see below). In this case, the order has no special meaning.

theta, alpha : real values giving in degree the spherical coordinates of the observation point.
leg : string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".
flag : a real vector of size three flag=[mode,type,box].
mode : string (treatment of hidden parts).
mode=0  the hidden parts of the surface are removed and the surface is painted with color mode.
mode>0  the hidden parts of the surface are drawn.
mode=0  only the shadow of the surface is painted with color or pattern id -mode. Use xset() to see the meaning of the ids.
type : an integer (scaling).
type=0  the plot is made using the current 3D scaling (set by a previous call to param3d, plot3d,
        contour or plot3d1).
type=1  rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument ebox.
type=2  rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.
type=3  3d isometric with box bounds given by optional ebox, similarly to type=1
        type=4  3d isometric bounds derived from the data, to similarly type=2
        type=5  3d expanded isometric bounds with box bounds given by optional ebox, similarly to type=1
        type=6  3d expanded isometric bounds derived from the data, similarly to type=2
box : an integer (frame around the plot).
box=0  nothing is drawn around the plot.
box=1  unimplemented (like box=0).
box=2  only the axes behind the surface are drawn.
box=3  a box surrounding the surface is drawn and captions are added.
box=4  a box surrounding the surface is drawn, captions and axes are added.
ebox : used when type in flag is 1. It specifies the boundaries of the plot as the vector [xmin,xmax,ymin,ymax,zmin,zmax,2]
DESCRIPTION:

contour draws level curves of a surface \( z = f(x, y) \). The level curves are drawn on a 3D surface. The optional arguments are the same as for the function plot3d (except zlev) and their meanings are the same. They control the drawing of level curves on a 3D plot. Only flag(1) = mode has a special meaning.

- \( \text{mode}=0 \) : the level curves are drawn on the surface defined by \((x, y, z)\).
- \( \text{mode}=1 \) : the level curves are drawn on a 3D plot and on the plan defined by the equation \( z = z_{\text{lev}} \).
- \( \text{mode}=2 \) : the level curves are drawn on a 2D plot.

You can change the format of the floating point number printed on the levels by using \texttt{xset("fpf",string)} where \texttt{string} gives the format in C format syntax (for example \texttt{string="%.3f"}). Use \texttt{string=""} to switch back to default format.

Usually we use \texttt{contour2d} to draw levels curves on a 2D plot.

Enter the command \texttt{contour()} to see a demo.

EXAMPLE:

```scilab
// Function definition for a 3D surface
deff("[z]=surf(x,y)","z=sin(x)*cos(y)" );

z = feval(t,t,surf);

// Define the grid
x = linspace(min_x, max_x, n1);
y = linspace(min_y, max_y, n2);

// Define the level curves
levels = linspace(zmin, zmax, 10);

// Draw level curves
contour2d(x, y, z, levels);
```

SEE ALSO: \texttt{contour2d} 91, \texttt{fcontour} 100, \texttt{fcontour2d} 100, \texttt{plot3d} 123, \texttt{xset} 154

AUTHOR: J.Ph.C.

1.2.15 \texttt{contour2d} \texttt{----------------------} level curves of a surface on a 2D plot

CALLING SEQUENCE:

```scilab
contour2d(x, y, z, nz, [style, strf, leg, rect, nax])
contour2d(x, y, z, nz, <opt_args>)
```

PARAMETERS:

- \( x, y \) : two real row vectors of size \( n1 \) and \( n2 \) the grid.
- \( z \) : real matrix of size \((n1, n2)\), the values of the function.
- \( nz \) : the level values or the number of levels.
- If \( nz \) is an integer, its value gives the number of level curves equally spaced from \( z_{\text{min}} \) to \( z_{\text{max}} \) as follows:

\[
z = z_{\text{min}} + (1:nz)*(z_{\text{max}}-z_{\text{min}})/(nz+1)
\]

Note that the \( z_{\text{min}} \) and \( z_{\text{max}} \) levels are not drawn (generically they are reduced to points) but they can be added with

\[
[im,jm] = \text{find}(z == z_{\text{min}}); \quad \text{// or } z_{\text{max}}
\]

- Plot level curves at \( z_{\text{min}} \) and \( z_{\text{max}} \)

```scilab
plot2d(x(im)',y(jm)',-9,"000")
```

- If \( nz \) is a vector, \( nz(i) \) gives the value of the \( i \)th level curve.

\(<\text{opt_args}>\) : This represents a sequence of statements \texttt{key1=value1, key2=value2,...} where \texttt{key1, key2, ...} can be one of the following: \texttt{style, leg, rect, nax, strf} or \texttt{axesflag} and \texttt{frameflag} (see \texttt{plot2d})
contour2di Scilab Function

style,strf,leg,rect,nax : see plot2d. The argument style gives the dash styles or colors which are to be used for level curves. It must have the same size as the number of levels.

DESCRIPTION:
contour2d draws level curves of a surface z=f(x,y) on a 2D plot. The values of f(x,y) are given by the matrix z at the grid points defined by x and y.

You can change the format of the floating point number printed on the levels by using xset("fp",string) where string gives the format in C format syntax (for example string="%.3f"). Use string="" to switch back to default format.

The optional arguments style,strf,leg,rect,nax, can be passed by a sequence of statements key1=value1, key2=value2, ... where keys may be style,strf,leg,rect,nax. In this case, the order has no special meaning.

Use contour to draw levels curves on a 3D surface.

Enter the command contour2d() to see a demo.

EXAMPLE:
contour2d(1:10,1:10,rand(10,10),5,rect=[0,0,11,11])
// changing the format of the printing of the levels
xset("fp","%.2f")
xbasc()
contour2d(1:10,1:10,rand(10,10),5,rect=[0,0,11,11])

SEE ALSO: contour 90, fcontour 100, fcontour2d 100, contour2di 92, plot2d 118, xset 154

AUTHOR: J.Ph.C.

1.2.16 contour2di __________________ compute level curves of a surface on a 2D plot

CALLING SEQUENCE:

[xc,yc]=contour2di(x,y,z,nz)

PARAMETERS:

x, y : two real row vectors of size n1 and n2: the grid.
z : real matrix of size (n1,n2), the values of the function.
nz : the level values or the number of levels.
- If nz is an integer, its value gives the number of level curves equally spaced from zmin to zmax as follows:
  \[ z = z_{min} + (1:nz)*(z_{max}-z_{min})/(nz+1) \]

  Note that the zmin and zmax levels are not drawn (generically they are reduced to points) but they can be added with

  \[ [im,jm] = find(z == zmin); \quad // or zmax \]
  \[ plot2d(x(im)',y(jm)',-9,"000") \]

- If nz is a vector, nz(i) gives the value of the ith level curve.

xc, yc : vectors of identical sizes containing the contours definitions. See below for details.

DESCRIPTION:
contour2di computes level curves of a surface z=f(x,y) on a 2D plot. The values of f(x,y) are given by the matrix z at the grid points defined by x and y.

xc(1) contains the level associated with first contour path, yc(1) contains the number N1 of points defining this contour path and (xc(1+(1:N1)),yc(1+(1:N1))) contain the coordinates of the paths points. The second path begin at xc(2+N1) and yc(2+N1) and so on.

EXAMPLE:

Scilab Group sept 2000 92
```plaintext
[xc,yc]=contour2di(1:10,1:10,rand(10,10),5);
k=1; n=yc(k); c=1;
while k+yc(k)<size(xc,'*')
    n=yc(k);
    plot2d(xc(k+(1:n)),yc(k+(1:n)),c)
    c=c+1;
    k=k+n+1;
end
```

**SEE ALSO:** contour 90, fcontour 100, fcontour2d 100, contour2d 91, plot2d 118, xset 154

**AUTHOR:** J.Ph.C.

### 1.2.17 contourf _________________ filled level curves of a surface on a 2D plot

**CALLING SEQUENCE:**
```
contourf(x,y,z,nz,[style,strf,leg,rect,nax])
```

**PARAMETERS:**
- `x,y`: two real row vectors of size n1 and n2: the grid.
- `z`: real matrix of size (n1,n2), the values of the function.
- `nz`: the level values or the number of levels.
  - If `nz` is an integer, its value gives the number of level curves equally spaced from `zmin` to `zmax` as follows:
    
    \[
    z = z_{\text{min}} + (1:nz)*(z_{\text{max}}-z_{\text{min}})/(nz+1)
    \]
  
  Note that the `zmin` and `zmax` levels are not drawn (generically they are reduced to points) but they can be added with
  
  ```plaintext
  [im,jm] = find(z == zmin); // or zmax
  plot2d(x(im)',y(jm)',-9,"000")
  ```
  
  - If `nz` is a vector, `nz(i)` gives the value of the ith level curve.

  **style,strf,leg,rect,nax**: see plot2d. The argument `style` gives the dash styles or colors which are to be used for level curves. It must have the same size as the number of levels.

**DESCRIPTION:**
`contourf` paints surface between two consecutive levels curves of a surface \( z = f(x,y) \) on a 2D plot. The values of \( f(x,y) \) are given by the matrix `z` at the grid points defined by `x` and `y`.

You can change the format of the floating point number printed on the levels by using `xset("fpf","string")` where `string` gives the format in C format syntax (for example `string="%f")`. Use `string=""` to switch back to default format.

Enter the command `contour2d()` to see a demo.

**EXAMPLE:**
```
contourf(1:10,1:10,rand(10,10),5,1:5,"011",",[0,0,11,11])
```

**SEE ALSO:** contour 90, fcontour 100, fcontour2d 100, contour2di 92, plot2d 118, xset 154

**AUTHOR:** J.Ph.C.
### 1.2.18 drawrect

**Calling Sequence:**

```
final_rect=dragrect(initial_rect)
```

**Parameters:**

- `initial_rect`: 4xn matrix containing the initial rectangles definition. Each column contains [x_left; y_top; width; height]. If only one rectangle is present the initial_rect may also be a vector.
- `final_rect`: a rectangle defined by [x_left, y_top, width, height]

**Description:**

`dragrect` tracks one or more rectangles anywhere on the screen. The 4xn matrix rect defines the rectangles. Each column of `initial_rect` must contain the initial rectangle position as [left;top;width;height] values. When a button is clicked `dragrect` returns the final rectangles definition in `final_rect`.

**Example:**

```scilab
xsetech(frect=[0,0,100,100])
r=dragrect([10;10;30;10])
xrect(r)
```

**See Also:** `xrect 151`, `xrects 151`, `xclick 139`, `xmouse ??`

### 1.2.19 drawaxis

**Calling Sequence:**

```
drawaxis([options])
```

// options: x,y,dir,sub_int,fontsize,format_n,seg,textcolor,ticscolor,tics

**Parameters:**

- `dir=string`: used to specify the tics direction. string can be chosen among 'u','r','d','l' and 'l' is the default value. the values 'u','r','d','l' stands respectively for up, right, down, left
- `tics=string`: A flag which describes how the tics are given. string can be chosen among 'v','r', and 'i', and, 'v' is the default value
- `x,y`: two vectors which give tics positions.
- `val= string matrix`: A string matrix, which, when given, gives the string to be drawn along the axis at tics positions.
- `fontsize=int`: specifies the fontsize to use for displaying values along the axis. Default value is -1 which stands for current fontsize
- `format_n=string`: format to use for displaying numbers along the axis
- `seg= 1 or 0`: A flag which controls the display of the base segment of the axis (default value is 1).
- `sub_int=integer`: an integer which gives the number of sub-intervals to draw between large tics.
- `textcolor=integer`: specify the color to use for displaying values along the axis. Default value is -1 which stands for current color.
- `ticscolor=integer`: specify the color to use for tics drawing. Default value is -1 which stands for current color.

**Description:**

`drawaxis` is used to draw an axis in vertical or horizontal direction. the direction of the axis is given by `dir` dir = 'u' or 'd' gives a horizontal axis with tics going up ('u') or down ('d'). dir = 'r' or 'l' give a vertical axis with tics going right ('r') or left ('l').
x and y give the axis tics positions. If the axis is horizontal then y must be a scalar or can be omitted and x is a Scilab vector. The meaning of x is controlled by tics.
If tics='v' then x gives the tics positions along the x-axis.
If tics='r' then x must be of size 3: x=[xmin,xmax,n] and n gives the number of intervals.
If tics='i' then x must be of size 4: x=[k1,k2,a,n]. then xmin=k1*10^a, xmax=k2*10^a and n gives the number of intervals.
If y is omitted then the axis will be positioned at the top of the frame if dir='u' or at the bottom if dir='d'. By default, numbers are drawn along the axis. They are drawn using a default format which can be changed with format. It is also possible to display given strings and not numbers, this is done if val is provided. The size of val must match the number of tics.

EXAMPLE :

plot2d(1:10,1:10,1,"020")
// horizontal axis
drawaxis(x=2:7,y=4,dir='u',tics='v')
// horizontal axis on top of the frame
drawaxis(x=2:7,dir='u',tics='v')
// horizontal axis at the bottom of the frame
drawaxis(x=2:7,dir='d',tics='v')
// horizontal axis given by a range
drawaxis(x=[2,7,3],y=4,dir='d',tics='r')
// vertical axis
drawaxis(x=4,y=2:7,dir='r',tics='v')
drawaxis(x=2,y=[2,7,3],dir='l',tics='r')

drawaxis(y=2:7,dir='r',tics='v')

// horizontal axis with strings displayed at tics positions
drawaxis(x=2:7,y=8,dir='u',tics='v',val='A'+string(1:6));
// vertical axis with strings displayed at tics positions
drawaxis(x=8,y=2:7,dir='r',tics='v',val='B'+string(1:6));

// horizontal axis given with a 'i' range.
drawaxis(x=[2,5,0,3],y=9,dir='u',tics='i');
drawaxis(x=9,y=[2,5,0,3],dir='r',tics='i',sub_int=5);

// horizontal axis again
drawaxis(x=2:7,y=4,dir='u',tics='v',fontsize=10,textcolor=9,ticscolor=7,seg=0,sub_int=2

AUTHOR: J.Ph.C.

1.2.20 driver select a graphics driver

CALLING SEQUENCE:

driver(driver_name)
current_driver=driver()

PARAMETERS:

driver_name : string, driver to be selected.

Scilab Group April 1993 95
errbar Scilab Function

DESCRIPTION:
This function is used to select a graphics driver, or with no arguments to get the current graphics driver name. Most of the time, a user can ignore this function and change the driver by calling high level functions such as xbas or xbasimp. The selected driver can be one of the followings:

"X11" : output to the screen of the computer.
"Pos" : output into Postscript format.
"Rec" : output to the screen of the computer with recording of all the graphics commands. This is the default driver.
"Fig" : output into XFig format. Clipping of objects is not provided in XFig.
"GIF" : output into Gif format (beta test driver written by Tom Leitner (http://wiis.tu-graz.ac.at/people/tom.html).

Note that line thickness is not handled yet.

SEE ALSO: xtape 159, xbas 137, xbasimp 137

AUTHOR: J.Ph.C.

1.2.21 edit_curv ___________________________ interactive graphic curve editor

CALLING SEQUENCE:

[x,y,ok,gc] = edit_curv(y)
[x,y,ok,gc] = edit_curv(x,y)
[x,y,ok,gc] = edit_curv(x,y,job)
[x,y,ok,gc] = edit_curv(x,y,job,tit)
[x,y,ok,gc] = edit_curv(x,y,job,tit,gc)

PARAMETERS:

x : vector of x coordinates
y : vector of y coordinates
job : a character string formed by one to three of the characters 'a','x','y'
'a' : to add points to the edited curve
'x' : to modify x coordinates of the edited curve points
'y' : to modify y coordinates of the edited curve points:
points : a vector of three character strings which give the curve legend
gc : a list of graphic window parameters: gc=list(rect,nax)
rect : bounds of the graphics (see plot2d for details)
nax : graduation parameters (see plot2d for details); indicator if ok==%t user as returned with 'ok' menu else user as returned with 'abort' menu : list (graphical objects created under edit_curv

DESCRIPTION:
edit_curv is an interactive graphic curve editor. To add a new point simply click at the desired location, the added point will be connected to the nearest end-point. to move a point click on it, drag the mouse to the new position and click to set the new position

AUTHOR: Serge Steer

1.2.22 errbar ___________________________ add vertical error bars on a 2D plot

CALLING SEQUENCE:

Scilab Group April 1993 96
eval3dp Scilab Function

errbar(x, y, em, ep)

PARAMETERS:

x, y, em, ep : four matrices of the same size.

DESCRIPTION:

errbar adds vertical error bars on a 2D plot. x and y have the same meaning as in plot2d. em(i, j) and ep(i, j) stands for the error interval on the value y(i, j): [y(i, j) - em(i, j), y(i, j) + ep(i, j)]. Enter the command errbar() to see a demo.

EXAMPLE:

t=[0:0.1:2*%pi]';
y=[sin(t) cos(t)]; x=[t t];
plot2d(x,y)
errbar(x,y,0.05*ones(x),0.03*ones(x))

SEE ALSO: plot2d

AUTHOR: J.Ph.C.

1.2.23 eval3d ______________________ values of a function on a grid

CALLING SEQUENCE:

[z]=eval3d(fun, x, [y])

PARAMETERS:

fun : function accepting vectors as arguments.
x, y : 2 vectors of size (1,n1) and (1,n2). (default value for y : y=x).
z : matrix of size (n1,n2).

DESCRIPTION:

This function returns a matrix z(n1, n2). z(i, j)=fun(x(i), y(j)). If the function fun doesn't accept arguments of type vector use the primitive feval.

EXAMPLE:

x=-5:5; y=x;
def('[z]=f(x,y)','[z= x.*y]');
z=eval3d(f, x, y);
plot3d(x, y, z);
//
def('[z]=f(x,y)','[z= x*y]');
z=feval(x, y, f);
plot3d(x, y, z);

SEE ALSO: feval

AUTHOR: Steer S.

Scilab Group September 1994
1.2.24 eval3dp ___________________________ compute facets of a 3D surface

Calling Sequence:

\[ [x,y,z] = \text{eval3dp}(\text{fun}, \text{p1}, \text{p2}) \]

Parameters:

- \( x, y, z \) : matrices of size \((4,n-1\times m-1)\). \( x(:,i), y(:,i), z(:,i) \) are respectively the x-axis, y-axis and z-axis coordinates of the 4 points of the \( i \)th four sided facet.
- \( \text{fun} \) : a Scilab function.
- \( \text{p1} \) : a vector of size \( n \).
- \( \text{p2} \) : a vector of size \( m \).

Description:

\( \text{eval3dp} \) computes a four sided facets representation of a 3D surface defined by the function \( \text{fun} \).
\( \text{fun}(\text{p1}, \text{p2}) \) computes the x-axis, y-axis and z-axis coordinates of the corresponding points on the surface, as \([x(i), y(i), z(i)] = \text{fun}(\text{p1}(i), \text{p2}(i))\). This is used for efficiency.

Example:

\[
\begin{align*}
\text{p1} &= \text{linspace}(0,2*\%\pi,10); \\
\text{p2} &= \text{linspace}(0,2*\%\pi,10); \\
\text{deff}("[x,y,z]=\text{scp}(\text{p1}, \text{p2})","["x=\text{p1}.*\sin(\text{p1}).\cos(\text{p2})";.."y=\text{p1}.*\cos(\text{p1}).\cos(\text{p2})";.."z=\text{p1}.*\sin(\text{p2})"]")
\end{align*}
\]

\([x,y,z] = \text{eval3dp}(\text{scp}, \text{p1}, \text{p2})
\]

plot3d(x,y,z)

See Also: \( \text{genfac3d 105}, \ text{plot3d 123} \)

1.2.25 evans ______________________________ Evans root locus

Calling Sequence:

\( \text{evans}(H [, \text{kmax}]) \)

Parameters:

- \( H \) : list (linear system \( \text{syslin} \) )
- \( \text{kmax} \) : real (maximum gain desired for the plot)

Description:

\( \text{evans} \) gives the Evans root locus for a linear system in state-space or transfer form \( H(s)(\text{syslin} \ list) \). This is the locus of the roots of \( 1+k*H(s)=1+k*N(s)/D(s) \), in the complex plane. For a selected sample of gains \( k \leq \text{kmax} \), the imaginary part of the roots of \( D(s)+k*N(s) \) is plotted vs the real part.

To obtain the gain at a given point of the locus you can simply execute the following instruction: \( K=-1/\text{real(horner}(h,[1,\%i]) \) and click the desired point on the root locus. If the coordinates of the selected point are in the real 2 x 1 vector \( P=\text{locate}(1) \) this \( k \) solves the equation \( k*N(w)+D(w)=0 \) with \( w=P(1)+\%i*P(2)=[1,\%i]*P \).

Example:

Scilab Group

April 1993
H=syslin('c',352*poly(-5,'s')/poly([0,0,200,200,25,1],'s','c'));
evens(H,100)
P=3.0548543 - 8.8491842*%i; //P=selected point
k=-1/real(horner(H,P));
Ns=H('num');Ds=H('den');
roots(Ds+k*Ns) //contains P as particular root
// Another one
xbasc();s=poly(0,'s');n=1+s;
d=real(poly([-1 -2 -%i %i],'s'));
evens(n,d,100);
//
xbasc();n=real(poly([0.1-%i 0.1+%i,-10],'s'));
evens(n,d,80);

SEE ALSO:  kpure 342,  krac2 343,  locate 112

1.2.26  fac3d 3D plot of a surface (obsolete)

CALLING SEQUENCE :

fac3d(x,y,z,[theta,alpha,leg,flag,ebox])
fac3d1(x,y,z,[theta,alpha,leg,flag,ebox])

DESCRIPTION :
These functions are obsolete and have been replaced by plot3d and plot3d1.
SEE ALSO:  plot3d 123,  plot3d1 125

1.2.27  fchamp  direction field of a 2D first order ODE

CALLING SEQUENCE :

fchamp(f,t,xr,yr,[arfact,rect,strf])
fchamp(x,y,xr,yr,<opt_args>)

PARAMETERS :

f : An external (function or character string) or a list which describes the ODE.
- It can be a function name f, where f is supposed to be a function of type y=f(t,x,[u]). f returns
  a column vector of size 2, y, which gives the value of the direction field f at point x and at time t.
- It can also be an object of type list, list(f,u1) where f is a function of type y=f(t,x,u) and
  u1 gives the value of the parameter u.
t : The selected time.
xr,yr : Two row vectors of size n1 and n2 which define the grid on which the direction field is computed.
<opt_args> : This represents a sequence of statements key1=value1, key2=value2.... where
  key1,key2,... can be one of the following: arfact, rect, strf (see below).
arfact,rect,strf : Optional arguments, see champ.

DESCRIPTION :
fchamp is used to draw the direction field of a 2D first order ODE defined by the external function f.
Note that if the ODE is autonomous, argument t is useless, but it must be given.

Enter the command fchamp() to see a demo.
fcontour2d Scilab Function

```scilab
deff("[xdot] = derpol(t,x)","...
    "xd1 = x(2);
    "xd2 = -x(1) + (1 - x(1)**2)*x(2);
    "xdot = [ xd1 ; xd2 ]")
xf= -1:0.1:1;
yf= -1:0.1:1;
fchamp(derpol,0,xf,yf)
xbasc()
fchamp(derpol,0,xf,yf,1,[-2,-2,2,2],"011")

SEE ALSO: champ 87, champ1 88

AUTHOR: J.Ph.C.
```

### 1.2.28 fcontour ____________ level curves on a 3D surface defined by a function

**CALLING SEQUENCE:**

```scilab
fcontour(xr,yr,f,nz,[theta,alpha,leg,flag,ebox,zlev])
fcontour(xr,yr,f,nz,<opt_args>)
```

**PARAMETERS:**

- `xr, yr`: two real row vectors of size n1 and n2.
- `f`: is an external which defines the surface \( z=f(x,y) \). It is first computed on the grid specified by `xr, yr`. Then, control is passed to the routine `contour`.
- `nz`: see `contour`.
- `theta, alpha, leg, flag, ebox, zlev`: see `contour`.
- `<opt_args>`: see `contour`.

**DESCRIPTION:**

Draws level curves of a surface \( z=f(x,y) \). The level curves are drawn on a 3D surface. The surface is given by the external function `f`. See `contour`.

The optional arguments `theta, alpha, leg, flag, ebox, zlev`, can be passed by a sequence of statements `key1=value1, key2=value2`, `flag, ebox, zlev`. In this case, the order has no special meaning.

Enter the command `fcontour()` to see a demo.

**EXAMPLE:**

```scilab
deff("[z]=surf(x,y)","z=sin(x)*cos(y)");
t=%pi*[-10:10]/10;
fcontour(t,t,surf,10)
xbasc();fcontour(t,t,surf,10,ebox=[-4 4 -4 4 -1 1],zlev=-1,flag=[0 1 4])
```

**SEE ALSO:** contour 90, contour2d 91, fcontour2d 100

**AUTHOR:** J.Ph.C.

### 1.2.29 fcontour2d ___ level curves of a surface defined by a function on a 2D plot

**CALLING SEQUENCE:**

Scilab Group April 1993 100
fcontour2d(xr,yr,f,nz,[style,strf,leg,rect,nax])
fcontour2d(xr,yr,f,nz,<opt_args>)

PARAMETERS :

xr,yr : two real row vectors of size n1 and n2.
f : is an external which defines the surface z=f(x,y). It is first computed on the grid specified by
xr,yr. Then, control is passed to the routine contour2d.
nz : the level values or the number of levels.
   - If nz is an integer, its value gives the number of level curves equally spaced from zmin to zmax.
   - If nz is a vector, nz(i) gives the value of the ith level curve.

<opt_args> : This represents a sequence of statements key1=value1, key2=value2,... where
key1,key2,... can be one of the following: style, leg, rect, nax, strf or axesflag and frameflag
(see plot2d)
[style,strf,leg,rect,nax] : see contour2d.

DESCRIPTION :
Draws level curves of a surface z=f(x,y). The level curves are drawn on a 2D plot. The surface is given
by the external function f. See contour2d.

Enter the command fcontour2d() to see a demo.

EXAMPLE :

deff('z=surf(x,y)','z=x^4-y^4')
x=-3:0.1:3;
y=x;
fcontour2d(x,y,surf,10);

SEE ALSO:  contour 90, contour2d 91, fcontour 100

AUTHOR : J.Ph.C.

1.2.30  fec  --------------  contour level of a function defined on a triangular mesh

CALLING SEQUENCE :

fec(x,y,triangles,func,[strf,leg,rect,nax,zminmax,colminmax])

PARAMETERS :

x,y : two vectors of size n. (x(i),y(i)) gives the coordinates of node i
func : a vector of size n : func(i) gives the value of the function for which we want the level curves.
triangles : is a [Ntr,5] matrix. Each line of triangles specifies a triangle of the mesh
   triangle(j) = [number,node1,node2,node3,flag]. node1,node2,node3 are the
   number of the nodes which constitutes the triangle. number is the number of the triangle and flag
   is an integer not used in the fec function
strf,leg,rect,nax : see plot2d
zminmax : useful only for animation with fec, zminmax is a vector of size 2 [zmin zmax] which gives
   the z values associated with the first and the last color (of the current colormap). (More exactly if
   the colormap have nc colors and if we note dz = (zmax-zmin)/nc, then the part of the triangulation
   where zmin + (i-1)dz <= z < zmin + i dz is filled with the color i). By default zmin = min(func) and zmax
   = max(func). If you want to do an animation with func values that varie in time, take for zmin and
   zmax the global minimum and maximum or something close. CAUTION : for func values greater
   than zmax the last color is used and for func values less than zmin this is the first color (so you don't
   see that the zminmax levels are crossed).
fplot2d Scilab Function

**DESCRIPTION:**

See the demo files demos/fec.

-fec.ex1 is a simple demo file in which a mesh and a function on that mesh is completely built in Scilab syntax

-fec.ex2 is an example for which the mesh and the function value where computed by an external mesh builder (amdba type mesh) and an external program. A set of macros (provided in file macros.sci) can be used to read the data files in Scilab and plot the results.

**SEE ALSO:** Sfgrayplot 84, Sgrayplot 84

### 1.2.31 fgrayplot _________ 2D plot of a surface defined by a function using colors

**CALLING SEQUENCE:**

fgrayplot(x,y,f,[strf,rect,nax])
fgrayplot(x,y,f,<opt_args>)

**PARAMETERS:**

x,y : real row vectors.
f : external of type y=f(x,y).
<opt_args> : This represents a sequence of statements key1=value1, key2=value2,... where key1, key2,... can be one of the following: rect, nax, strf or axesflag and frameflag (see plot2d).
strf,rect,nax : see plot2d.

**DESCRIPTION:**

fgrayplot makes a 2D plot of the surface given by $z=f(x,y)$ on a grid defined by $x$ and $y$. Each rectangle on the grid is filled with a gray or color level depending on the average value of $z$ on the corners of the rectangle.

Enter the command fgrayplot() to see a demo.

**EXAMPLE :**

t=-1:0.1:1;
deff("[z]=surf(x,y)","z=x**2+y**2")
fgrayplot(t,t,surf,rect=[-2,-2,2,2])

**SEE ALSO:** grayplot 109, plot2d 118, Sgrayplot 84, Sfgrayplot 84

### 1.2.32 fplot2d ______________________ 2D plot of a curve defined by a function

**CALLING SEQUENCE:**

fplot2d(xr,f,[style,strf,leg,rect,nax])
fplot2d(xr,f,<opt_args>)

**PARAMETERS:**

xr : vector.
fplot3d1 Scilab Function

\[ f : \text{external of type } y=f(x) \] i.e. a scilab function or a dynamically linked routine referred to as a string.

\[ \text{style, strf, leg, rect, nax} : \text{see plot2d} \]

\text{DESCRIPTION :}

fplot2d plots a curve defined by the external function \( f \). The curve is approximated by a piecewise linear interpolation using the points \((x_r(i), f(x_r(i)))\). The values of \( f(x) \) are obtained by \texttt{feval(xr,f)}.

Enter the command \texttt{fplot2d()} to see a demo.

\text{EXAMPLE :}

\begin{verbatim}
deff(["y"]="f(x)","y=sin(x)+cos(x)")
x=[0:0.1:10]*%pi/10;
fplot2d(x,f)
xbasc();
fplot2d(1:10,'parab')
\end{verbatim}

SEE ALSO: \texttt{plot2d} \texttt{118}, \texttt{feval} \texttt{41}, \texttt{paramfplot2d} \texttt{117}

AUTHOR : J.Ph.C.

1.2.33 fplot3d ________________ 3D plot of a surface defined by a function

\text{CALLING SEQUENCE :}

\texttt{fplot3d(xr,yr,f,[theta,alpha,leg,flag,ebox])}
\texttt{fplot3d(xr,yr,f,<opt_args>)}

\text{PARAMETERS :}

\( xr \) : row vector of size \( n1 \).
\( yr \) : row vector of size \( n2 \).
\( f \) : external of type \( z=f(x,y) \).
\( \theta, \alpha, \text{leg, flag, ebox} : \text{see plot3d.} \)
\( <\text{opt_args}> : \text{see plot3d.} \)

\text{DESCRIPTION :}

fplot3d plots a surface defined by the external function \( f \) on the grid defined by \( xr \) and \( yr \).

Enter the command \texttt{fplot3d()} to see a demo.

\text{EXAMPLE :}

\begin{verbatim}
deff(’z=f(x,y)’,’z=x^4-y^4’)
x=-3:0.2:3 ;y=x ;
xbasc() ;fplot3d(x,y,f,alpha=5,theta=31)
\end{verbatim}

SEE ALSO: \texttt{plot3d} \texttt{123}

AUTHOR : J.Ph.C.

1.2.34 fplot3d1 ___ 3D gray or color level plot of a surface defined by a function

\text{CALLING SEQUENCE :}

\texttt{fplot3d1(xr,yr,f,[theta,alpha,leg,flag,ebox])}
\texttt{fplot3d1(xr,yr,f,<opt_args>)}

Scilab Group April 1993 103
PARAMETERS:

xr : row vector of size n1.
yr : row vector of size n2.
f : external of type z=f(x,y).
theta, alpha, leg, flag, ebox : see plot3d.
<opt_args> : see plot3d.

DESCRIPTION:

fplot3d1 plots a 3D gray or color level plot of a surface defined by the external function f on the grid defined by xr and yr.

Enter the command fplot3d1() to see a demo.

EXAMPLE:

deff(’z=f(x,y)’,’z=x^4-y^4’)
x=-3:0.2:3 ; y=x ;
xbasc() ; fplot3d1(x,y,f,alpha=5,theta=31)

SEE ALSO: plot3d1 125

AUTHOR: J.Ph.C.

1.2.35 gainplot .......................................................... magnitude plot

CALLING SEQUENCE:

gainplot(sl,fmin,fmax [,step] [,comments] )
gainplot(frq,db,phi [,comments])
gainplot(frq, repf [,comments])

PARAMETERS:

sl : list(syslin SIMO linear system).
fmin, fmax : real scalars (frequency interval).
step : real (discretization step (logarithmic scale))
comments : string
frq : matrix (row by row frequencies)
db,phi : matrices (magnitudes and phases corresponding to frq)
repf : complex matrix. One row for each frequency response.

DESCRIPTION:

Same as Bode but plots only the magnitude.

EXAMPLE:

s=poly(0,’s’)
h=syslin(’c’,(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
gainplot(h,0.01,100,’(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)’)
xbasc()
h1=h*syslin(’c’,(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
gainplot([h1;h],0.01,100, [’h1’;’h’])

SEE ALSO: bode 86, black 85, nyquist 115, freq 339, repfreq 355, g_margin 340, p_margin 352
1.2.36  genfac3d  compute facets of a 3D surface

CALLING SEQUENCE :

[xx,yy,zz]=genfac3d(x,y,z,[mask])

PARAMETERS :

xx,yy,zz : matrices of size (4,n-1xm-1). xx(:,i) , yy(:,i) and zz(:,i) are respectively
the x-axis, y-axis and z-axis coordinates of the 4 points of the ith four sided facet.
x : x-axis coordinates vector of size m.
y : y-axis coordinates vector of size n.
z : matrix of size (m,n). z(i,j) is the value of the surface at the point (x(i),y(j)).
mask : boolean optional matrix with same size as z used to select the entries of z to be represented by
facets.

DESCRIPTION :

genfac3d computes a four sided facets representation of a 3D surface z=f(x,y) defined by x, y and
z.

EXAMPLE :

t=[0:0.3:2*pi]'; z=sin(t).*cos(t');
[xx,yy,zz]=genfac3d(t,t,z);
plot3d(xx,yy,zz)

SEE ALSO :  eval3dp, plot3d

1.2.37  geom3d  projection from 3D on 2D after a 3D plot

CALLING SEQUENCE :

[x,y]=geom3d(x1,y1,z1)

PARAMETERS :

x1,y1,z1 : real vectors of the same size (points in 3D).
x,y : real vectors of the same size as x1,y1 and z1.

DESCRIPTION :

After having used a 3D plot function such as plot3d, plot3d1 or param3d, geom3d gives the mapping
between a point in 3D space (x1(i),y1(i),z1(i)) and the corresponding point (x(i),y(i))
in the projected 2D plan. Then all the 2D graphics primitives working on (x,y) can be used for superpo-
sition on the 3D plot.

EXAMPLE :

deff("[z]=surf(x,y)","z=sin(x)*cos(y)")
t=%pi*(-10:10)/10;
// 3D plot of the surface
fplot3d(t,t,surf,35,45,"X@Y@Z")
// now (t,t,sin(t).*cos(t)) is a curve on the surface
// which can be drawn using geom3d and xpoly
[x,y]=geom3d(t,t,sin(t).*cos(t));
xpoly(x,y,"lines")
// adding a comment
[x,y]=geom3d([0,0],[0,0],[5,0]);
xsegs(x,y)
xstring(x(1),y(1),"point (0,0,0)"

AUTHOR : J.Ph.C.
1.2.38  getcolor  

dialog to select colors in the current colormap

CALLING SEQUENCE :

c=getcolor(title,[cini])
c=getcolor()

PARAMETERS :

title : string, dialog title.
cini : vector of initial selected color ids. Default value is xget("pattern").
c : vector of selected color ids, or [] if the user has clicked on the "Cancel" button.

DESCRIPTION :

getcolor opens a dialog choice box with as many palettes as cini vector size. Palettes depend on the current colormap.

SEE ALSO :  xset 154,  getmark 107,  getfont 106

1.2.39  getfont  

dialog to select font

CALLING SEQUENCE :

[fId,fSize]=getfont()
[fId,fSize]=getfont(str)

PARAMETERS :

str : character (e.g. "a")
fId : integer, the number of the selected font
fSize : integer, the size of the selected font

DESCRIPTION :

getfont opens a graphic window to select a font. Example of use: [fId,fSize]=getfont().
xset("font",fId,fSize).plot2d(0,0,rect=[0 0 10 10],axesflag=0):xstring(5,5,"string").

SEE ALSO :  xset 154,  getmark 107

1.2.40  getlinestyle  

dialog to select linestyle

CALLING SEQUENCE :

k=getlinestyle()

PARAMETERS :

k : integer, selected linestyle or [] if the "Cancel" button has been clicked.

DESCRIPTION :

getlinestyle opens a graphic window to select a line style. Example: k=getlinestyle() plot2d(1:10,10,style=k).

SEE ALSO :  xset 154
1.2.41 getmark ________________________ dialog to select mark (symbol)

CALLING SEQUENCE:
[mark,mrkSize]=getmark()

PARAMETERS:
mark : integer, the number of the selected mark
mrkSize : integer, the size of the selected mark

DESCRIPTION:
getmark opens a graphic window to select a mark (symbol). Usage: [mark,mrkSize]=getmark().
xset("mark size",mrkSize).plot2d(x,y,style=mark).
SEE ALSO: xset 154, getfont 106

1.2.42 getsymbol ________________________ dialog to select a symbol and its size

CALLING SEQUENCE:
c=getsymbol([title])

PARAMETERS:
title : string, dialog title.
c : vector of size 2 [n,sz].

DESCRIPTION:
getsymbol opens a dialog choice box with title title if given where the user can select a symbol and its size. getsymbol returns the id of the mark n and the id of its size sz.
SEE ALSO: xset 154

1.2.43 gr_menu __________________________ simple interactive graphic editor

CALLING SEQUENCE:
[sd1]=gr_menu([sd,flag,no_frame])

PARAMETERS:
sd : list (output of gr_menu), or vector of length four [xmin,ymin,xmax,ymax] (boundaries of the plot).
sd1 : list (graphical objects created under gr_menu)
flag,noframe : integers with 0, 1 value. Use flag=1 for non interactive mode (i.e to redraw saved gr_menu graphics) and no_frame=1 to avoid a frame around gr_menu graphics.

DESCRIPTION:
gr_menu is a simple interactive graphic editor. When you execute gr_menu(), three new menus, Objects, Settings and Edit are added to the current graphics window. Use the item Exit of menu Edit to exit gr_menu.

The created graphics are saved as a list which can be given to gr_menu as an entry value.

[sd]=gr_menu([xmin,ymin,xmax,ymax]) : enters gr_menu with a given frame
[sd]=gr_menu(); : enters gr_menu with the frame [0 0 100 100].
[sd]=gr_menu(sd) : redraws the graphics stored in sd and enters interactive mode
[sd]=gr_menu(sd,1) : only draws the graphics stored in sd.
[sd]=gr_menu(sd,1,1) : only draws the graphics stored in sd and no frame is added.

AUTHOR : S.S. & J.Ph.C.
1.2.44 graduate----------------- pretty axis graduations

CALLING SEQUENCE:

\[ [\xi, \xa, \np] = \text{graduate}(\ \text{xmi}, \ \text{xma}, \text{n1}, \text{n2}) \]
\[ [\xi, \xa, \np] = \text{graduate}(\ \text{xmi}, \ \text{xma}) \]

PARAMETERS:

\text{xmi, xma} : real scalars
\text{n1, n2} : integers with default values 3, 10
\text{xi, xa} : real scalars
\text{np} : integer

DESCRIPTION:

graduate looks for the minimum interval \([\xi, \xa]\) and a number of tics \(\np\) such that:
\[ \xi / \text{BO} = \text{xmi} / \text{BO} = \text{xma} / \text{BO} = \xa \]
\[ \xa - \xi / \np = k(10^n), k \text{ in } [1, 3, 5] \text{ for an integer } n \]
\text{n1} < \text{np} < \text{n2}

EXAMPLE:

\[ y = (0:0.33:145.78)'; \]
\[ \text{xbasc();plot2d1('enn',0,y)} \]
\[ [\text{ymn, ymx, np}] = \text{graduate}((\text{mini}(y), \text{ maxi}(y)) \]
\[ \text{rect=[1, ymn, prod(size(y)), ymx]}; \]
\[ \text{xbasc();plot2d1('enn',0,y,1,'011',' ',rect,[10,3,10,np])} \]

SEE ALSO: xsetech 155, plot2d1 118

AUTHOR: S. Steer 1992

1.2.45 graycolormap----------------- linear gray colormap

CALLING SEQUENCE:

\text{cmap} = \text{graycolormap}(\text{n})

PARAMETERS:

\text{n} : integer \geq 1, the colormap size.
\text{cmap} : matrix with 3 columns \([R, G, B]\).

DESCRIPTION:

graycolormap computes a colormap with \text{n} gray colors varying linearly from black to white.

EXAMPLE:

\[ \text{xset("colormap",graycolormap(32))} \]
\text{plot3d1()} \]

SEE ALSO: colormap 89, hotcolormap 111, xset 154

Scilab Group April 1993 108
1.2.46  grayplot  2D plot of a surface using colors

CALLING SEQUENCE :

grayplot (x, y, z, [strf, rect, nax])
greyplot (x, y, z, <opt_args>)

PARAMETERS :

x, y : real row vectors of size n1 and n2.
z : real matrix of size (n1,n2). z(i, j) is the value of the surface at the point (x(i),y(j)).
<opt_args> : This represents a sequence of statements key1=value1, key2=value2,... where
   key1, key2, ... can be one of the following: rect, nax, strf or axesflag and frameflag (see
   plot2d).
strf, rect, nax : see plot2d.

DESCRIPTION :

grayplot makes a 2D plot of the surface given by z on a grid defined by x and y. Each rectangle
on the grid is filled with a gray or color level depending on the average value of z on the corners of the
rectangle.

Enter the command grayplot() to see a demo.

EXAMPLE :

x=-10:10; y=-10:10; m=rand(21,21);
greyplot(x, y, m, rect=[-20,-20,20,20])
t=-%pi:0.1:%pi; m=sin(t)'*cos(t);
xbasc()
greyplot(t,t,m)

SEE ALSO:  fgrayplot 102, plot2d 118, Sgrayplot 84, Sfgrayplot 84

AUTHOR : J.Ph.C.

1.2.47  graypolarplot  Polar 2D plot of a surface using colors

CALLING SEQUENCE :

graypolarplot (theta, rho, z, [strf, rect])

PARAMETERS :

rho : a vector with size n1, the discretization of the radius
theta : a vector with size n2, the discretization of the the angle.
z : real matrix of size (n1,n2). z(i, j) is the value of the surface at the point (rho(i),theta(j)).
strf : is a string of length 3 "xy0".
The default is "030".
x : controls the display of captions.
x=0 : no captions.
x=1 : captions are displayed. They are given by the optional argument leg.
y : controls the computation of the frame.
y=0 : the current boundaries (set by a previous call to another high level plotting function) are used. Useful
   when superposing multiple plots.
y=1 : the optional argument rect is used to specify the boundaries of the plot.
y=2 : the boundaries of the plot are computed using min and max values of x and y.
y=3 : like y=1 but produces isoview scaling.
y=4: like y=2 but produces isoview scaling.

y=5: like y=1 but plot2d can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

y=6: like y=2 but plot2d can change the boundaries of the plot and the ticks of the axes to produce pretty graduations. When the zoom button is activated, this mode is used.

y=7: like y=5 but the scale of the new plot is merged with the current scale.

y=8: like y=6 but the scale of the new plot is merged with the current scale.

leg: a string. It is used when the first character x of argument strf is 1. leg has the form "leg1@leg2@...." where leg1, leg2, etc. are respectively the captions of the first curve, of the second curve, etc.

The default is " ".

rect: This argument is used when the second character y of argument strf is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: rect=[xmin,ymin,xmax,ymax].

**DESCRIPTION:**

Takes a 2D plot of the surface given by z on a polar coordinate grid defined by rho and theta. Each grid region if filled with a gray or color level depending on the average value of z on the corners of the grid.

**EXAMPLES:**

```scilab
rho=1:0.1:4;theta=(0:0.02:1)*2*%pi;
z=30+round(theta'*(1+rhoˆ2));
xset('colormap',hotcolormap(128))
xset('background',xget('white'))
xbasc();graypolarplot(theta,rho,z)
```

1.2.48 **hist3d** ___________________________ 3D representation of a histogram

**CALLING SEQUENCE:**

```scilab
hist3d(f,[theta,alpha,leg,flag,ebox])
hist3d(list(f,x,y),[theta,alpha,leg,flag,ebox])
```

**PARAMETERS:**

- f : matrix of size (m,n) defining the histogram f(i,j)=F(x(i),y(j)), where x and y are taken as 0:m and 0:n.
- list(f,x,y) : where f is a matrix of size (m,n) defining the histogram f(i,j)=F(x(i),y(j)), with x and y vectors of size (1,n+1) and (1,m+1).
- theta, alpha, leg, flag, ebox : see plot3d.

**DESCRIPTION:**

hist3d represents a 2d histogram as a 3D plot. The values are associated to the intervals [x(i) x(i+1)[ X [y(i) y(i+1)].

Enter the command hist3d() to see a demo.

**SEE ALSO:** histplot 110, plot3d 123

**AUTHOR:** Steer S. & JPhilippe C.

1.2.49 **histplot** ___________________________ plot a histogram

**CALLING SEQUENCE:**

```scilab
Scilab Group April 1993 110
```
histplot(npoint, data, [style, strf, leg, rect, nax])

PARAMETERS :

npoint : integer or a row vector of increasing values.
data : real vector.
style, strf, leg, rect, nax : see plot2d.

DESCRIPTION :
- If npoint is an integer, histplot plots a histogram of the values stored in data using npoint equally spaced classes.
- If npoint is a vector histplot plots a histogram of the values stored in data using the classes npoint(k), npoint(k+1)].

Enter the command histplot() to see a demo.

SEE ALSO: hist3d 110, plot2d 118

1.2.50 hotcolormap ___________________________ red to yellow colormap

CALLING SEQUENCE :

cmap=hotcolormap(n)

PARAMETERS :

n : integer >= 3, the colormap size.
cmap : matrix with 3 columns [R, G, B].

DESCRIPTION :

hotcolormap computes a colormap with n hot colors varying from red to yellow.

EXAMPLE :

xset("colormap", hotcolormap(32))
plot3d1()

SEE ALSO: colormap 89, graycolormap 108, xset 154

1.2.51 isoview . set scales for isometric plot (do not change the size of the window)

CALLING SEQUENCE :

isoview(xmin, xmax, ymin, ymax)

PARAMETERS :

xmin, xmax, ymin, ymax : four real values

DESCRIPTION :

This function is obsolete, use preferably the frameflag=4 plot2d option which enable window resizing. isoview is used to have isometric scales on the x and y axes. It does not change the size of the graphics window. The rectangle xmin, xmax, ymin, ymax will be contained in the computed frame of the graphics window. isoview set the current graphics scales and can be used in conjunction with graphics routines which request the current graphics scale (for instance strf="x0z" in plot2d).

EXAMPLE :
t=[0:0.1:2*%pi]';
plot2d(sin(t),cos(t))
xbasc()
isoview(-1,1,-1,1)
plot2d(sin(t),cos(t),1,"001")
xset("default")

plot2d(sin(t),cos(t),frameflag=4)

SEE ALSO: square 133, xsetech 155

AUTHOR: Steer S.

1.2.52 legends ---------------------------------------- draw graph legend

CALLING SEQUENCE:

legends(strings,style [,opt])
legends(strings,style,xy)

PARAMETERS:

strings : n vector of strings, strings(i) is the legend of the ith curve
style : integer row vector of size n (the plot styles, third parameter of plot2d) or an integer 2 x n
matrix, style(1,k) contains the plot style for the kth curve and style(2,k) contains the line style (if
style(1,k)>0) or mark color (if style(1,k)<0).

opt = Upper right-hand corner 2 = Upper left-hand corner
3 = Lower left-hand corner
4 = Lower right-hand corner
5 = Interactive placement with the mouse (default)
xy : a vector [x,y] which gives the coordinates of the upper left corner of the legend box.

DESCRIPTION:

Puts a legend on the current plot using the specified strings as labels.
In the interactive placement (opt=5) move the legend box with the mouse and press the left button to release
it.
This function allow more flexible placement of the legends than the leg plot2d argument.

EXAMPLE:

t=0:0.1:2*%pi;
plot2d(t,[cos(t'),cos(2*t'),cos(3*t')],[-1,2 3]);
legends(['cos(t)';'cos(2*t)';'cos(3*t)'],[-1,2 3],4)
xset("line style",2);plot2d(t,cos(t),style=5);
xset("line style",4);plot2d(t,sin(t),style=3);
legends("[sin(t) ; cos(t) ]",[[5;2],[3;4]])

SEE ALSO: plot2d 118, xstring 157, xtitle 159

1.2.53 locate ------------------------------------------ mouse selection of a set of points

CALLING SEQUENCE:

Scilab Group        April 1993        112
milkdrop Scilab Function

\texttt{x=locate([n,flag])}

**PARAMETERS:**

\(x\) : matrix of size (2,n1). \(n1=n\) if the parameter \(n\) is given.
\(n,\text{flag}\) : integer values.

**DESCRIPTION:**

\texttt{locate} is used to get the coordinates of one or more points selected with the mouse in a graphics window. The coordinates are given using the current graphics scale.

If \(n>0\), \(n\) points are selected and their coordinates are returned in the matrix \(x\).
If \(n<=0\), points are selected until the user clicks with the left button of the mouse which stands for stop. The last point (clicked with the left button) is not returned.
\(x=\text{locate}()\) is the same as \(x=\text{locate}(-1)\).
If \(\text{flag}=1\) a cross is drawn at the points where the mouse is clicked.

**SEE ALSO:** \texttt{xclick 139, xgetmouse 145}

**AUTHOR:** S.S. & J.Ph.C

1.2.54 \texttt{m_circle} \texttt{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~} M-circle plot

**CALLING SEQUENCE:**

\begin{align*}
m_{\text{circle}}() \\
m_{\text{circle}}(\text{gain})
\end{align*}

**PARAMETERS:**

\(\text{gain}\) : vector of gains (in DB). The default value is \(\text{gain} =[-12 \cdot -6 \cdot -5 \cdot -4 \cdot -3 \cdot -2 \cdot -1.4 \cdot -1 \cdot -0.5 \cdot 0.25 \cdot 0.5 \cdot 0.7 \cdot 1 \cdot 1.4 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 8 \cdot 12]\)

**DESCRIPTION:**

\texttt{m_circle} is used with \texttt{nyquist}.

**EXAMPLE:**

//Example 1:
\begin{verbatim}
s=poly(0,'s');
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))
nyquist(h,0.01,100,'(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)')
m_circle();
\end{verbatim}
//Example 2:
\begin{verbatim}
xbas(:)
hi=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
nyquist([hi;h],0.01,100,['hi';'h'])
m_circle([-8 -6 -4]);
\end{verbatim}

**SEE ALSO:** \texttt{nyquist 115, chart 88, black 85}

**AUTHOR:** S.Steer.

Scilab Group April 1993 113
1.2.55  milk_drop ........................................ milk drop 3D function

CALLING SEQUENCE :

z=milk_drop(x,y)

PARAMETERS :

x,y : two row vectors of size n1 and n2.
z : matrix of size (n1,n2).

DESCRIPTION :
milk_drop is a function representing the surface of a milk drop falling down into milk. It can be used to test functions eval3d adn plot3d.

EXAMPLE :

x=-2:0.1:2; y=x;
z=eval3d(milk_drop,x,y);
plot3d(x,y,z)

SEE ALSO:  eval3d 97,  plot3d 123

AUTHOR : Steer S.

1.2.56  nf3d ........................................ rectangular facets to plot3d parameters

DESCRIPTION :

[xx,yy,zz]=nf3d(x,y,z)

PARAMETERS :

x,y,x,xx,yy,zz : 6 real matrices

DESCRIPTION :
Utility function. Used for transforming rectangular facets coded in three matrices x,y,z to scilab code for facets accepted by plot3d.

EXAMPLE :

// A sphere...
u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
x= cos(u)'*cos(v);
y= cos(u)'*sin(v);
z= sin(u)'*ones(v);
// plot3d2(x,y,z) is equivalent to...
[xx,yy,zz]=nf3d(x,y,z); plot3d(xx,yy,zz)

SEE ALSO:  plot3d 123,  plot3d2 126
param3d Scilab Function

1.2.57  nyquist  nyquist plot

CALLING SEQUENCE:

nyquist( sl, [fmin, fmax] [,step] [,comments] )
nyquist( sl, frq [,comments] )
nyquist(frq, db, phi [,comments])
nyquist(frq, repf [,comments])

PARAMETERS:

sl : syslin list (SIMO linear system in continuous or discrete time )
fmin, fmax : real scalars (frequency bounds (in Hz))
step : real (logarithmic discretization step)
comments : string vector (captions).
frq : vector or matrix of frequencies (in Hz) (one row for each output of sl).
db,phi : real matrices of modulus (in Db) and phases (in degree) (one row for each output of sl).
repf : matrix of complex numbers. Frequency response (one row for each output of sl)

DESCRIPTION:

Nyquist plot i.e Imaginary part versus Real part of the frequency response of sl.
For continuous time systems sl(2*%i*%pi*w) is plotted. For discrete time system or discretized systems
sl(exp(2*%i*%pi*w*fd) is used ( fd=1 for discrete time systems and fd=sl('dt') for
discretized systems )
sl can be a continuous-time or discrete-time SIMO system (see syslin). In case of multi-output the
outputs are plotted with different symbols.
The frequencies are given by the bounds fmin, fmax (in Hz) or by a row-vector (or a matrix for multi-
output) frq.
step is the (logarithmic ) discretization step. (see calfrq for the choice of default value).
comments is a vector of character strings (captions).
db,phi are the matrices of modulus (in Db) and phases (in degrees). (One row for each response).
repf is a matrix of complex numbers. One row for each response.
Default values for fmin and fmax are 1.d-3,1.d+3 if sl is continuous-time or 1.d-3,0.5 if sl
is discrete-time.
Automatic discretization of frequencies is made by calfrq.

EXAMPLE:

xbasc();
s=poly(0,'s');
h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01));
comm='(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01)';
nyquist(h,0.01,100,comm);
h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))
xbasc();
nyquist([h1;h],0.01,100,['h1';'h'])
xbasc();nyquist([h1;h])

SEE ALSO:  bode 86, black 85, calfrq 324, freq 339, repfreq 355, phasemag 353

1.2.58  param3d  3D plot of a curve

CALLING SEQUENCE:

Scilab Group  April 1993  115
param3d(x, y, z, [theta, alpha, leg, flag, ebox])

PARAMETERS:

x, y, z : three vectors of the same size (points of the parametric curve).
theta, alpha : real values giving in degree the spherical coordinates of the observation point.
leg : string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".
flag = [type, box] : type and box have the same meaning as in plot3d:
type : an integer (scaling).
type = 0 the plot is made using the current 3D scaling (set by a previous call to param3d, plot3d, contour or plot3d1).
type = 1 rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument ebox.
type = 2 rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.
type = 3 3d isometric with box bounds given by optional ebox, similarly to type = 1

type = 4 3d isometric bounds derived from the data, to similarly type = 2

type = 5 3d expanded isometric bounds with box bounds given by optional ebox, similarly to type = 1

type = 6 3d expanded isometric bounds derived from the data, similarly to type = 2

box : an integer (frame around the plot).
box = 0 nothing is drawn around the plot.
box = 1 unimplemented (like box = 0).
box = 2 only the axes behind the surface are drawn.
box = 3 a box surrounding the surface is drawn and captions are added.
box = 4 a box surrounding the surface is drawn, captions and axes are added.

ebox : used when type in flag is 1. It specifies the boundaries of the plot as the vector [xmin, xmax, ymin, ymax, zmin, zmax].

DESCRIPTION:
	param3d is used to plot a 3D curve defined by its coordinates x, y and z.
    Use param3d1 to do multiple plots.
    Enter the command param3d() to see a demo.

EXAMPLE:

t = 0:0.1:5*%pi;
param3d(sin(t), cos(t), t/10, 35, 45, "X@Y@Z", [2, 3])

SEE ALSO: param3d1 116, plot3d 123

1.2.59 param3d1 3D plot of curves

CALLING SEQUENCE:

param3d1(x, y, z, [theta, alpha, leg, flag, ebox])
param3d1(x, y, list(z, colors), [theta, alpha, leg, flag, ebox])

PARAMETERS:

x, y, z : matrices of the same size (nl, nc). Each column i of the matrices corresponds to the coordinates of the ith curve.

You can give a specific color for each curve by using list(z, colors) instead of z, where colors is a vector of size nc. If color(i) is negative the curve is plotted using the mark with id abs(style(i))+1; if style(i) is strictly positive, a plain line with color id style(i) or a dashed line with dash id style(i) is used. Use xset() to see the mark and color ids.

theta, alpha : real values giving in degree the spherical coordinates of the observation point.
paramfplot2d Scilab Function

leg : string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".
flag=[type,box] : type and box have the same meaning as in plot3d.
type : an integer (scaling).
type=0 the plot is made using the current 3D scaling (set by a previous call to param3d, plot3d, contour or plot3d1).
type=1 rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument ebox.
type=2 rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.
type=3 3d isometric with box bounds given by optional ebox, similarly to type=1

type=4 3d isometric bounds derived from the data, to similarly type=2
type=5 3d expanded isometric bounds with box bounds given by optional ebox, similarly to type=1
type=6 3d expanded isometric bounds derived from the data, similarly to type=2
box : an integer (frame around the plot).
box=0 nothing is drawn around the plot.
box=1 unimplemented (like box=0).
box=2 only the axes behind the surface are drawn.
box=3 a box surrounding the surface is drawn and captions are added.
box=4 a box surrounding the surface is drawn, captions and axes are added.
ebox : used when type in flag is 1. It specifies the boundaries of the plot as the vector [xmin,xmax,ymin,ymax,zmin,zmax].

DESCRIPTION :
param3d1 is used to plot 3D curves defined by their coordinates x, y and z.

Enter the command param3d1() to see a demo.

EXAMPLE :

```
t=[0:0.1:5*%pi]';
param3d1([sin(t),sin(2*t)],[cos(t),cos(2*t)],..
   list([t/10,sin(t)],[3,2]),35,45,"X@Y@Z",[2,3])
```

SEE ALSO : param3d 115, plot3d 123, xset 154

AUTHOR : J.Ph.C.

1.2.60 paramfplot2d __________ animated 2D plot, curve defined by a function

CALLING SEQUENCE :

```
paramfplot2d(f,x,theta)
paramfplot2d(f,x,theta,flag)
paramfplot2d(f,x,theta,flag,rect)
```

PARAMETERS :

x : real vector.
f : function y=f(x,t) . f is a Scilab function or a dynamically linked routine (referred to as a string).
theta : real vector (set of parameters).
flag : string 'no' or 'yes': If "yes" screen is cleared between two consecutive plots.
rect : "rectangle" [xmin, xmax, ymin, ymax] (1 x 4 real vector),

DESCRIPTION :

Animated plot of the function x–>f(x,t) for t=theta(1),theta(2),etc. f can be a either Scilab function or a dynamically linked routine since y=f(x,t) is evaluated as y=feval(x(:),t,f). See feval. f: mapping x,t -> f(x,t) = R^N valued function for x= vector of R^N and t=real number. x is a N-vector of x-values and for each t in theta, f(x,t)=N-vector of y-values.

EXAMPLE :

```
Scilab Group April 1993 117
```
```scilab
deff('y=f(x,t)','y=t*sin(x)')
x=linspace(0,2*%pi,50);theta=0:0.05:1;
paramfplot2d(f,x,theta);

SEE ALSO:  plot2d 118,  feval 41,  fplot2d 102,  pixmap ?

1.2.61  plot  ____________________________________________________________________ simple plot

CALLING SEQUENCE :

plot (x,y,[xcap,ycap,caption])
plot (y)

PARAMETERS :

x, y : two vectors with same sizes
xcap, ycap, caption : character strings or string matrices

DESCRIPTION :
Plot y as function of x. xcap and ycap are captions for x-axis and y-axis respectively and caption is the caption of the plot.

Invoked with only one argument, plot (y) plots the y vector or, if y is a matrix, it plots all its row vectors on the same plot. This plot is done with respect to the vector 1:<number of columns of y>.

plot is obsolete. Use plot2d instead.

EXAMPLE :

x=0:0.1:2*%pi;
// simple plot
plot(sin(x))
// using captions
xbasc()
plot(x,sin(x),"sin","time","plot of sinus")
// plot 2 functions
xbasc()
plot([sin(x);cos(x)])

SEE ALSO:  plot2d 118

AUTHOR : J.Ph.C.

1.2.62  plot2d  ____________________________________________________________________ 2D plot

CALLING SEQUENCE :

plot2d([x],y)
plot2d([x],y,<opt_args>)
plot2d([logflag],x,y,[style,strf,leg,rect,nax])

PARAMETERS :

x, y : two matrices or vectors.
- If y is a vector, x must be a vector with the same size. If x is not given, it is supposed to be the vector 1:<size of y>.
- If y is a matrix, x can be:
  + a vector with size equal to the row dimension of y (each column of y is plotted with respect to x)
```

Scilab Group

April 1993 118
+ a matrix with the same dimensions as \( y \) (each column of \( y \) is plotted with respect to the corresponding column of \( x \))
+ If \( x \) is not given, it is supposed to be the vector \( 1: \text{<row dimension of } y \rangle \)

\(<\text{opt_args}>\) : This represents a sequence of statements \( \text{key1=\text{value1}}, \text{key2=\text{value2}}, \ldots \) where \( \text{key1, key2,} \ldots \) can be one of the following:

- **style** : sets the style for each curve, see below for value.
- **leg** : sets the curves captions. If this key is given and \( \text{strf} \) is not given then \( x \) character of \( \text{strf} \) is supposed to be 1. See below for value.
- **rect** : sets the bounds of the plot. If this key is given and neither \( \text{frameflag} \) nor \( \text{strf} \) is given then the \( y \) character of \( \text{strf} \) is supposed to be 7. See below for value.
- **nax** : sets the grids definition. If this key is given and neither \( \text{axesflag} \) nor \( \text{strf} \) is given then the \( z \) character of \( \text{strf} \) is supposed to be 1. See below for value.
- **logflag** : sets the graduation type (linear or logarithmic) along the axes. See below for value.
- **frameflag** : specifies how the frame of the plot is computed. The value is an integer ranging from 0 to 8. It corresponds to the \( y \) character of \( \text{strf} \). See below.
- **axesflags** : specifies what kind of axes are drawn around the plot. The value is an integer ranging from 0 to 5. It corresponds to the \( z \) character of \( \text{strf} \). See below.
- **style** : is a real row vector of size \( nc \). The style to use for curve \( i \) is defined by \( \text{style}(i) \).

The default style is \( 1: nc \) (1 for the first curve, 2 for the second, etc.).
- if \( \text{style}(i) \) is negative or zero, the curve is plotted using the mark with id \( \text{abs(\text{style}(i))} \); use \( \text{xset()} \) to set the mark id and \( \text{xget('mark')} \) to get the current mark id.
- if \( \text{style}(i) \) is strictly positive, a plain line with color id \( \text{style}(i) \) or a dashed line with dash id \( \text{style}(i) \) is used; use \( \text{xset()} \) to see the color ids.
- When only one curve is drawn, \( \text{style} \) can be the row vector of size 2 \( [\text{sty},\text{pos}] \) where \( \text{sty} \) is used to specify the style and \( \text{pos} \) is an integer ranging from 1 to 6 which specifies a position to use for the caption. This can be useful when a user wants to draw multiple curves on a plot by calling the function \( \text{plot2d} \) several times and wants to give a caption for each curve.

\( \text{strf} \) : is a string of length 3 "xyz". The default is "081".

- **x** : controls the display of captions.
  - \( x=0 \) : no caption.
  - \( x=1 \) : captions are displayed. They are given by the optional argument \( \text{leg} \).

- **y** : controls the computation of the actual coordinate ranges from the minimal requested values. Actual ranges can be larger than minimal requirements.

<table>
<thead>
<tr>
<th>requirements</th>
<th>ranges of a previous plot</th>
<th>ranges given by rect arg</th>
<th>ranges computed from ( x ) and ( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>requested one</td>
<td>( y=0 )</td>
<td>( y=1 )</td>
<td>( y=2 )</td>
</tr>
<tr>
<td>Computed for isometric view</td>
<td>( y=3 )</td>
<td>( y=4 )</td>
<td></td>
</tr>
<tr>
<td>Enlarged For pretty axes</td>
<td>( y=5 )</td>
<td>( y=6 )</td>
<td></td>
</tr>
<tr>
<td>Previous and current plots merged</td>
<td>( y=7 )</td>
<td>( y=8 )</td>
<td></td>
</tr>
</tbody>
</table>

- **z** : controls the display of information on the frame around the plot. If axes are requested, the number of tics can be specified by the \( \text{nax} \) optional argument.
  - \( z=0 \) : nothing is drawn around the plot.
  - \( z=1 \) : axes are drawn, the \( y \)-axis is displayed on the left.
  - \( z=2 \) : the plot is surrounded by a box without tics.
  - \( z=3 \) : axes are drawn, the \( y \)-axis is displayed on the right.
z=4 : axes are drawn centred in the middle of the frame box.
z=5 : axes are drawn so as to cross at point (0, 0). If point (0, 0) does not lie inside the frame, axes will not appear on the graph.

leg : a string. It is used when the first character x of argument strf is 1. leg has the form "leg1@leg2@...." where leg1, leg2, etc. are respectively the captions of the first curve, of the second curve, etc.

The default is " ".

rect : This argument is used when the second character y of argument strf is 1, 3 or 5. It is a row vector of size 4 and gives the dimension of the frame: rect=[xmin,ymin,xmax,ymax].

nax : This argument is used when the third character z of argument strf is 1. It is a row vector with four entries [nx,Nx,ny,Ny] where nx (ny) is the number of subgraduations on the x (y) axis and Nx (Ny) is the number of graduations on the x (y) axis.

logflag : a string formed by to characters h (for horizontal axis) and v (for vertical axis) each of these characters can take the values "n" or "l". "l" stands for logarithmic graduation and "n" for normal graduation. For example "ll" stands for a log-log plot. Default value is "nn".

DESCRIPTION :
plot2d plots a set of 2D curves. Piecewise linear plotting is used.

By default, successive plots are superposed. To clear the previous plot, use xbasc().
See the meaning of the parameters above for a complete description.
Enter the command plot2d() to see a demo.
Other high level plot2d function exist:

plot2d2 : same as plot2d but the curve is supposed to be piecewise constant.
plot2d3 : same as plot2d but the curve is plotted with vertical bars.
plot2d4 : same as plot2d but the curve is plotted with arrows.

EXAMPLE :

// simple plot
x=[0:0.1:2*%pi]’;
plot2d(sin(x))
xbasc()
plot2d(x,sin(x))
// multiple plot
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
// multiple plot giving the dimensions of the frame
// old syntax and new syntax
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],1:3,"011"," ",[0,0,6,0.5])
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5])
// multiple plot with captions and given tics
// old syntax and new syntax
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],.. [1,2,3],"l11","L1@L2@L3",[0,-2,2*%pi,2],[2,10,2,10])
xbasc()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],.. [1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*%pi,2])
// isoview
xbasc()
plot2d(x,sin(x),1,"041")
// scale
xbasc()
plot2d(x,sin(x),1,"061")
// auto scaling with previous plots
xbasc();
plot2d(x,sin(x),1)
plot2d(x,2*sin(x),2)
plot2d(2*x,cos(x),3)
// axis on the right
xbasc();
plot2d(x,sin(x),1,"183","sin(x)")
// centered axis
xbasc();
plot2d(x,sin(x),1,"184","sin(x)")
// axis centered at (0,0)
xbasc();
plot2d(x-4,sin(x),1,"185","sin(x)")

SEE ALSO : plot2d1 121, plot2d2 122, plot2d3 122, plot2d4 123, xbasc 137, xset 154

AUTHOR : J.Ph.C.

1.2.63 plot2d1 ____________________________ 2D plot (logarithmic axes) (obsolete)

CALLING SEQUENCE :

plot2d1(str,x,y,[style,strf,leg,rect,nax])

PARAMETERS :

str : is a string of length three "abc".  
   a : can have the following values: e, o or g.  
   e : means "empty". It specifies the fact that the value of x is not used (the x values are supposed to be 
      regularly spaced, ie 1:<number of rows of y>). The user must anyway give a value for x, 1 for instance: 
      plot2d1("enn",1,y).
   o : means "one". If there are many curves, they all have the same x-values: x is a column vector of size nl 
      and y is a matrix of size (nl,nc). For example: x=[0:0.1:2*%pi]'; plot2d1("onn",x,[sin(x) 
      cos(x)]).
   g : means "general". x and y must have the same size (nl,nc). Each column of y is plotted with respect 
      to the corresponding column of x. nc curves are plotted using nl points.

b, c : can have the values n (normal) or l (logarithmic).

b=1 : a logarithmic axis is used on the x-axis

c=1 : a logarithmic axis is used on the y-axis

x,y,[style,strf,leg,rect,nax] : these arguments have the same meaning as in the plot2d 
   function.

opt_args : these arguments have the same meaning as in the plot2d function.

DESCRIPTION :  
This function is obsolete. Use plot2d instead. 
plot2d1 plots a set of 2D curves. It is the same as plot2d but with one more argument str which 
enables logarithmic axis. Moreover, it allows to specify only one column vector for x when it is the same 
for all the curves.  
By default, successive plots are superposed. To clear the previous plot, use xbasc.  
Enter the command plot2d1() to see a demo.
// multiple plot without giving x
x=[0:0.1:2*%pi]’;
plot2d1("enn",1,[sin(x) sin(2*x) sin(3*x)])
// multiple plot using only one x
xbasc()
plot2d1("onn",x,[sin(x) sin(2*x) sin(3*x)])
// logarithmic plot
x=[0.1:0.1:3]’; xbasc()
plot2d1("oll",x,[exp(x) exp(x^2) exp(x^3)])

SEE ALSO: plot2d 118, plot2d2 122, plot2d3 122, plot2d4 123, xbasc 137

1.2.64 plot2d2 _____________________________ 2D plot (step function)

CALLING SEQUENCE:

plot2d2([x],y)
plot2d2([x],y,<opt_args>)
plot2d2([logflag],x,y,[style,strf,leg,rect,nax])

PARAMETERS:

[]: see plot2d for a description of parameters.

DESCRIPTION:

plot2d2 is the same as plot2d but the functions given by (x,y) are supposed to be piecewise constant.

By default, successive plots are superposed. To clear the previous plot, use xbasc().

Enter the command plot2d2() to see a demo.

EXAMPLE:

// plots a step function of value i on the segment [i,i+1]
// the last segment is not drawn
plot2d2([1:4],[1:4],1,"111","step function",[0,0,5,5])
// compare the following with plot2d
x=[0:0.1:2*%pi]’;
xbasc()
plot2d2(x,[sin(x) sin(2*x) sin(3*x)])

SEE ALSO: plot2d 118, plot2d3 122, plot2d4 123, subplot 133, xbasc 137, xset 154

1.2.65 plot2d3 _____________________________ 2D plot (vertical bars)

CALLING SEQUENCE:

plot2d3([logflags,] x,y,[style,strf,leg,rect,nax])
plot32d(y)
plot2d3(x,y <,opt_args>)}
plot3d Scilab Function

PARAMETERS:
[]: see plot2d for a description of parameters.

DESCRIPTION:
plot3d is the same as plot2d but curves are plotted using vertical bars.
By default, successive plots are superposed. To clear the previous plot, use xbase().
Enter the command plot3d() to see a demo.

EXAMPLE:
// compare the following with plot2d1
x=[0:0.1:2*%pi]’;
plot3d(x,[sin(x) sin(2*x) sin(3*x)])

SEE ALSO: plot2d 118, plot2d2 122, plot2d3 122, xbase 137, xset 154

AUTHOR: J.Ph.C.

1.2.66 plot2d4 --------------------------------------------- 2D plot (arrows style)

CALLING SEQUENCE:
plot2d4([logflag,] x,y,[style,strf,leg,rect,nax])
plot2d4(y)
plot2d4(x,y <,opt_args>)

PARAMETERS:
[]: see plot2d for a description of parameters.

DESCRIPTION:
plot2d4 is the same as plot2d but curves are plotted using arrows style. This can be useful when
plotting solutions of an ODE in a phase space.
By default, successive plots are superposed. To clear the previous plot, use xbase().
Enter the command plot2d4() to see a demo.

EXAMPLE:
// compare the following with plot2d1
x=[0:0.1:2*%pi]’;
plot2d4(x,[sin(x) sin(2*x) sin(3*x)])

SEE ALSO: fchamp 99, plot2d 118, plot2d2 122, plot2d3 122, subplot 133,
xbase 137, xset 154

AUTHOR: J.Ph.C.

1.2.67 plot3d --------------------------------------------- 3D plot of a surface

CALLING SEQUENCE:
plot3d(x,y,z,[theta,alpha,leg,flag,ebox])
plot3d(x,y,z,<opt_args>)

plot3d(xf,yf,zf,[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,zf,<opt_args>)

plot3d(xf,yf,list(zf,colors),[theta,alpha,leg,flag,ebox])
plot3d(xf,yf,list(zf,colors),<opt_args>)
**PARAMETERS:**

- **x, y**: row vectors of sizes n1 and n2 (x-axis and y-axis coordinates). These coordinates must be monotone.
- **z**: matrix of size (n1,n2). \( z(i,j) \) is the value of the surface at the point \( (x(i),y(j)) \).
- **xf, yf, zf**: matrices of size (nf,n). They define the facets used to draw the surface. There are \( n \) facets.

  Each facet \( i \) is defined by a polygon with \( nf \) points. The x-axis, y-axis and z-axis coordinates of the points of the \( i \)th facet are given respectively by \( xf(:,i), yf(:,i) \) and \( zf(:,i) \).

- **colors**: a vector of size \( n \) giving the color of each facets or a matrix of size \( (nf,n) \) giving color near each facet boundary (facet color is interpolated)

  <opt_args> : This represents a sequence of statements \( key1=value1, key2=value2... \) where \( key1, key2, ... \) can be one of the following: \( \theta, \alpha, \text{leg}, \text{flag}, \text{ebox} \) (see definition below)

**theta, alpha**: real values giving in degree the spherical coordinates of the observation point.

**leg**: string defining the captions for each axis with @ as a field separator, for example "X@Y@Z".

**flag**: a real vector of size three \( \text{flag}=[\text{mode}, \text{type}, \text{box}] \).

- **mode**: string (treatment of hidden parts).
  - \( \text{mode}=0 \) the hidden parts of the surface are removed and the surface is painted with color \( \text{mode} \).
  - \( \text{mode}=0 \) only the hidden parts of the surface are drawn.

**type**: an integer (scaling).

- **type=0**: the plot is made using the current 3D scaling (set by a previous call to \( \text{param3d}, \text{plot3d}, \text{contour} \) or \( \text{plot3d1} \)).
- **type=1**: rescales automatically 3d boxes with extreme aspect ratios, the boundaries are specified by the value of the optional argument \( \text{ebox} \).
- **type=2**: rescales automatically 3d boxes with extreme aspect ratios, the boundaries are computed using the given data.
- **type=3**: 3d isometric with box bounds given by optional \( \text{ebox} \), similarly to \( \text{type}=1 \)
- **type=4**: 3d isometric bounds derived from the data, to similarly \( \text{type}=2 \)
- **type=5**: 3d expanded isometric bounds with box bounds given by optional \( \text{ebox} \), similarly to \( \text{type}=1 \)
- **type=6**: 3d expanded isometric bounds derived from the data, similarly to \( \text{type}=2 \)

**box**: an integer (frame around the plot).

- **box=0**: nothing is drawn around the plot.
- **box=1**: unimplemented (like box=0).
- **box=2**: only the axes behind the surface are drawn.
- **box=3**: a box surrounding the surface is drawn and captions are added.
- **box=4**: a box surrounding the surface is drawn, captions and axes are added.

**ebox**: used when \( \text{type} \) in \( \text{flag} \) is 1. It specifies the boundaries of the plot as the vector \( [xmin,xmax,ymin,ymax,zmin,zmax] \).

**DESCRIPTION:**

\( \text{plot3d}(x,y,z,[\text{theta}, \text{alpha}, \text{leg}, \text{flag}, \text{ebox}]) \) draws the surface \( z=f(x,y) \).

\( \text{plot3d}(xf,yf,zf,[\text{theta}, \text{alpha}, \text{leg}, \text{flag}, \text{ebox}]) \) draws a surface defined by a set of facets. You can draw multiple plots by replacing \( xf, yf \) and \( zf \) by multiple matrices assembled by rows as \( [xf1 \ xf2 \ ...], [yf1 \ yf2 \ ...] \) and \( [zf1 \ zf2 \ ...] \).

You can give a specific color for each facet by using \( \text{list}(zf, \text{colors}) \) instead of \( zf \), where \( \text{colors} \) is a vector of size \( n \). If \( \text{colors}(i) \) is positive it gives the color of facet \( i \) and the boundary of the facet is drawn with current line style and color. If \( \text{colors}(i) \) is negative, color id \( -\text{colors}(i) \) is used and the boundary of the facet is not drawn. Use \( \text{xset()} \) to see the ids of the colors.

It is also possible to get interpolated color for facets. For that the color argument must be a matrix of size \( nfxn \) giving the color near each boundary of each facets. In this case positive values for colors mean that the boundary are not drawn.

The optional arguments \( \text{theta}, \text{alpha}, \text{leg}, \text{flag}, \text{ebox} \) can be passed by a sequence of statements \( \text{key1=value1, key2=value2...} \). In this case, the order has no special meaning.

You can use the function \( \text{genfac3d} \) to compute four sided facets from the surface \( z=f(x,y) \). \( \text{eval3dp} \) can also be used.

Enter the command \( \text{plot3d}() \) to see a demo.

**EXAMPLE:**

Scilab Group  
April 1993  
124
plot3d1 Scilab Function

// simple plot using z=f(x,y)
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
plot3d(t,t,z)
// same plot using facets computed by genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
xbasch()
plot3d(xx,yy,zz)
// multiple plots
xbasch()
plot3d([xx xx],[yy yy],[zz 4+zz])
// multiple plots using colors
xbasch()
plot3d([xx xx],[yy yy],list([zz zz+4],[4*ones(1,400) 5*ones(1,400)]))
// simple plot with viewpoint and captions
xbasch()
plot3d(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[2,2,3])
// plot of a sphere using facets computed by eval3dp
deff('[x,y,z]=sph(alp,tet)',["x=r*cos(alp).*cos(tet)+orig(1)*ones(tet)";..
    "y=r*cos(alp).*sin(tet)+orig(2)*ones(tet)";..
    "z=r*sin(alp)+orig(3)*ones(tet)"]);
    r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
xbasch();plot3d(xx,yy,zz)
xbasch();xset('colormap',hotcolormap(128));
r=0.3;orig=[1.5 0 0];
[xxl,yy1,zz1]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
cc=(xx+zz+2)*32;cc1=(xx1-orig(1)+zz1/r+2)*32;
xbasch();plot3d1([xx xx1],[yy yy1],list([zz,zz1],[cc cc1]),70,80)
xbasch();plot3d1([xx xx1],[yy yy1],list([zz,zz1],[cc cc1]),theta=70,alpha=80,flag=[5,6,3])

See Also: eval3dp 98, genfac3d 105, geom3d 105, param3d 115, plot3d1 125,
xset 154

AUTHOR: J.Ph.C.

1.2.68 plot3d1 ------------------------ 3D gray or color level plot of a surface

DESCRIPTION:

plot3d1(x,y,z, [theta, alpha, leg, flag, ebox])
plot3d1(xf,yf,zf, [theta, alpha, leg, flag, ebox])

PARAMETERS:
See plot3d for a full description. There is just a slight difference, only the sign of the flag(1)=mode
parameter is used: if it is negative the grid is not drawn.

DESCRIPTION:
plot3d plots a surface z=f(x,y) with colors depending on the z-level of the surface.

Enter the command plot3d1() to see a demo.

EXAMPLE:

// simple plot using z=f(x,y)
t=[0:0.3:2*%pi]'; z=sin(t)*cos(t');
plot3d3(t,t,z)
// same plot using facets computed by genfac3d
[xx,yy,zz]=genfac3d(t,t,z);
xbasca()
plot3d3(xx,yy,zz)
// multiple plots
xbasca()
plot3d3([xx xx],[yy yy],[zz 4+zz])
// simple plot with viewpoint and captions
xbasca()
plot3d3(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[2,2,3])
// same plot without grid
xbasca()
plot3d3(1:10,1:20,10*rand(10,20),35,45,"X@Y@Z",[-2,2,3])
// plot of a sphere using facets computed by eval3dp
deff("[x,y,z]=sph(alp,tet)","x=r*cos(alp).*cos(tet)+orig(1)*ones(tet);
    y=r*cos(alp).*sin(tet)+orig(2)*ones(tet);
    z=r*sin(alp)+orig(3)*ones(tet)"");
r=1; orig=[0 0 0];
[xx,yy,zz]=eval3dp(sph,linspace(-%pi/2,%pi/2,40),linspace(0,%pi*2,20));
xbasca()
plot3d3(xx,yy,zz)

SEE ALSO: plot3d123

AUTHOR: J.Ph.C.

1.2.69  plot3d2  plots surface defined by rectangular facets

DESCRIPTION:

plot3d2(X,Y,Z [,vect,theta,alpha,leg,flag,ebox])
plot3d2(X,Y,Z, <opt_args>)

PARAMETERS:

X,Y,Z : 3 real matrices
vect : real vector
<opt_args> : This represents a sequence of statements key1=value1, key2=value2... where
                 key1,key2,... can be one of the following: vect (see above), theta, alpha ,leg,flag,ebox (see plot3d)

DESCRIPTION:

plot3d2 plots a surface defined by rectangular facets. (X,Y,Z) are three matrices which describe a
surface. The surface is composed of four sided polygons. The X-coordinates of a facet are given by
X(i,j),X(i+1,j),X(i,j+1),X(i+1,j+1). And similarly Y and Z are Y and Z coordinates. The vect vector is
used when multiple surfaces are coded in the same (X,Y,Z) matrices. vect (j) gives the line at which the
coding of the jth surface begins. See plot3d for a full description.

EXAMPLE:

u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)'*cos(v);
Y = cos(u)'*sin(v);
Z = sin(u)'*ones(v);
plot3d2(X,Y,Z);

SEE ALSO: plot3d123, genfac3d105

Scilab Group  April 1993  126
plot3d3(X,Y,Z [,vect,theta,alpha,leg,flag,ebox])
plot3d3(X,Y,Z, <opt_args>)

DESCRIPTION :
plot3d3 performs a mesh plot of a surface defined by rectangular facets. (X,Y,Z) are three matrices which describe a surface. The surface is composed of four sided polygons. The X-coordinates of a facet are given by X(i,j),X(i+1,j),X(i,j+1),X(i+1,j+1). And similarly Y and Z are Y and Z coordinates. The vect vector is used when multiple surfaces are coded in the same (X,Y,Z) matrices. vect(j) gives the line at which the coding of the jth surface begins. See plot3d2 for a full description.

EXAMPLE :

u = linspace(-%pi/2,%pi/2,40);
v = linspace(0,2*%pi,20);
X = cos(u)’*cos(v);
Y = cos(u)’*sin(v);
Z = sin(u)’*ones(v);
plot3d3(X,Y,Z);

SEE ALSO:  plot3d2 126,  plot3d 123,  param3d 115

plotframe _________ plot a frame with scaling and grids

CALLING SEQUENCE :

plotframe(rect,tics,[arg_opt1,arg_opt2,arg_opt3])

PARAMETERS :
rect : vector [xmin,ymin,xmax,ymax].
tics : vector [nx,mx,ny,my] where mx, nx (resp. my, ny) are the number of x-axis (resp. y-axis) intervals and subintervals.
arg_optX : optional arguments up to three and choosen among.
flags : vector [wantgrids,findbounds] where wantgrids is a boolean variable (%t or %f) which indicates gridding. findbounds is a boolean variable. If findbounds is %t, the bounds given in rect are allowed to be slightly modified (in fact always increased) in order to have simpler graduations: then tics(2) and tics(4) are ignored.
captions : vector of 3 strings [title,x-leg,y-leg] corresponding respectively to the title of the plot and the captions on the x-axis and the y-axis.
subwin : a vector of size 4 defining the sub window. The sub window is specified with the parameter subwin=[x,y,w,h] (upper-left, width, height). The values in subwin are specified using proportion of the width or height of the current graphics window (see xsetech).
DESCRIPTION:
plotframe is used with 2D plotting functions plot2d, plot2d1,... to set a graphics frame. It must be used before plot2d which should be invoked with the "000" superposition mode.

EXAMPLE:

```scilab
x=[-0.3:0.8:27.3]';
y=rand(x);
rect=[min(x),min(y),max(x),max(y)];
tics=[4,10,2,5]; //4 x-intervals and 2 y-intervals
plotframe(rect,tics,["My plot","x","y"],[0,0,0.5,0.5])
plot2d(x,y,2,"000")
plotframe(rect,tics,["My plot with grids","x","y"],[0.5,0,0.5,0.5])
plot2d(x,y,3,"000")
plotframe(rect,tics,[...,"
["My plot with grids and automatic bounds","x","y"],[0,0.5,0.5,0.5])
plot2d(x,y,4,"000")
plotframe(rect,tics,["My plot with grids and automatic bounds","x","y"],[0,0.5,0.5,0.5])
plot2d(x,y,5,"000")
xset("default")
```

SEE ALSO: plot2d 118, graduate 108, xsetech 155

### 1.2.72 plzr ________________________________ pole-zero plot

CALLING SEQUENCE:

```scilab
plzr(sl)
```

PARAMETERS:

*sl*: list(syslin)

DESCRIPTION:
produces a pole-zero plot of the linear system sl (syslin list)

EXAMPLE:

```scilab
s=poly(0,'s');
n=[1+s 2+3*s+4*s^2 5; 0 1-s s];
d=[1+3*s 5-s^3 s+1;1+s 1+s+s^2 3*s-1];
h=syslin('c',n./d);
plzr(h);
```

SEE ALSO: trzeros 368, roots 496, syslin 224

### 1.2.73 polarplot __________________________ Plot polar coordinates

CALLING SEQUENCE:

```scilab
polarplot(theta,rho,[style,strf,leg,rect])
polarplot(theta,rho,<opt_args>)
```

PARAMETERS:

Scilab Function Scilab Group 128
printing Scilab Function

rho : a vector, the radius values
theta : a vector with same size than rho, the angle values.

<opt_args> : a sequence of statements key1=value1, key2=value2, ... where keys may be

style, leg, rect, strf or frameflag

style : is a real row vector of size nc. The style to use for curve i is defined by style(i).

The default style is 1:nc (1 for the first curve, 2 for the second, etc.).
- if style(i) is negative, the curve is plotted using the mark with id abs(style(i))+1; use
  xset() to see the mark ids.
- if style(i) is strictly positive, a plain line with color id style(i) or a dashed line with dash id
  style(i) is used; use xset() to see the color ids.
- When only one curve is drawn, style can be the row vector of size 2 [sty,pos] where sty is
  used to specify the style and pos is an integer ranging from 1 to 6 which specifies a position to use
  for the caption. This can be useful when a user wants to draw multiple curves on a plot by calling the
  function plot2d several times and wants to give a caption for each curve.

strf : is a string of length 3 "xy0".
The default is "030".

x : controls the display of captions,
  x=0 : no captions.
  x=1 : captions are displayed. They are given by the optional argument leg.

y : controls the computation of the frame. same as frameflag
  y=0 : the current boundaries (set by a previous call to another high level plotting function) are used. Useful
  when superposing multiple plots.
  y=1 : the optional argument rect is used to specify the boundaries of the plot.
  y=2 : the boundaries of the plot are computed using min and max values of x and y.
  y=3 : like y=1 but produces isoview scaling.
  y=4 : like y=2 but produces isoview scaling.
  y=5 : like y=1 but plot2d can change the boundaries of the plot and the ticks of the axes to produce
  pretty graduations. When the zoom button is activated, this mode is used.
  y=6 : like y=2 but plot2d can change the boundaries of the plot and the ticks of the axes to produce
  pretty graduations. When the zoom button is activated, this mode is used.
  y=7 : like y=5 but the scale of the new plot is merged with the current scale.
  y=8 : like y=6 but the scale of the new plot is merged with the current scale.

leg : a string. It is used when the first character x of argument strf is 1. leg has the form "leg1@leg2@...."
       where leg1, leg2, etc. are respectively the captions of the first curve, of the second curve, etc.

The default is " ".

rect : This argument is used when the second character y of argument strf is 1, 3 or 5. It is a row
       vector of size 4 and gives the dimension of the frame: rect=[xmin,ymin,xmax,ymax].

DESCRIPTION :

polarplot creates a polar coordinate plot of the angle theta versus the radius rho. theta is the angle from the
x-axis to the radius vector specified in radians; rho is the length of the radius vector specified in dataspace
units.

EXAMPLES :

t= 0:.01:2*%pi;
xbas();polarplot(sin(7*t),cos(8*t))

xbas();polarplot([sin(7*t') sin(6*t')],[cos(8*t') cos(8*t')],[1,2])

1.2.74 printing ___________________________________________ printing scilab graphics

CALLING SEQUENCE :

Scilab Group April 1993 129
Blatexpr xscale yscale filename.ps
BEpsf filename.ps
Blpr "Title" filename1.ps filename2.ps ... filenamen.ps | lpr

DESCRIPTION:
The scilab graphics can be saved with the xbasimp command into unix files. The Scilab command:
xbasimp(xx,'des.ps',0)
will save the graphics recorded in the graphic window xx in the file des.ps.xx. This file can't be directly send to a Postscript printer and a set of programs (in the bin Scilab directory) are given with Scilab to print it:
BEpsf: The BEpsf command will create an Epsf file from your des.ps.xx under the name des.epsf, this Epsf file can be printed on a Postscript printer or inserted into an other Postscript document.
Blatexpr: The Blatexpr command will create an Epsf file from your des.ps.xx

mv des.ps.xx des.ps
Blatexpr 1.0 1.0 des.ps

under the name des.epsf and a LaTeX file des.tex. The file des.tex can be inserted in a LaTeX file in order to get the latex figure as follows (the postscript file is inserted with the special command of LaTeX)
\input des.tex
\dessin{caption}{label}

Blpr: The Blpr command is used to print a set of graphics on a same sheet of paper. For example to print two graphics on a unique page, one can use:
Blpr "Two graphics" file1.ps.0 file2.ps.1 | lpr
Blatexprs: The Blatexprs command is used to insert in a single LaTeX figure a set of Scilab Graphics
Blatexprs res file1.ps.0 file2.ps.1

will create two files res.ps and res.tex. The file res.tex is used as in the Blatexpr command in order to get the figure.

SEE ALSO: xbasimp 137

1.2.75 replot ________ redraw the current graphics window with new boundaries

CALLING SEQUENCE:
replot(rect)

PARAMETERS:
rect : row vector of size 4.

DESCRIPTION:
replot is used to redraw the content of the current graphics window with new boundaries defined by rect=[xmin,ymin,xmax,ymax]. It works only with the driver "Rec".

EXAMPLE:
x=[0:0.1:2*pi]';
plot2d(x,sin(x))
replot([-1,-1,10,2])

SEE ALSO: xbasr 138

AUTHOR: J.Ph.C.
**1.2.76 rotate**

**Calling Sequence:**

\[ xy1 = \text{rotate}(xy, \theta, \text{orig}) \]

**Parameters:**

- \( xy \): matrix of size (2,).
- \( xy1 \): matrix of size (2,).
- \( \theta \): real, angle in radian; default value is 0.
- \( \text{orig} \): center of the rotation; default value is \([0;0]\).

**Description:**

\( \text{rotate} \) performs a rotation with angle \( \theta \):

\[ xy1(:,i) = M(\theta) \cdot xy(:,i) + \text{orig} \]

where \( M \) stands for the corresponding rotation matrix.

**Example:**

\[
\begin{align*}
\text{xsetech}([0,0,1,1],[-1,-1,1,1]) \\
\text{xy}=([0:0.1:10];\sin(0:0.1:10))/10; \\
\text{for } i=2*\pi*(0:10)/10, \\
\quad [xy1]=\text{rotate}(xy,i); \\
\quad \text{xpoly}(xy1(1,:),xy1(2,:),"lines") \\
\end{align*}
\]

**1.2.77 scaling**

**Calling Sequence:**

\[ xy1 = \text{scaling}(xy, \text{factor}, \text{orig}) \]

**Parameters:**

- \( xy1 \): matrix of size (2,).
- \( xy \): matrix of size (2,).
- \( \text{factor} \): real scalar, coefficient of the linear transformation.
- \( \text{orig} \): shift vector; default value is \([0;0]\).

**Description:**

\( \text{scaling} \) performs an affine transformation on the set of points defined by the coordinates \( xy \):

\[ xy1(:,i) = \text{factor} \cdot xy(:,i) + \text{orig}. \]

**1.2.78 sd2sci**

**Calling Sequence:**

\[ \text{txt} = \text{sd2sci}(\text{sd}[,\text{sz} [,\text{orig}]])) \]

**Parameters:**

- \( \text{sd} \): data structure build by \text{gr\_menu}.
- \( \text{sz} \): vector of number or strings with two components, give the x and y zoom factors.
orig : vector of number or strings with two components, give the origin translation vector

DESCRIPTION :
given a sd data structure generated by gr_menu sd2sci forms a vector of scilab instructions corresponding to the graphic edited by gr_menu.

The optional parameters sz and orig allows to zoom and shift the initial graphic.

If sz or orig are given by strings generated instructions are relative use then as formal expressions.

AUTHOR : Serge Steer INRIA 1988

SEE ALSO : gr_menu 107, execstr 37

1.2.79  secto3d  3D surfaces conversion

CALLING SEQUENCE :

[m[,x]]=secto3d(seclist,npas)
[m]=secto3d(seclist ,x)

PARAMETERS :

seclist : a list whose elements are (2,.) matrices
npas : an integer
m : a matrix
x : a vector

DESCRIPTION :
Considering a surface given through a list seclist of sections in the (x,z) plane

[m [,x]]=secto3d(seclist [,npas]) returns a matrix m which contains a regular discretization of the surface.

- The i-th row of the matrix m corresponds to the i-th section
- The j-th column of m corresponds to the x(j)

Each section seclist(i) is described by a (2,.) matrix which gives respectively the x and z coordinates of points.

[m]=secto3d(seclist ,x) : in that case the x-vector gives the discretization of the x-axis for all the sections

SEE ALSO : plot3d 123

AUTHOR : Steer S.

1.2.80  sgrid  s-plane grid lines.

CALLING SEQUENCE :

sgrid()
sgrid('new')
sgrid(zeta,wn [,color])

DESCRIPTION :
Used in conjunction with evans, plots lines of constant damping ratio (zeta) and natural frequency (wn).

sgrid() : add a grid over an existing continuous s-plane root with default values for zeta and wn.
sgrid('new') : clears the graphic screen and then plots a default s-plane grid
sgrid(zeta,wn [,color]) : same as sgrid() but uses the provided damping ratio and natural frequency.
EXAMPLE:
H=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
evans(H,100)
sgrid()
sgrid(0.6,2,7)

SEE ALSO: evans 98

1.2.81 square _______ set scales for isometric plot (change the size of the window)

CALLING SEQUENCE:
square(xmin,ymin,xmax,ymax)

PARAMETERS:
xmin,xmax,ymin,ymax : four real values

DESCRIPTION:
square is used to have isometric scales on the x and y axes. The requested values xmin, xmax, ymin, ymax are the boundaries of the graphics frame and square changes the graphics window dimensions in order to have an isometric plot. square set the current graphics scales and can be used in conjunction with graphics routines which request the current graphics scale (for instance fstrf="x0z" in plot2d).

EXAMPLE:
t=[0:0.1:2*%pi]';
plot2d(sin(t),cos(t))
xbasc()
square(-1,-1,1,1)
plot2d(sin(t),cos(t))
xset("default")

SEE ALSO: isoview 111. xsetech 155

AUTHOR: Steer S.

1.2.82 subplot _______ divide a graphics window into a matrix of sub-windows

CALLING SEQUENCE:
subplot(m,n,p)
subplot(mnp)

PARAMETERS:
m, n, p : positive integers
mnp : an integer with decimal notation mnp

DESCRIPTION:
subplot(m,n,p) or subplot(mnp) breaks the graphics window into an m-by-n matrix of sub-windows and selects the p-th sub-window for drawing the current plot. The number of a sub-window into the matrices is counted row by row ie the sub-window corresponding to element (i,j) of the matrix has number (i-1)*m + j.

EXAMPLE:
```scilab
subplot(221)
plot2d()
subplot(222)
plot3d()
subplot(2,2,3)
param3d()
subplot(2,2,4)
hist3d()

SEE ALSO: plot2d 118, plot3d 123, xstring 157, xtitle 159

1.2.83 titlepage ________________ add a title in the middle of a graphics window

CALLING SEQUENCE:

titlepage(str)

PARAMETERS:

str : matrix of strings

DESCRIPTION:

titlepage displays the matrix of strings str in the middle of the current graphics window with a font as big as possible.

SEE ALSO: xtitle 159

AUTHOR: S. S.

1.2.84 winsid ________________________ return the list of graphics windows

CALLING SEQUENCE:

x=winsid()

PARAMETERS:

x : row vector.

DESCRIPTION:

winsid is used to get the list of graphics windows as a vector of windows numbers.

1.2.85 xarc ___________________________ draw a part of an ellipse

CALLING SEQUENCE:

xarc(x,y,w,h,a1,a2)

PARAMETERS:

x,y,w,h : four real values defining a rectangle.
a1,a2 : real values defining a sector.
```
DESCRIPTION:
xarc draws a part of an ellipse contained in the rectangle \((x, y, w, h)\) (upper-left point, width, height), and in the sector defined by the angle \(alpha_1\) and the angle \(alpha_1+alpha_2\). \(alpha_1\) and \(alpha_2\) are given respectively by \(a1/64\) degrees and \(a2/64\) degrees. This function uses the current graphics style and scale.

EXAMPLE:

// isoview scaling
plot2d(0,0,-1,"031",",",[-2,-2,2,2])
xset("dashes",3)
xarc(-1,1,2,2,0,90*64)
xarc(-1.5,1.5,3,3,0,360*64)

SEE ALSO:  xarcs [135], xfarc [141], xfarcs [142]

AUTHOR: J.Ph.C.

1.2.86 xarcs ____________________________ draw parts of a set of ellipses

CALLING SEQUENCE:

xarcs(arcs,[style])

PARAMETERS:

arcs: matrix of size (6,n) describing the ellipses.
style: row vector of size n giving the style to use.

DESCRIPTION:
xarcs draws parts of a set of ellipses described by arcs: arcs=[x y w h a1 a2;x y w h a1 a2;...]' where each ellipse is defined by the 6 parameters \((x, y, w, h, a1, a2)\) (see xarc). style(i) gives the dash style used to draw ellipse number i.

EXAMPLE:

plot2d(0,0,-1,"031",",",[-1,-1,1,1])
arcs=[-1.0 0.0 0.5; // upper left x
     1.0 0.0 0.5; // upper left y
     0.5 1.0 0.5; // width
     0.5 0.5 1.0; // height
     0.0 0.0 0.0; // angle 1
     180*64 360*64 90*64]; // angle 2
xarcs(arcs,[1,2,3])

SEE ALSO:  xarc [134], xfarc [141], xfarcs [142]

AUTHOR: J.Ph.C.

1.2.87 xarrows ________________________________ draw a set of arrows

CALLING SEQUENCE:

xarrows(nx,ny,[arsize,style])

PARAMETERS:

nx, ny: real vectors or matrices of same size.
arsize : real scalar, size of the arrow head. The default value can be obtained by setting arsize to -1.

style : matrix or scalar. If style is a positive scalar it gives the dash style to use for all arrows. If it is a negative scalar then the current dash style is used. If it is a vector style(i) gives the style to use for arrow i.

DESCRIPTION :

xarrows draws a set of arrows given by nx and ny. If nx and ny are vectors, the ith arrow is defined by (nx(i), ny(i)) -- (nx(i+1), ny(i+1)). If nx and ny are matrices:

nx=[xi_1 x1_2 ...; xf_1 xf_2 ...]
ny=[yi_1 y1_2 ...; yf_1 yf_2 ...]

the kth arrow is defined by (xi_k, yi_k) -- (xf_k, yf_k).

xarrows uses the current graphics scale which can be set by calling a high level drawing function such as plot2d.

EXAMPLE :

x=2*%pi*(0:9)/8;
x1=[sin(x);9*sin(x)];
y1=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022")
xset("clipgrf")
xarrows(x1,y1,1:10)
xset("clipoff")

AUTHOR : J.Ph.C.

1.2.88 xaxis ---------------------------------------- draw an axis

CALLING SEQUENCE :

xaxis(alpha,nsteps,size,init)

PARAMETERS :

alpha : real, slope in degree of the axis.
nsteps : real vector of size 2, number of big and small intervals.
size : real vector of size 3, size of the small intervals, and small and big tics.
init : real vector of size 2, origin of the axis.

DESCRIPTION :

xaxis draws an axis.

The direction of the axis is given by alpha in degree.

init=[x0 y0] is the initial point of the axis.
nsteps=[n1,n2] gives the number of big and small intervals separated by tics.
size=[s1,s2,c1] where s1 gives the size of the small intervals, s2 gives the size of the small tics along the axis and s2*c1 gives the size of the big tics. All the sizes are given using the current x-scale and y-scale and are given as dimensions along the drawn axis.

example : n1=3, n2=2, alpha=0

(s2*c1)
|     |     |     |     |
(s2)
|_____|_____|_____|____|
s1
EXAMPLE:

x = [-%pi:0.1:%pi]';
// plot without axis
plot2d(x, sin(x), 1, "010", "", [-4 -1 4 1])
// draw x axis
xpoly([-4 4], [0 0], "lines")
axis([0, [2 2], [2 0.1 3], [-4 0]])
xstring(-4.1, -0.25, "-4"); xstring(-0.2, -0.1, "0"); xstring(4, -0.25, "4")
// draw y axis
xpoly([0 0], [-1 1], "lines")
axis([0, [2 2], [0.5 0.025 3], [0 1]])
xstring(-0.5, -1.05, "-1"); xstring(-0.35, 0.95, "1")

AUTHOR: J.Ph.C.

1.2.89  xbasc  clear a graphics window and erase the associated recorded graphics

CALLING SEQUENCE:

xbasc([window-id])

PARAMETERS:

window-id: integer scalar or vector

DESCRIPTION:
Without any argument, this function clears the current graphics window and erases the recorded graphics. Otherwise it clears the graphics windows whose numbers are included in the vector window-id, and erases the corresponding recorded graphics. For example xbasc(1:3) clears windows 1, 2 and 3 and erases the corresponding recorded graphics. If one of the windows does not exist, then it is automatically created.

SEE ALSO:  xclear 139

1.2.90  xbasimp  send graphics to a Postscript printer or in a file

CALLING SEQUENCE:

xbasimp(win_num, [filen, printer])

PARAMETERS:

win_num: integer scalar or vector
filen: string. Postscript file name (default value is "file"). The window number is appended to filen.
printer: string. printer name. If printer is present or if there is only one argument in the calling sequence, the created file is printed on printer printer.

DESCRIPTION:
xbasimp sends the recorded graphics of the window win_num into the Postscript file filen and prints the Postscript file with the command Blpr. This function works only if the selected driver is "Rec". If win_num is a vector, several files are generated, one for each selected window (with names filenxx), and the files are printed on a unique page with the command Blpr. The window number is appended to filen.

SEE ALSO:  printing 129,  xs2fig 152

Scilab Group  April 1993  137
1.2.91  xbasr -------------------------------- redraw a graphics window

CALLING SEQUENCE :

xbasr(win_num)

DESCRIPTION :
xbasr is used to redraw the content of the graphics window with id win_num. It works only with the driver "Rec".
SEE ALSO:  driver 95, replot 130, xtape 159

AUTHOR : J.Ph.C.

1.2.92  xchange ----------------------------- transform real to pixel coordinates

CALLING SEQUENCE :

[x1,y1,rect]=xchange(x,y,dir)

PARAMETERS :

x,y : two matrices of size (n1,n2) (coordinates of a set of points).
x1,y1 : two matrices of size (n1,n2) (coordinates of the set of points).
rect : a vector of size 4.

DESCRIPTION :
After having used a graphics function, xchange computes pixel coordinates from real coordinates and conversely, according to the value of the parameter dir: "f2i" (float to int) means real to pixel and "i2f" (int to float) means pixel to real. x1 and y1 are the new coordinates of the set of points defined by the old coordinates x and y.
rect is the coordinates in pixel of the rectangle in which the plot was done: [upper-left point, width, height].

EXAMPLE :

```
t=[0:0.1:2*%pi]’;
plot2d(t,sin(t))
[x,y,rect]=xchange(1,1,"f2i")
[x,y,rect]=xchange(0,0,"i2f")
```

AUTHOR : J.Ph.C.

1.2.93  xclea -------------------------------- erase a rectangle

CALLING SEQUENCE :

xclea(x,y,w,h)

PARAMETERS :

x,y,w,h : real values defining the rectangle.

DESCRIPTION :
xclea clears the rectangle [x,y,w,h] (upper left point, width, height) in the current graphics window.

EXAMPLE :

Scilab Group         April 1993
x=[0:0.1:2*%pi]’;
plot2d(x,sin(x))
xcla(1,1,1,1)

1.2.94 xclear ............................ clear a graphics window

CALLING SEQUENCE:

xclear([window-id])

PARAMETERS:

window-id : integer scalar or vector

DESCRIPTION:

Without any argument, this function clears the current window. Otherwise it clears the graphics windows whose numbers are included in the vector window-id. For example xclear(1:3) clears windows 1, 2 and 3. If one of the windows does not exist, then it is automatically created.

Warning: in recording mode xclear clears the window, but it does not erase the recorded commands. In this case you must use the function xbasc.

SEE ALSO: xbasc 137

1.2.95 xclick .............................. wait for a mouse click

CALLING SEQUENCE:

[c_i,c_x,c_y,c_w,c_m]=xclick([flag])

PARAMETERS:

c_i : integer, mouse button number.

Real scalars, position of the mouse.

c_w : integer, window number.

String, menu callback.

flag : integer. If present, the click event queue is not cleared when entering xclick.

DESCRIPTION:

xclick waits for a mouse click in the graphics window.

If it is called with 3 left hand side arguments, it waits for a mouse click in the current graphics window. If it is called with 4 or 5 left hand side arguments, it waits for a mouse click in any graphics window.

The returned values are described below.

c_i : an integer which gives the number of the mouse button that was pressed 0, 1 or 2 (for left, middle and right) or -1 in case of problems with xclick.

c_x, c_y : the coordinates of the position of the mouse click in the current graphics scale.

c_w : the window number where the click has occurred.

c_m : string associated with a dynamic menu. If xclick returns due to a click on a menu, c_i, c_x, c_y, and c_w take arbitrary values.

KNOWN TROUBLES:

xclick can return the message "Can’t grab the pointer” if the graphics window is iconified when calling it.

SEE ALSO: locate 112, xgetmouse 145

AUTHOR: J.Ph.C.
1.2.96 xclip ___________________________________________ set a clipping zone

CALLING SEQUENCE:

xclip([x,y,w,h])
xclip(rect)
xclip("clipgrf")

PARAMETERS:

x, y, w, h : real values.
rect : row vector of size 4.

DESCRIPTION:

xclip set a clipping zone given by the coordinates, in the current graphics scale, of the rectangle x, y, w, h (upper-left point, width, height). If only one argument is given, it stands for a rectangle specification rect=[x, y, w, h]. xclip("clipgrf") is used to clip the usual rectangle boundaries. To unclip a region use the command xclip().

EXAMPLE:

x=0:0.2:2*%pi;
x1=[sin(x);100*sin(x)];
y1=[cos(x);100*cos(x)];
y1=y1+20*ones(y1);
// No clip
plot2d([-100,500],[-100,600],[-1,-1],"022")
xsegs(10*x1+200*ones(x1),10*y1+200*ones(y1))
// rectangle clipping zone
xbasc(); plot2d([-100,500],[-100,600],[-1,-1],"022")
xrect(150,460,100,150)
xclip(150,460,100,150)
xsegs(10*x1+200*ones(x1),10*y1+200*ones(y1))
// usual rectangle boundaries clipping zone
xbasc(); plot2d([-100,500],[-100,600],[-1,-1],"022")
xclip("clipgrf")
xsegs(10*x1+200*ones(x1),10*y1+200*ones(y1));
// clipping of
xclip()

AUTHOR: J.Ph.C.

1.2.97 xdel ____________________________ delete a graphics window

CALLING SEQUENCE:

xdel([win-nums])

DESCRIPTION:

xdel deletes the graphics windows win-nums or the current graphics window if no argument is given.

AUTHOR: J.Ph.C.
1.2.98  **xend**                                             close a graphics session

**CALLING SEQUENCE:**

xend()

**DESCRIPTION:**

*xend* is used to close a graphics session. Under the Postscript, Xfig or Gif drivers *xend* closes the file which was opened by *xinit*.

**EXAMPLE:**

```plaintext
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
```

SEE ALSO:  xbasimp 137, xend 141

**AUTHOR:** J.Ph.C.

1.2.99  **xfarc**                                          fill a part of an ellipse

**CALLING SEQUENCE:**

xfarc(x,y,w,h,a1,a2)

**PARAMETERS:**

x, y, w, h : four real values defining a rectangle.
a1, a2 : real values defining a sector.

**DESCRIPTION:**

*xfarc* fills a part of an ellipse contained in the rectangle \((x, y, w, h)\) (upper-left point, width, height), and in the sector defined by the angle \(\alpha_1\) and the angle \(\alpha_1 + \alpha_2\). \(\alpha_1\) and \(\alpha_2\) are given respectively by \(a1/64\) degrees and \(a2/64\) degrees. This function uses the current graphics style and scale.

**EXAMPLE:**

```plaintext
// isoview scaling
plot2d(0,0,-1,"031","",[-2,-2,2,2])
xfarc(-0.5,0.5,1,1,0,90*64)
xset("pattern",2)
```

SEE ALSO:  xarc 134, xarcs 135, xfarcs 142

**AUTHOR:** J.Ph.C.

Scilab Group  April 1993  141
1.2.100  xfarcs  fill parts of a set of ellipses

CALLING SEQUENCE :

xfarcs(arcs,[style])

PARAMETERS :

arcs : matrix of size (6,n) describing the ellipses.
sty le : row vector of size n giving the style to use.

DESCRIPTION :

xfarcs fills parts of a set of ellipses described by
arcs: arcs=[x y w h a1 a2;x y w h a1
a2;...]' where each ellipse is defined by the 6 parameters
(x,y,w,h,a1,a2) (see xfarc).
sty le(i) gives the dash style used to draw ellipse number i.

EXAMPLE :

plot2d(0,0,-1,"031"," ",[-1,-1,1,1])
arcs=[-1.0 0.0 0.5; // upper left x
1.0 0.0 0.5; // upper left y
0.5 1.0 0.5; // width
0.5 0.5 1.0; // height
0.0 0.0 0.0; // angle 1
180*64 360*64 90*64]; // angle 2
xfarcs(arcs,[1,2,3])

SEE ALSO:  xarc134,  xfarc141,  xfarc141

AUTHOR : J.Ph.C.

1.2.101  xfpoly  fill a polygon

CALLING SEQUENCE :

xfpoly(xv,yv,[close])

PARAMETERS :

xv,yv : two vectors of same size (the points of the polygon).
close : integer. If close=1, the polyline is closed; default value is 0.

DESCRIPTION :

xfpoly fills a polygon with the current pattern. If close is equal to 1 a
point is added to the polyline xv,yv to define a polygon.

EXAMPLE :

x=sin(2*%pi*(0:5)/5);
y=cos(2*%pi*(0:5)/5);
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("pattern",5)
xfpoly(x,y)
xset("default")

SEE ALSO:  xfpolys143,  xpoly150,  xpolys150

AUTHOR : J.Ph.C.
1.2.102  xfpolys _______________________________ fill a set of polygons

CALLING SEQUENCE:

xfpolys(xpols,ypols,[fill])

PARAMETERS:

xpols,ypols : matrices of the same size (p,n) (points of the polygons).
fill : vector of size n.

DESCRIPTION:

xfpolys fills a set of polygons of the same size defined by the two matrices xpols and ypols. The coordinates of each polygon are stored in a column of xpols and ypols. The pattern for filling polygon number i is given by fill(i):

- if fill(i)<0, the polygon is filled with pattern id -fill(i).
- if fill(i)=0, the polygon is drawn with the current dash style (or current color).
- if fill(i)>0, the polygon is filled with pattern id fill(i). Then its contour is drawn with the current dash (or color) and closed if necessary.

EXAMPLE:

plot2d(0,0,[-1],"012","",[0,-10,210,40])
x1=[0,10,20,30,20,10,0]';
y1=[15,30,30,15,0,0,15]';
xpols=[x1 x1 x1 x1]; xpols=xpols+[0,60,120,180].*ones(x1);
ypols=[y1 y1 y1 y1];
// setting the current dash (or line color)
xset("dashes",5)
xfpolys(xpols,ypols,[-1,0,1,2])
xset("default")

See Also:  xfpoly142, xpoly150, xpolys150

Author: J.Ph.C.

1.2.103  xfrect _______________________________ fill a rectangle

CALLING SEQUENCE:

xfrect(x,y,w,h)
xfrect(rect) // rect = [x,y,w,h]

PARAMETERS:

x,y,w,h : four real values defining the rectangle.

DESCRIPTION:

xfrect fills a rectangle defined by [x,y,w,h] (upper-left point, width, height) using the current scale and style.

EXAMPLE:

plot2d(0,0,-1,"010","",[-2,-2,2,2])
xset("pattern",5)
xfrect(-1,1,2,2)
xset("default")

See Also:  xrect151, xrects151

Author: J.Ph.C.
CALLING SEQUENCE:

[x1]=xget(str,[flag])

xget()  

PARAMETERS:

str : string.
flag : optional. Set to 1 gives a verbose mode.

DESCRIPTION:

This function is used to get values from the graphics context on the topic specified by the string str.
When called with no argument, a choice menu is created showing the current values and changes can be performed through toggle buttons.

number=xget("alufunction") : Get the logical function number used for drawing. See xset.
str=xset("auto clear") : Get the auto clear status ("on" or "off").
color=xget("background") : Get the background color of the current graphics window.
rect=xget("clipping") : Get the clipping zone as a rectangle rect=[x,y,w,h] (Upper-Left point Width Height).
c=xget("color")  : Get the default color for filling, line or text drawing functions. c is an integer projected in the interval [0,whiteid]. 0 stands for black filling and whiteid for white. The value of whiteid can be obtained with xget("white").
cmap=xget("colormap") : Get the colormap used for the current graphics window as a m x 3 RGB matrix.
dash=xget("dashes") : Get the dash style dash=[dash_number] where dash_number is the id of the dash. This keyword is obsolete, please use xget("color") or xget("line style") instead.
font=xget("font") : Get font=[fontid,fontsize], the default font and the default size for fonts. size.
fontsize=xget("font size") : Get the default size for fonts size.
color=xget("foreground") : Get the foreground color of the current graphics window.
str=xget("fpf") : Get the floating point format for number display in contour functions. Note that str is "" when default format is used.
color=xget("hidden3d") : Get the color number for hidden faces in plot3d.
pat=xget("lastpattern") : Get the id of the last available pattern or color, with the current colormap of the current window. In fact pat+1 and pat+2 are also available and stand respectively for black and white pattern.
type=xget("line mode") : Get the line drawing mode. type=1 is absolute mode and type=0 is relative mode. (Warning: the mode type=0 is has bugs)
xget("line style") : Get the default line style (1: solid, >1 for dashed lines).
mark=xget("mark") : Get the default mark id and the default mark size. mark=[markid,marksize].
marksize=xget("mark size") : Get the default mark size.
pat=xget("pattern") : Get the current pattern or the current color. pat is an integer in the range [1,last]. When one uses black and white, 0 is used for black filling and last for white. The value of last can be obtained with xget("lastpattern").
value=xget("thickness") : Get the thickness of lines in pixel (0 and 1 have the same meaning: 1 pixel thick).
flag=xget("use color") : Get the flag 0 (use black and white) or 1 (use colors). See xset.
[x,y]=xget("viewport") : Get the current postion of the visible part of graphics in the panner.
dim=xget("wdim") : Get the width and the height of the current graphics window dim=[width,height].
win=xget("window") : Get the current window number win.
pos=xget("wpos",x,y) ; : Get the position of the upper left point of the graphics window pos=[x,y].

SEE ALSO:  xset 154,  colormap 89

AUTHOR : J.Ph.C.
**1.2.105  xgetech  get the current graphics scale**

**CALLING SEQUENCE:**

\[ [\text{wrect}, \text{frect}, \text{logflag}, \text{arect}] = \text{xgetech()} \]

**PARAMETERS:**

- \text{wrect, frect} : real vectors.
- \text{logflag} : string of size 2"xy".

**DESCRIPTION:**

\text{xgetech} returns the current graphics scale (of the current window). The rectangle \([\text{xmin, ymin, xmax, ymax}]\) given by \text{frect} is the size of the whole graphics window. The plotting will be made in the region of the current graphics window specified by \text{wrect}.

\text{wrect}=[\text{x, y, w, h}] \ (upper-left \ point, \ width, \ height) \ describes \ a \ region \ inside \ the \ graphics \ window. \ The \ values \ in \ \text{wrect} \ are \ specified \ using \ proportion \ of \ the \ width \ and \ height \ of \ the \ graphics \ window:

- \text{wrect}=[0 0 1 1] means that the whole graphics window is used.
- \text{wrect}=[0.5 0 0.5 1] means that the graphics region is the right half of the graphics window.

\text{logflag} is a string of size 2"xy", where \text{x} and \text{y} can be "n" or "l". "n" stands for normal and "l" stands for logscale. \text{x} stands for the \text{x}-axis and \text{y} stands for the \text{y}-axis.

\text{arect}=[\text{x_left, x_right, y_up, y_down}] gives the frame size inside the subwindow. The graphic frame is specified (like \text{wrect}) using proportion of the width or height of the current graphic subwindow. Default value is \(1/8*[1,1,1,1]\). If \text{arect} is not given, current value remains unchanged.

**EXAMPLE:**

```scilab
// first subwindow
xsetech([0,0,1.0,0.5])
plot2d()
// then xsetech is used to set the second sub window
xsetech([0,0.5,1.0,0.5])
greyplot()
// get the graphic scales of first subwindow
xsetech([0,0,1.0,0.5])
[wrect,frect,logflag,arect]=xgetech();
// get the graphic scales of second subwindow
xsetech([0,0.5,1.0,0.5])
[wrect,frect,logflag,arect]=xgetech();
xvsc();
xset(’default’)```

**SEE ALSO:**  \text{xsetech 155}

**AUTHOR:** J.Ph.C.

**1.2.106  xgetmouse  get the current position of the mouse**

**CALLING SEQUENCE:**

\[ \text{rep} = \text{xgetmouse([flag])} \]

**PARAMETERS:**

- \text{rep} : vector of size 3, [\text{x, y, ibutton}].
- \text{flag} : integer. If present, the click event queue is not cleared when entering \text{xgetmouse}.

Scilab Function  Scilab Group  145
DESCRIPTION:
If the mouse is located in the current graphics window, xgetmouse returns in rep the current mouse position (x,y) and the value ibutton.

The ibutton value indicates the action of the button at this point:

- if ibutton is -1 then no button was clicked.
- if ibutton is -5,-4 or -2 then left, middle or right button was released
- if ibutton is 0, 1 or 2, then the left, middle or right button was pressed.

If the mouse is not located in the current graphics window, xgetmouse waits.

EXAMPLE:

xselect(); xbasc(); xsetech([0 0 1 1],[0 0 100 100])
xset("alufunction",6)
xtitle(" drawing a rectangle ")
[b,x0,y0]=xclick(); rep=[x0,y0,-1]; x=x0; y=y0;
xrect(x0,y0,x-x0,y-y0)
while rep(3)==-1 then
    rep=xgetmouse(0)
xrect(x0,y0,x-x0,y0-y)
x=rep(1); y=rep(2);
xrect(x0,y0,x-x0,y0-y)
end
xset("alufunction",3)

SEE ALSO: locate 112, xclick 139

1.2.107 xgraduate .............................................. axis graduation

CALLING SEQUENCE:

[xi, xa, np1, np2, kMinr, kMaxr, ar]=xgraduate(xmi, xma)

PARAMETERS:

xmi, xma : real scalars
xi, xa, kMinr, kMaxr, ar : real scalars
np1, np2 : integer

DESCRIPTION:

xgraduate returns the axis graduations which are used by the plot routines (with pretty print flag enabled). It returns an interval [xi, xa] which contains the given interval [xmi, xma] and such that xi= kMinr*10^ar, xa=kMaxr*10^ar and the interval can be divided into np2 intervals and each interval is divided in np1 sub-intervals.

EXAMPLE:

[x1, xa, np1, np2, kMinr, kMaxr, ar]=xgraduate(-0.3, 0.2)

SEE ALSO: graduate 108, plot2d 118

AUTHOR: S. Steer

AUTHOR: J.P.C

Scilab Group April 1993 146
1.2.108  \texttt{xgrid} \hfill \text{add a grid on a 2D plot}\\

\textbf{CALLING SEQUENCE:} \\
xgrid([\text{style}])\\

\textbf{PARAMETERS:} \\
\text{style} : \text{integer}\\

\textbf{DESCRIPTION:} \\
xgrid adds a grid on a 2D plot. style is the dash id or the color id to use for the grid plotting. Use \texttt{xset()} for the meaning of id. \\

\textbf{EXAMPLE:} \\
x=[0:0.1:2*%pi]’; \\
plot2d(sin(x)) \\
xgrid(2)\\

\textbf{SEE ALSO:} \texttt{xset 154}, \texttt{plot2d 118}\\

\textbf{AUTHOR: J.Ph.C.}\\

1.2.109  \texttt{xinfo} \hfill \text{draw an info string in the message subwindow}\\

\textbf{CALLING SEQUENCE:} \\
xinfo(\text{info})\\

\textbf{PARAMETERS:} \\
\text{info} : \text{string}\\

\textbf{DESCRIPTION:} \\
xinfo draws the string \text{info} in the message subwindow of the current graphics window.\\

1.2.110  \texttt{xinit} \hfill \text{initialisation of a graphics driver}\\

\textbf{CALLING SEQUENCE:} \\
xinit([\text{driver-name}])\\

\textbf{PARAMETERS:} \\
\text{driver-name} : \text{string.}\\

\textbf{DESCRIPTION:} \\
Initialisation of the given driver. \\
For X Window \text{driver-name} can be a string which gives the name of a display and a new graphics window is created. If the argument is omitted the value of the environment variable DISPLAY is used if it exists or the value “unix:0.0” is used. \\
For the Postscript, Xfig or Gif driver, \text{driver-name} is a name of the file where all the graphics operations are recorded. \\

\textbf{EXAMPLE:} \\

\text{Scilab Group April 1993 147}
driver("Pos")
xinit("foo.ps")
plot2d()
xend()
driver("X11")

SEE ALSO: xbasimp 137, xend 141

AUTHOR: J.Ph.C.

1.2.111 xlfont _________ load a font in the graphics context or query loaded font

CALLING SEQUENCE:

xlfont(font-name, font-id)
fonts=xlfont()

PARAMETERS:

font-name: string, name of the font family.
font-id: integer between 0 and 6.
fonts: a column vector of currently loaded font names.

DESCRIPTION:
Without any argument, xlfont() returns the list of currently loaded fonts.
With arguments, xlfont is used to load a new font at different sizes in the graphics context. The font must be available with size 8, 10, 12, 14, 18 and 24.
Default fonts are "Courier Roman" (0), "Symbol" (1), "Times Roman" (2), "Times Italic" (3), "Times Bold" (4) and "Times Bold Italic" (5).
font-name can be of 2 types:
- if it contains the character "%", it is supposed to be an X11 font name with %s in the size field of the name, for example "-b&h-lucidabright-demibold-r-normal--%-s=-75-75-p-*=iso8859-1"
- if it does not contain the character "%", it is supposed to be an alias name and the fonts aliased by font-name08,...,font-name24 are loaded.
font-id: give the id font-id to the loaded font font-name.

SEE ALSO: xset 154

AUTHOR: J.Ph.C.

1.2.112 xload ______________________________ load a saved graphics

CALLING SEQUENCE:

xload(file-name, [win-num])

PARAMETERS:

file-name: string, name of the file.
win-num: integer, the graphics window number. If not given, the current graphics window is used.

DESCRIPTION:

xload reloads the graphics contained in the file file-name in the graphics window win-num.

SEE ALSO: xsave 152

AUTHOR: J.Ph.C.
1.2.113  \textbf{xname} \hspace{1cm} \texttt{\underline{c\textit{hange the name of the current graphics window}}}

\textbf{CALLING SEQUENCE:}

\texttt{xname(name)}

\textbf{PARAMETERS:}

\texttt{name} : string, new name of the graphics window.

\textbf{DESCRIPTION:}

\texttt{xname} changes the name of the current graphics window.

\textbf{AUTHOR:} J.Ph.C.

1.2.114  \textbf{xnumb} \hspace{1cm} \texttt{\underline{draw numbers}}

\textbf{CALLING SEQUENCE:}

\texttt{xnumb(x,y,nums,[box,angle])}

\textbf{PARAMETERS:}

\texttt{x,y,nums} : vectors of same size.
\texttt{box} : integer value.
\texttt{angle} : optional vector of same size as \texttt{x}

\textbf{DESCRIPTION:}

\texttt{xnumb} draws the value of \texttt{nums(i)} at position \texttt{x(i),y(i)} in the current scale. If \texttt{box} is 1, a box is drawn around the numbers. If \texttt{angle} is given, it gives the direction for string drawing.

\textbf{EXAMPLE:}

\texttt{plot2d([-100,500],[-100,600],[-1,-1],"022")}
\texttt{x=0:100:200;}
\texttt{xnumb(x,500*ones(x),[10,20,35],1)}

\texttt{SEE ALSO: xstring 157}

\textbf{AUTHOR:} J.Ph.C.

1.2.115  \textbf{xpause} \hspace{1cm} \texttt{\underline{suspend Scilab}}

\textbf{CALLING SEQUENCE:}

\texttt{xpause(microsecs)}

\textbf{DESCRIPTION:}

\texttt{xpause} suspends the current process for the number of microseconds specified by the argument. The actual suspension time may be longer because of other activities in the system, or because of the time spent in processing the call.

\textbf{AUTHOR:} J.Ph.C.
### 1.2.116 xpoly  
**draw a polyline or a polygon**

**CALLING SEQUENCE:**
```plaintext```
xpoly(xv,yv [,dtype [,close]])
```

**PARAMETERS:**
- `xv, yv`: matrices of the same size (points of the polyline).
- `dtype`: string (drawing style). default value is "lines".
- `close`: integer. If close=1, the polyline is closed; default value is 0.

**DESCRIPTION:**
`xpoly` draws a single polyline described by the vectors of coordinates `xv` and `yv`. If `xv` and `yv` are matrices they are considered as vectors by concatenating their columns. `dtype` can be "lines" for using the current line style or "marks" for using the current mark to draw the polyline.

**EXAMPLE:**
```plaintext```
x=sin(2*%pi*(0:5)/5);
y=cos(2*%pi*(0:5)/5);
plot2d(0,0,-1,"010"," ",[-2,-2,2,2])
xset("dashes",5)
xpoly(x,y,"lines",1)
xset("default")
```

**SEE ALSO:** xfpoly 142, xfpolys 143, xpolys 150

**AUTHOR:** J.Ph.C.

### 1.2.117 xpolys  
**draw a set of polylines or polygons**

**CALLING SEQUENCE:**
```plaintext```
xpolys(xpols,ypols,[draw])
```

**PARAMETERS:**
- `xpols,ypols`: matrices of the same size (p,n) (points of the polylines).
- `draw`: vector of size n.

**DESCRIPTION:**
`xpolys` draws a set of polylines using marks or dashed lines. The coordinates of each polyline are stored in a column of `xpols` and `ypols`. The style of polyline i is given by `draw(i)`:  
- If `draw(i)` is negative, the mark with id `-draw(i)` is used to draw polyline i (marks are drawn using the current pattern). Use xset() to see the meaning of the ids.
- If `draw(i)` is strictly positive, the line style (or color) with id `abs(draw(i))` is used to draw polyline i. Use xset() to see the meaning of the ids.

**EXAMPLES:**
```plaintext```
plot2d(0,0,-1,"012"," ",[0,0,1,1])
rand("uniform")
xset("pattern",3)
xpolys(rand(3,5),rand(3,5),[-1,-2,0,1,2])
xset("default")
```

**SEE ALSO:** xfpoly 142, xfpolys 143, xpoly 150

**AUTHOR:** J.Ph.C.
1.2.118 \texttt{xrect} \hspace{1cm} \text{draw a rectangle}

**CALLING SEQUENCE:**

\begin{verbatim}
  xrect(x, y, w, h)
  xrect(rect) // rect = [x, y, w, h]
\end{verbatim}

**PARAMETERS:**

\begin{itemize}
  \item \texttt{x, y, w, h} : four real values defining the rectangle.
\end{itemize}

**DESCRIPTION:**

\texttt{xrect} draws a rectangle defined by \([x, y, w, h]\) (upper-left point, width, height) using the current scale and style.

**EXAMPLE:**

\begin{verbatim}
plot2d(0, 0, -1, "010", ",([-2, -2, 2, 2])
xset("pattern", 5)
xrect(-1, 1, 2, 2)
xset("default")
\end{verbatim}

\textbf{SEE ALSO:} \texttt{xfrect 143}, \texttt{xrects 151}

\textbf{AUTHOR:} J.Ph.C.

1.2.119 \texttt{xrects} \hspace{1cm} \text{draw or fill a set of rectangles}

**CALLING SEQUENCE:**

\begin{verbatim}
xrects(rects, [fill])
\end{verbatim}

**PARAMETERS:**

\begin{itemize}
  \item \texttt{rects} : matrix of size (4,n).
  \item \texttt{fill} : vector of size n.
\end{itemize}

**DESCRIPTION:**

\texttt{xrects} draws or fills a set of rectangles. Each column of \texttt{rects} describes a rectangle (upper-left point, width, height): \texttt{rects=[x1 y1 w1 h1; x2 y2 w2 h2; ...']}. \texttt{fill(i)} gives the pattern to use for filling or drawing rectangle \(i\):

- if \(fill(i)<0\), rectangle \(i\) is drawn using the line style (or color) \(-fill(i)\)
- if \(fill(i)>0\), rectangle \(i\) is filled using the pattern (or color) \(fill(n)\)
- if \(fill(i)=0\), rectangle \(i\) is drawn using the current line style (or color).

**EXAMPLE:**

\begin{verbatim}
plot2d([-100, 500], [-50, 50], [-1, -1], "022")
cols = [-34, -33, -32, -20:5:20, 32, 33, 34];
x = 400*(0:14)/14; step = 20;
rects = [x; 10*ones(x); step*ones(x); 30*ones(x)];
xrects(rects, cols)
xnumb(x, 15*ones(x), cols)
\end{verbatim}

\textbf{SEE ALSO:} \texttt{xfrect 143}, \texttt{xrect 151}

\textbf{AUTHOR:} J.Ph.C.
1.2.120  

**xrpoly**  
__________________________    draw a regular polygon

**CALLING SEQUENCE:**

xrpoly(orig,n,r,[theta])

**PARAMETERS:**

- orig  : vector of size 2.
- n  : integer, number of sides.
- r  : real scalar.
- theta  : real, angle in radian; 0 is the default value.

**DESCRIPTION:**

xrpoly  draws a regular polygon with n  sides contained in the circle of diameter r  and with the origin of the circle set at point orig. theta  specifies a rotation angle in radian. This function uses the current graphics scales.

**EXAMPLE:**

```markdown
plot2d(0,0,-1,"012",",",[0,0,10,10])
xrpoly([5,5],5,5)
```

**SEE ALSO:**  xrect 151

1.2.121  

**xs2fig**  
__________________________    send graphics to a file in Xfig syntax

**CALLING SEQUENCE:**

xs2fig(win_num,filen,[color])

**PARAMETERS:**

- win_num  : integer scalar or vector.
- filen  : string, file name.
- color  : optional integer. 0 means black and white and 1 means color. The default value is to use a value compatible with the screen status.

**DESCRIPTION:**

xs2fig  sends the recorded graphics of the window win_num  in the file filen  in Xfig syntax. This function works only if the selected driver is "Rec".

**SEE ALSO:**  xbasimp 137

1.2.122  

**xsave**  
__________________________    save graphics into a file

**CALLING SEQUENCE:**

xsave(file-name,[win-num])

**PARAMETERS:**

- file-name  : string, name of the file.
- win-num  : integer, the graphics window number. If not given, the current graphics window is used.

**DESCRIPTION:**

xsave  saves the graphics contained in the graphics window win-num  in the binary file file-name. The graphics are stored in a machine independent way (using the xdr library) and can be reloaded with xload.

**SEE ALSO:**  xload 148

**AUTHOR:** J.Ph.C.

Scilab Group  
April 1993  
152
1.2.123  xsegs ___________________________ draw unconnected segments

CALLING SEQUENCE:

xsegs(xv,yv,[style])

PARAMETERS:

xv, yv : matrices of the same size.
style : vector or scalar. If style is a positive scalar, it gives the dash style to use for all segments. If style is a negative scalar, then current dash style is used. If style is a vector, then style(i) gives the style to use for segment i.

DESCRIPTION:

xsegs draws a set of unconnected segments given by xv and yv. If xv and yv are matrices they are considered as vectors by concatenating their columns. The coordinates of the two points defining a segment are given by two consecutive values of xv and yv:

(xv(i),yv(i))--> (xv(i+1),yv(i+1)).

For instance, using matrices of size (2,n), the segments can be defined by:

xv=[xi_1 xi_2 ...; xf_1 xf_2 ...]
yv=[yi_1 yi_2 ...; yf_1 yf_2 ...]

and the segments are (xi,k,yi,k)-->(xf,k,yf,k).

EXAMPLE:

x=2*%pi*(0:9)/8;
xv=[sin(x);9*sin(x)];
yv=[cos(x);9*cos(x)];
plot2d([-10,10],[-10,10],[-1,-1],"022")
xsegs(xv,yv,1:10)

AUTHOR : J.Ph.C.

1.2.124  xselect ___________________________ raise the current graphics window

CALLING SEQUENCE:

xselect()

DESCRIPTION:

xselect raises the current graphics window. It creates the window if none exists. If the current graphics window is iconified nothing is done.

AUTHOR : J.Ph.C.
1.2.125  xset ____________________ set values of the graphics context

CALLING SEQUENCE :

xset(choice-name,x1,x2,x3,x4,x5)
xset()

PARAMETERS :

choice-name : string
x1,...,x5 : depending on choice-name

DESCRIPTION :

xset is used to set default values of the current window graphic context.
When called no argument, a choice menu is created showing the current values and changes can be
performed through toggle buttons.

Use xset() to display or set the current color, mark and fonts used.

xset("alufunction",number) : Used to set the logical function for drawing. The logical function
used is set by x1. Usual values are: 3 for copying (default), 6 for animation and 0 for clearing. See
alufunctions for more details.

xset("auto clear","on"/"off") : Switch "on" or "off" the auto clear mode for graphics. When
the auto clear mode is "on", successive plots are not superposed, ie an xbas() operation (the
graphics window is cleared and the associated recorded graphics is erased) is performed before each
high level graphics function. Default value is "off".

xset("background",color) : Set the background color of the current graphics window.

xset("clipping",x,y,w,h) : Set the clipping zone (the zone of the graphics window where plots
can be drawn) to the rectangle (x,y,w,h) (Upper-Left point Width Height). This function uses the
current coordinates of the plot.

xset("color",value) : Set the default color for filling, line or text drawing functions. value
is an integer projected in the interval [0,whiteid]. 0 is used for black filling and whiteid for white. The
value of whiteid can be obtained with xget("white").

xset("colormap",cmap) : Set the colormap a samx3 matrix. m is the number of colors. Color
number i is given as a 3-uple cmap(i,1), cmap(i,2), cmap(i,3) corresponding respectively to red, green
and blue intensity between 0 and 1.

xset("dashes",i) : In black and white mode (xset("use color",0)), set the dash style to
style i (0 for solid line). In color mode (xset("use color",1 )) this is used to set line,
mark and text color. This keyword is obsolete, please use xset('color',i) or xset('line
style',i) instead.

xset("default") : Reset the graphics context to default values.

xset("font",fontid,fontsize) : Set the current font and its current size.
Note that fontsize applies to all fonts not only fontid.

xset("font size",fontsize) : Set the fonts size.

xset("foreground",color) : Set the foreground color of the current graphics window.

xset("fpf",string) : Set the floating point format for number display in contour functions. string
is a string giving the format in C format syntax (for example string="%.3f"). Use string="" to
switch back to default format.

xset("hidden3d",colorid) : Set the color number for hidden faces in plot3d.
colorid=0 zero suppress the drawing of backward facing faces of 3d objects. This is technically
called 'culling' and is useful for closed surfaces.

xset("line mode",type) : This function is used to set the line drawing mode. Absolute mode is
set with type=1 and relative mode with type=0. (Warning: the mode type=0 has bugs)

xset("line style",value) : Set the current line style (1: solid, >1 for dashed lines).

xset("mark",markid,marksize) : Set the current mark and the current mark size. Use xset()
to see the marks. Note that marksize applies to all marks not only markid.
**xsetech** Scilab Function

**xset**("mark size", marksize) : Set the marks size.

**xset**("pattern", value) : Set the current pattern for filling functions. value is an integer projected in the interval [0, whiteid]. 0 is used for black filling and whiteid for white. The value of whiteid can be obtained with **xget**("white"). "pattern" is equivalent to "color".

**xset**("pixmap", flag) If flag=0 the graphics are directly displayed on the screen.

If flag=1 the graphics are done on a pixmap and are sent to the graphics window with the command **xset**("wshow"). The pixmap is cleared with the command **xset**("wwpc"). Note that the usual command **xbasc()** also clears the pixmap.

**xset**("thickness", value) : Set the thickness of lines in pixel (0 and 1 have the same meaning: 1 pixel thick).

**xset**("use color", flag) If flag=1 then **xset**("pattern", .) or **xset**("dashes", .) will be used so as to change the default color for drawing or for filling patterns.

If flag=0 then we switch back to the gray and dashes mode.

**xset**("viewport", x, y) : Set the position of the panner.

**xset**("wdim", width, height) : Set the width and the height of the current graphics window. This option is not used by the postscript driver.

**xset**("wpdim", width, height) : Sets the width and the height of the current physical graphic window (which can be different from the actual size in mode wresize 1). This option is not used by the postscript driver.

**xset**("window", window-number) : Set the current window to the window number and creates the window if it does not exist.

**xset**("wpos", x, y) : Set the position of the upper left point of the graphics window.

**xset**("wresize", flag) If flag=1 then the graphic is automatically resized to fill the graphics window.

**xsetchte**();**xset**("wresize",1);**plot2d()**;**xset**("wdim",1000,500)

If flag=0 the scale of the graphic is left unchanged when the graphics window is resized. Top left panner or keyboard arrows may be used to scroll over the graphic.

**xsetchte**();**plot2d()**;**xset**("wresize",0);**xset**("wdim",1000,500)

**xset**("wshow") : See **xset**("pixmap",1) above.

**xset**("wwpc") : See **xset**("pixmap",1) above.

**SEE ALSO:** colormap 89, **xget** 144, **getcolor** 106, **getsymbol** 107

**AUTHOR:** J.Ph.C.

### 1.2.126  xsetech ________ set the sub-window of a graphics window for plotting

**CALLING SEQUENCE:**

**xsetech**(wrect, [frect, logflag])

**xsetech**(wrect=[...], frect= [...], logflag="..", arect=[...])

**xsetech**()

**PARAMETERS:**

wrect : vector of size 4, defining the sub-window to use.

frect : vector of size 4.

logflag : string of size 2 "xy", where x and y can be "n" or "l". "n" stands for normal and "l" stands for logscale. x stands for the x-axis and y stands for the y-axis.

arect : vector of size 4.
DESCRIPTION:

**xsetech** is mainly used to set the sub-window of the graphics window which will be used for plotting. The sub-window is specified with the parameter `wrect=[x,y,w,h]` (upper-left point, width, height). The values in `wrect` are specified using proportion of the width or height of the current graphic window. For instance `wrect=[0,0,1,1]` means that the whole graphics window will be used, and `wrect=[0.5,0,0.5,1]` means that the graphics region will be the right half of the graphics window.

`xsetech` also sets the current graphics scales for 2D plotting and can be used in conjunction with graphics routines which request the current graphics scale (for instance `strf="x0z"` or `frameflag=0` in `plot2d`).

`frect=[xmin,ymin,xmax,ymax]` is used to set the graphics scale and is just like the `rect` argument of `plot2d`. If `frect` is not given the current value of the graphic scale remains unchanged. The default value of `rect` is `[0,0,1,1]` (at window creation, when switching back to default value with `xset('default')` or when clearing graphic recorded events `xbasc()`).

`arect=[x_left, x_right, y_up, y_down]` is used to set the graphic frame inside the sub-window. The graphic frame is specified (like `wrect`) using proportion of the width or height of the current graphic subwindow. Default value is `1/8*[1,1,1,1]`. If `arect` is not given, current value remains unchanged.

EXAMPLE:

// To get a graphical explanation of xsetech parameters enter:
exec('SCI/demos/graphics/xsetechfig.sce');

// Here xsetech is used to split the graphics window in two parts
// first xsetech is used to set the first sub-window
// and the graphics scale
xsetech([0,0,1.0,0.5],[-5,-3,5,3])
// we call plot2d with the "001" option to use the graphics scale
// set by xsetech
plot2d([1:10]',[1:10]',1,"001"," ")
// then xsetech is used to set the second sub-window
xsetech([0,0.5,1.0,0.5])
// the graphics scale is set by xsetech to [0,0,1,1] by default
// and we change it with the use of the rect argument in plot2d
plot2d([1:10]',[1:10]',1,"011"," ",[-6,-6,6,6])
// Four plots on a single graphics window
xbasc()
xset("font",2,0)
xsetech([0,0,0.5,0.5]); plot3d()
xsetech([0.5,0,0.5,0.5]); plot2d()
xsetech([0.5,0.5,0.5,0.5]); grayplot()
xsetech([0,0.5,0.5,0.5]); histplot()
// back to default values for the sub-window
xsetech([0,0,1,1])
// One plot with changed arect
xbasc()
xset("default")
xsetech(arect=[0,0,0,0])
x=1:0.1:10;plot2d(x',sin(x)'")
xbasc()
xsetech(arect=[1/8,1/8,1/16,1/4])
x=1:0.1:10;plot2d(x',sin(x)'")
xbasc()
xset("default")
xstringb

Scilab Function

S EE A LSO :

xgetech 145,

subplot 133,

isoview 111,

square 133
AUTHOR : J.Ph.C.

1.2.127

xsetm

dialog to set values of the graphics context

CALLING SEQUENCE :

xsetm()
DESCRIPTION :

xsetm() is the same as xset().
xset 154
S EE A LSO :
AUTHOR : J.Ph.C.

1.2.128

xstring

draw strings

CALLING SEQUENCE :

xstring(x,y,str,[angle,box])
PARAMETERS :

x,y : real scalars, coordinates of the lower-left point of the strings.
str : matrix of strings.
angle : real, clockwise angle in degree; default is 0.
box : integer, default is 0.
DESCRIPTION :

xstring draws the matrix of strings str at location x,y (lower-left point) in the current graphic
scale: each row of the matrix stands for a line of text and row elements stand for words separated by a white
space. If angle is given, it gives the slope in degree used for drawing the strings. If box is 1 and angle
is 0, a box is drawn around the strings.
EXAMPLE :

plot2d([0;1],[0;1],0)
xstring(0.5,0.5,["Scilab" "is"; "not" "esilaB"])
//Other example
alphabet=["a" "b" "c" "d" "e" "f" "g" ..
"h" "i" "j" "k" "l" "m" "n" ..
"o" "p" "q" "r" "s" "t" "u" ..
"v" "w" "x" "y" "z"];
xbasc()
plot2d([0;1],[0;2],0)
xstring(0.1,1.8,alphabet)
// alphabet
xstring(0.1,1.6,alphabet,0,1) // alphabet in a box
xstring(0.1,1.4,alphabet,20) // angle
xset("font",1,1)
// use symbol fonts
xstring(0.1,0.1,alphabet)
xset("font",1,3)
// change size font
xstring(0.1,0.3,alphabet)
xset("font",1,24); xstring(0.1,0.6,"a") //big alpha
xset("default")
S EE A LSO :

titlepage 134,

xnumb 149,

xstringb 158,

xstringl 158,

xtitle 159

AUTHOR : J.Ph.C.

Scilab Group

April 1993

157


1.2.129 xstringb __________________________ draw strings into a box

CALLING SEQUENCE:

xstringb(x,y,str,w,h,[option])

PARAMETERS:

x, y, w, h : vector of 4 real scalars defining the box.
str : matrix of strings.
option : string.

DESCRIPTION:

xstringb draws the matrix of strings str centered inside the rectangle rect=[x,y,w,h] (lower-left point, width, height) in the current graphic scale. If option is given with the value "fill", the character size is computed so as to fill as much as possible in the rectangle.

Enter the command xstringb() to see a demo.

EXAMPLE:

str=["Scilab" ; "is" ; "not" ; "elisaB"];
plot2d(0,0,[-1,1],"010","",[0,0,1,1]);
r=[0,0,1,0.5];
xstringb(r(1),r(2),str,r(3),r(4),"fill");
xrect(r(1),r(2)+r(4),r(3),r(4))
r=[r(1),r(2)+r(4)+0.01,r(3),r(4)/2];
xrect(r(1),r(2)+r(4),r(3),r(4))
xstringb(r(1),r(2),str,r(3),r(4),"fill");
r=[r(1),r(2)+r(4)+0.01,r(3),r(4)/2];
xrect(r(1),r(2)+r(4),r(3),r(4))
xstringb(r(1),r(2),str,r(3),r(4),"fill");

SEE ALSO: titlepage 134, xstring 157, xstringl 158, xtitle 159

Author: J.Ph.C.

1.2.130 xstringl __________________________ compute a box which surrounds strings

CALLING SEQUENCE:

rect=xstringl(x,y,str)

PARAMETERS:

rect : vector of 4 real scalars defining the box.
x, y : real scalars, coordinates of the lower-left point of the strings.
str : matrix of strings.

DESCRIPTION:

xstringl returns in rect=[x,y,w,h] (upper-left point, width, height) the size of a rectangle in the current graphic scale which would surround the strings str drawn at location x, y (lower-left point).

The result can be approximative when using the Postscript driver.

EXAMPLE:

Scilab Group April 1993 158
plot2d([0;1],[0;1],0)
str="Scilab "is";not" "elisA";
r=xstringl(0.5,0.5,str)
xrects([r(1) r(2)+r(4) r(3) r(4)]')
xstring(r(1),r(2),str)

SEE ALSO: titlepage 134, xstring 157, xstringl 158, xtitle 159

1.2.131 xtape ______________________________ set up the record process of graphics

CALLING SEQUENCE:

xtape(str,[num,rect])

PARAMETERS:

str : string, "on", "clear", "replay" or "replaysc".
num : integer.
rect : row vector of size 4.

DESCRIPTION:

xtape is used to set up the record process of graphics:
xtape("on") just selects the driver "Rec" which records all the graphics operations.
xtape("clear",num) clears the graphics window num and clears the recorded graphics associated with
window num.
xtape("replay",num) redisplays all the recorded graphics in the window num.
xtape("replaysc",num,rect) replots the graphics window num using rect=[xmin,ymin,xmax,ymax] as
x and y bounds.

SEE ALSO: driver 95, replot 130,xbasc 137, xbasr 138

1.2.132 xtitle ______________________________ add titles on a graphics window

CALLING SEQUENCE:

xtitle(xtit,[xax,yax,encad])

PARAMETERS:

xtit,xax,yax : matrices of strings.
encad : integer value. If it is 1, a box is drawn around each title.

DESCRIPTION:

xtitle add titles on a 2D or 3D plot. xtit is the general title, xax is the title on the x-axis and yax
is the title on the y-axis. xtitle must be called after a call to a high level plotting function (plot2d,
plot3d...). If the arguments are matrices, each line of the matrices is displayed on a different line.
Enter the command xtitle() to see a demo.

SEE ALSO: titlepage 134

AUTHOR: J.Ph.C.
CALLING SEQUENCE:

zgrid()

DESCRIPTION:
plots z-plane grid lines: lines of constant damping factor (zeta) and natural frequency (Wn) are drawn in within the unit Z-plane circle.
Iso-frequency curves are shown in frequency*step on the interval [0,0.5]. Upper limit corresponds to Shannon frequency (1/dt > 2*f).
SEE ALSO: frep2tf 338, freson 339
1.3 Utilities and Elementary Functions
1.3.1  abs  .............................................................. absolute value, magnitude

CALLING SEQUENCE :

\[ t = \text{abs}(x) \]

PARAMETERS :

- \( x \) : real or complex vector or matrix
- \( t \) : real vector or matrix

DESCRIPTION :

\( \text{abs}(x) \) is the absolute value of the elements of \( x \). When \( x \) is complex, \( \text{abs}(x) \) is the complex modulus (magnitude) of the elements of \( x \).

EXAMPLE :

\[ \text{abs}([1, \%i, -1, -\%i, 1+\%i]) \]

1.3.2  acos  .............................................................. element wise cosine inverse

CALLING SEQUENCE :

\[ t = \text{acos}(x) \]

PARAMETERS :

- \( x \) : real or complex vector
- \( t \) : real or complex vector

DESCRIPTION :

The components of vector \( t \) are cosine inverse of the corresponding entries of vector \( x \). Definition domain is \([-1, 1]\).

\( \text{acos} \) takes values in:

\[ 0, \pi \times [x] - \infty + \infty \]

\[ [0] \times [0, +\infty] \text{ and } [\pi] \times -\infty, 0 \] (real x imag)

EXAMPLE :

\( x = [1, \%i, -1, -\%i] \)
\( \text{cos(acos}(x)) \)

1.3.3  acosh ............................................................. hyperbolic cosine inverse

CALLING SEQUENCE :

\[ [t] = \text{acosh}(x) \]

PARAMETERS :

- \( x \) : real or complex vector
- \( t \) : real or complex vector
DESCRIPTION:
The components of vector \( t \) are the ArgCosh of the corresponding entries of vector \( x \). Definition domain is \([1, +\infty[\). It takes his values in

\[ 0, +\infty[ \times \pi, \pi \] and \([0] \times [0, \pi] \]

EXAMPLE:
\[
x = [0, 1, \%i];
cosh(acosh(x))
\]

1.3.4 acoshm

**calling sequence:**
\[ t = \text{acoshm}(x) \]

**parameters:**
\( x, t \) : real or complex square matrix

**description:**
acoshm is the matrix hyperbolic inverse cosine of the matrix \( x \). Uses the formula \( t = \logm(x + (x + \text{eye()} \times \sqrtm((x - \text{eye()}))) \)

For non symmetric matrices result may be inaccurate.

**example:**
\[
A = [1, 2; 3, 4];
coshm(acoshm(A))
\]

**see also:** acosh, logm, sqrtm

1.3.5 acosm

**calling sequence:**
\[ t = \text{acosm}(x) \]

**parameters:**
\( x \) : real or complex square matrix
\( t \) : real or complex square matrix

**description:**
\( t \) are cosine inverse of the \( x \) matrix. Diagonalization method is used. For nonsymmetric matrices result may be inaccurate. One has \( t = -\%i \times \logm(x + \%i \times \sqrtm(\text{eye()} - x \times x)) \)

**example:**
\[
A = [1, 2; 3, 4];
cosm(acosm(A))
\]

**see also:** acos, sqrtm, logm

Scilab Group

April 1993

163
1.3.6 addf __________________________________________ symbolic addition

CALLING SEQUENCE:

```
addf("a", "b")
```

PARAMETERS:

"a", "b" : character strings

DESCRIPTION:

```
addf("a", "b") returns the character string "a+b". Trivial simplifications such as addf("0", "a") or addf("1", "2") are performed.
```

EXAMPLE:

```
addf('0', '1')
addf('1', 'a')
addf('1', '2')
'a'+'b'
```

SEE ALSO: mulf 203, subf 222, ldivf 194, rdivf 208, eval 184, evstr 35

1.3.7 adj2sp ________________________ converts adjacency form into sparse matrix.

CALLING SEQUENCE:

```
A = adj2sp(xadj, adjncy, anz)
A = adj2sp(xadj, adjncy, anz, mn)
```

PARAMETERS:

* TP 7
  * xadj
    * integer vector of length (n+1).
  * TP 7
  * adjncy
    * integer vector of length nz containing the row indices for the corresponding elements in anz
  * TP 7
  * anz
    * column vector of length nz, containing the non-zero elements of A
  * TP 7
  * mn
    * row vector with 2 entries, \fVmn=size(A)\fR (optional).
  * TP 7
  * A
    * real or complex sparse matrix (nz non-zero entries)

DESCRIPTION:

```
\fVsp2adj\fR converts an adjacency form representation of a matrix into its standard Scilab representation (utility fonction).
\fVxadj, adjncy, anz\fR = adjacency representation of \fVA\fR i.e:
  * TP
  \fVxadj(j+1)-xadj(j)\fR = number of non zero entries in row j.
  \fVadjncy\fR = column index of the non zeros entries
```

Scilab Group JUN 1997 164
in row 1, row 2, ..., row n.
\fVanz\fR = values of non zero entries in row 1, row 2, ..., row n.
\fVxadj\fR is a (column) vector of size n+1 and
\fVadjncy\fR is an integer (column) vector of size \fVnz=nnz(A)\fR.
\fVanz\fR is a real vector of size \fVnz=nnz(A)\fR.

EXAMPLE:
A = sprand(100,50,.05);
[xadj,adjncy,anz]=sp2adj(A);
[n,m]=size(A);
p = adj2sp(xadj,adjncy,anz,[n,m]);
A-p,

SEE ALSO: sp2adj 213, spcompack 215

1.3.8 amell ____________________________ Jacobi’s am function

CALLING SEQUENCE:

[sn]=amell(u,k)

PARAMETERS:

u : real scalar or vector
k : scalar
sn : real scalar or vector

DESCRIPTION:
Computes Jacobi’s elliptic function am(u,k) where k is the parameter and u is the argument. If u is a vector sn is the vector of the (element wise) computed values. Used in function \%sn.

SEE ALSO: delip 180, \%sn 446, \%asn 445

1.3.9 and ________________________________ - logical and

CALLING SEQUENCE:

b=and(A), b=and(A,**)
b=and(A,’r’), b=and(A,1)
b=and(A,’c’), b=and(A,2)
A&B

DESCRIPTION:
and(A) is the logical AND of elements of the boolean matrix A and (A) returns %T ("true") iff all entries of A are %T.
y=and(A,’r’) (or, equivalently, y=and(A,1)) is the rowwise and. It returns in each entry of the row vector y the and of the rows of x (The and is performed on the row index: y(j) = and(A(i,j),i=1,m)).
y=and(A,’c’) (or, equivalently, y=and(A,2)) is the columnwise and. It returns in each entry of the column vector y the and of the columns of x (The and is performed on the column index: y(i) = and(A(i,j),j=1,n)).

A&B gives the element-wise logical and of the booleans matrices A and B. A and B must be matrices with the same dimensions or one from them must be a single boolean.

SEE ALSO: not 204, or 205
1.3.10  asin  __________________________________________________________________ sine inverse

CALLING SEQUENCE :

\[ [t]=\text{asin}(x) \]

PARAMETERS :

x : real or complex vector/matrix  
t : real or complex vector/matrix

DESCRIPTION :

The entries of t are sine inverse of the corresponding entries of x. Definition domain is \([-1, 1]\). It takes his values in sets

\[ ] - \pi /2, \pi /2 [x] - \infty + \infty [ \]

\[ [-\pi /2] \times [0, +\infty] \text{ and } [\pi /2] \times [0, \infty] \text{ (real x imag)} \]

EXAMPLE :

A=[1,2;3,4]  
\text{sin}(\text{asin}(A))

SEE ALSO:  \text{sin} 210,  \text{sinm} 211,  \text{asinm} 167

1.3.11  asinh  __________________________________________________________________ hyperbolic sine inverse

CALLING SEQUENCE :

\[ [t]=\text{asinh}(x) \]

PARAMETERS :

x : real or complex vector/matrix  
t : real or complex vector/matrix

DESCRIPTION :

The entries of t are the hyperbolic sine inverse of the corresponding entries of x. Definition domain is \([-1,i]\). It takes his values in sets

\[ ] - \infty + \infty [x] - \pi /2, \pi /2 [ \]

\[ [-\infty, 0] \times [-\pi /2] \text{ and } [0, \infty] \times [\pi /2] \text{ (real x imag)} \]

EXAMPLE :

A=[1,2;2,3]  
\text{sinh}(\text{asinh}(A))
1.3.12  **asinhm**  ________________________ matrix hyperbolic inverse sine

**CALLING SEQUENCE:**

\[ t = \text{asinhm}(x) \]

**PARAMETERS:**

\( x, t \) : real or complex square matrix

**DESCRIPTION:**

\( \text{asinhm} \) is the matrix hyperbolic inverse sine of the matrix \( x \). Uses the formula \( t = \logm(x + \sqrtm(x \times x + \eye())) \). Results may be not reliable for non-symmetric matrix.

**EXAMPLE :**

\[
A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \\
\sinhm(\text{asinhm}(A))
\]

**SEE ALSO:**  \( \text{asinh 166, logm 196, sqrtm 219} \)

1.3.13  **asinm**  ________________________ matrix wise sine inverse

**CALLING SEQUENCE:**

\[ t = \text{asinm}(x) \]

**PARAMETERS:**

\( x, t \) : real or complex square matrix

**DESCRIPTION:**

\( t \) are sine inverse of the \( x \) matrix. Diagonalization method is used. For non symmetric matrices result may be inaccurate.

**EXAMPLE :**

\[
A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \\
\sinm(\text{asinm}(A)) \\
\text{asinm}(A) + \eye{} \times \logm(\eye{} + \sqrtm(\eye{} - A \times A))
\]

**SEE ALSO:**  \( \text{asin 166, sinm 211} \)

1.3.14  **atan**  ________________________ 2-quadrant and 4-quadrant inverse tangent

**CALLING SEQUENCE:**

\[ \phi = \text{atan}(x) \]

\[ \phi = \text{atan}(y, x) \]

**PARAMETERS:**

\( x \) : real or complex scalar, vector or matrix
\( \phi \) : real or complex scalar, vector or matrix
\( x, y \) : real scalars, vectors or matrices of the same size
atanhm Scilab Function

phi : real scalar, vector or matrix

DESCRIPTION :
The first form computes the 2-quadrant inverse tangent, which is the inverse of tan(phi). For real x, phi is in the interval (-pi/2, pi/2). For complex x, atan has two singular, branching points +%i,-%i and the chosen branch cuts are the two imaginary half-straight lines [i, i*oo) and (-i*oo, -i].

The second form computes the 4-quadrant arctangent (‘atan2’ in Fortran), this is, it returns the argument (angle) of the complex number x+i*y. The range of atan(y, x) is (-pi, pi].

For real arguments, both forms yield identical values if x>0.

In case of vector or matrix arguments, the evaluation is done element-wise, so that phi is a vector or matrix of the same size with phi(i,j)=atan(x(i,j)) or phi(i,j)=tan(y(i,j),x(i,j)).

EXAMPLES :

// examples with the second form
x=[1,%i,-1,%i]
phasex=atan(imag(x),real(x))
atan(0,-1)
atan(-%eps,-1)

// branch cuts, [+i, i*oo) and (-i*oo, -i]
atan(-%eps + 2*%i)
atan(+%eps + 2*%i)
atan(-%eps - 2*%i)
atan(+%eps - 2*%i)

// values at the branching points
ieee(2)
atan(%i)
atan(-%i)

SEE ALSO: tan 225, ieee 50

1.3.15 atanh ____________________________ hyperbolic tangent inverse

CALLING SEQUENCE :
t=atanh(x)

PARAMETERS :
x : real or complex vector/matrix
t : real or complex vector/matrix

DESCRIPTION :
The components of vector t are the hyperbolic tangent inverse of the corresponding entries of vector x. Definition domain is [-1,1[.

This function takes values in

\[-\infty, \infty[\times (-\pi/2, \pi/2]

\[-\infty, 0[\times [-\pi/2] \quad \text{and} \quad ]0, \infty[\times [\pi/2] \quad (\text{real x imag})

EXAMPLE :
x=[0,%i,-%i]
tanh(atanh(x))

Scilab Group April 1993 168
1.3.16 atanhm ______________________ matrix hyperbolic tangent inverse

CALLING SEQUENCE : 

\[ t = \text{atanhm}(x) \]

PARAMETERS :
\[ x : \text{real or complex square matrix} \]
\[ t : \text{real or complex square matrix} \]

DESCRIPTION :
\( \text{atanhm}(x) \) is the matrix hyperbolic tangent inverse of matrix \( x \). Results may be inaccurate if \( x \) is not symmetric.

EXAMPLE :
\[ A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}; \]
\[ \text{tanhm}(\text{atanhm}(A)) \]

SEE ALSO: atanh 168, tanhm 226

1.3.17 atanm ______________________ square matrix tangent inverse

CALLING SEQUENCE : 

\[ [t] = \text{atanm}(x) \]

PARAMETERS :
\[ x : \text{real or complex square matrix} \]
\[ t : \text{real or complex square matrix} \]

DESCRIPTION :
\( \text{atanm}(x) \) is the matrix arctangent of the matrix \( x \). Result may be not reliable if \( x \) is not symmetric.

EXAMPLE :
\[ \text{tanm}(\text{atanm}([1,2;3,4])) \]

SEE ALSO: atan 167

1.3.18 besseli __________ Modified I sub ALPHA Bessel functions of the first kind.

CALLING SEQUENCE :

\[ y = \text{besseli}(\alpha, x) \]
\[ y = \text{besseli}(\alpha, x, \text{ice}) \]

PARAMETERS :
\[ x : \text{real vector with non negative entries} \]
\[ \alpha : \text{real vector with non negative entries regularly spaced with increment equal to one} \]
\[ \text{ice} : \text{integer flag, with default value 1} \]
besseli(alpha,x) computes I sub ALPHA modified Bessel functions of the first kind, for real, non-negative order alpha and argument x. alpha and x may be vectors. The output is m-by-n with m = size(x,'*'), n = size(alpha,'*') whose (i,j) entry is besseli(alpha(j),x(i)).

If ice is equal to 2 exponentially scaled Bessel functions is computed

EXAMPLE:

besseli(0.5:3,1:4)
besseli(0.5:3,1:4,2)

SEE ALSO : besselj, besselk

1.3.19 besselj ________ Modified J sub ALPHA Bessel functions of the first kind.

CALLING SEQUENCE :

y = besselj(alpha,x)

PARAMETERS :

x : real vector with non negative entries
alpha : real vector with non negative entries regularly spaced with increment equal to one alpha=alpha0+(n1:n2)

DESCRIPTION :

besselj(alpha,x) computes J sub ALPHA modified Bessel functions of the first kind, for real, non-negative order alpha and argument x. alpha and x may be vectors. The output is m-by-n with m = size(x,'*'), n = size(alpha,'*') whose (i,j) entry is besselj(alpha(j),x(i)).

EXAMPLE:

besselj(0.5:3,1:4)

SEE ALSO : besseli, besselk

1.3.20 besselk ______ Modified K sub ALPHA Bessel functions of the second kind.

CALLING SEQUENCE :

y = besselk(alpha,x)
y = besselk(alpha,x,ice)

PARAMETERS :

x : real vector with non negative entries
alpha : real vector with non negative entries regularly spaced with increment equal to one alpha=alpha0+(n1:n2)

DESCRIPTION :

besselk(alpha,x) computes K sub ALPHA modified Bessel functions of the second kind, for real, non-negative order alpha and argument x. alpha and x may be vectors. The output is m-by-n with m = size(x,'*'), n = size(alpha,'*') whose (i,j) entry is besseli(alpha(j),x(i)).

EXAMPLE :

besselk(0.5:3,1:4)

SEE ALSO : besseli, besselk
DESCRIPTION:

\texttt{besselk(alpha,x)} computes \( K_{\alpha} \) modified Bessel functions of the second kind, for real, non-negative order \( \alpha \) and argument \( x \). \( \alpha \) and \( x \) may be vectors. The output is \( m \)-by-\( n \) with \( m = \text{size}(x,'*') \), \( n = \text{size}(\alpha,'*') \) whose \((i,j)\) entry is \( \text{besselk}(\alpha(j),x(i)) \).

If \( \text{ice} \) is equal to 2 exponentially scaled Bessel functions is computed.

EXAMPLE:

\begin{verbatim}
besselk(0.5:3,1:4)
besselk(0.5:3,1:4,2)
\end{verbatim}

SEE ALSO: \texttt{besselj \ 170}, \texttt{besseli \ 169}, \texttt{bessely \ 171}

1.3.21 \ \texttt{bessely} \ \ \text{Modified} \ Y_{\alpha} \text{Bessel functions of the second kind.}

CALLING SEQUENCE:

\begin{verbatim}
y = bessely(alpha,x)
\end{verbatim}

PARAMETERS:

\begin{verbatim}
x : real vector with non negative entries
alpha : real vector with non negative entries regularly spaced with increment
        equal to one alpha=alpha0+(n1:n2)
\end{verbatim}

DESCRIPTION:

\texttt{bessely(alpha,x)} computes \( Y_{\alpha} \) modified Bessel functions of the second kind, for real, non-negative order \( \alpha \) and argument \( x \). \( \alpha \) and \( x \) may be vectors. The output is \( m \)-by-\( n \) with \( m = \text{size}(x,'*') \), \( n = \text{size}(\alpha,'*') \) whose \((i,j)\) entry is \( \text{bessely}(\alpha(j),x(i)) \).

EXAMPLE:

\begin{verbatim}
bessely(0.5:3,1:4)
\end{verbatim}

SEE ALSO: \texttt{besselj \ 170}, \texttt{besseli \ 169}, \texttt{besselk \ 170}

1.3.22 \ \texttt{binomial} \ \ \text{binomial distribution}

CALLING SEQUENCE:

\begin{verbatim}
pr=binomial(p,n)
\end{verbatim}

PARAMETERS:

\begin{verbatim}
pr : vector, pr(k) = probability \( X=k \), with \( X=B(n,p) \).
p : real number
n : integer
\end{verbatim}

DESCRIPTION:

\texttt{pr=binomial(p,n)} return a binomial probability vector. \( pr(k) = \text{probability} \ (X=k), \ with \ X=B(n,p) \)
i.e the probability of \( k \) success in \( n \) independent trials, \( (p = \text{proba. of one success}) \). \( pr(k) = \binom{n}{k} p^k (1-p)^{n-k} \) with \( \binom{n}{k} = n!/(k!(n-k)!) = \text{prod}(1:n)/(\text{prod}(1:k)*\text{prod}(1:n-k)) \).

EXAMPLE:

\begin{verbatim}
pr=binomial(p,n)
\end{verbatim}
bloc2exp Scilab Function

n=10;p=0.3;plot2d3(0:n,binomial(p,n));
n=100;p=0.3;
mea=n*p;sigma=sqrt(n*p*(1-p));
x=((0:n)-mea)/sigma;
plot2d(x,sigma*binomial(p,n));
deff(’y=Gauss(x)’,’y=1/sqrt(2*%pi)*exp(-(x.*2)/2)’)
plot2d(x,Gauss(x));

SEE ALSO: cdfbin

1.3.23 bloc2exp ___________________________ block-diagram to symbolic expression

CALLING SEQUENCE:

[str]=bloc2exp(blocd)
[str,names]=bloc2exp(blocd)

PARAMETERS:

blocd : list
str : string
names : string

DESCRIPTION:

given a block-diagram representation of a linear system bloc2exp returns its symbolic evaluation. The first element of the list blocd must be the string ’blocd’. Each other element of this list (blocd(2),blocd(3),...) is itself a list of one the following types:

list(’transfer’,’name_of_linear_system’)

list(’link’,’name_of_link’,
    [number_of_upstream_box,upstream_box_port],
    [downstream_box_1,downstream_box_1_portnumber],
    [downstream_box_2,downstream_box_2_portnumber],
    ...)

The strings ’transfer’ and ’links’ are keywords which indicate the type of element in the block diagram.

Case 1 : the second parameter of the list is a character string which may refer (for a possible further evaluation) to the Scilab name of a linear system given in state-space representation (syslin list) or in transfer form (matrix of rationals).
To each transfer block is associated an integer. To each input and output of a transfer block is also associated its number, an integer (see examples)

Case 2 : the second kind of element in a block-diagram representation is a link. A link links one output of a block represented by the pair [number_of_upstream_box,upstream_box_port], to different inputs of other blocks. Each such input is represented by the pair [downstream_box_i,downstream_box_i_portnumber].
The different elements of a block-diagram can be defined in an arbitrary order.

For example
[I] S1*S2 with unit feedback.
There are 3 transfers S1 (number n_s1=2), S2 (number n_s2=3) and an adder (number n_add=4) with symbolic transfer function [’1’,’1’].
There are 4 links. The first one (named ’U’) links the input (port 0 of fictitious block -1, omitted) to port 1 of the adder. The second and third one respectively (output)port 1 of the adder to (input)port 1 of system S1, and (output)port 1 of S1 to (input)port 1 of S2. The fourth link (named ’Y’) links (output)port 1 of S2 to the output (port 0 of fictitious block -1, omitted) and to (input)port 2 of the adder.

Scilab Group
April 1993
172
bloc2ss  

Scilab Function

// Initialization
syst=list('blocd'); l=1;

// Systems
l=l+1; n_s1=l; syst(l)=list('transfer','S1'); // System 1
l=l+1; n_s2=l; syst(l)=list('transfer','S2'); // System 2
l=l+1; n_adder=l; syst(l)=list('transfer',['1','1']); // adder

// Links
// Inputs -1 --> input 1
l=l+1; syst(l)=list('link','U',[-1],[n_adder,1]);
// Internal
l=l+1; syst(l)=list('link',null,[n_adder,1],[n_s1,1]);
l=l+1; syst(l)=list('link',null,[n_s1,1],[n_s2,1]);
// Outputs // -1 -> output 1
l=l+1; syst(l)=list('link','Y',[-1],[n_adder,2]);

// Evaluation call
w=bloc2exp(syst);

The result is the character string: 
\[ w=-(s2*s1-eye())s1 \]

Note that invoked with two output arguments, 
\([\text{str}, \text{names}]=\text{blocd}(\text{syst})\) returns in \text{names} the list of symbolic names of named links. This is useful to set names to inputs and outputs.

[2] second example

// Initialization
syst=list('blocd'); l=1;

// System (2x2 blocks plant)
l=l+1; n_s=l; syst(l)=list('transfer',['P11','P12';'P21','P22']);

// Controller
l=l+1; n_k=l; syst(l)=list('transfer','k');

// Links
l=l+1; syst(l)=list('link','w',[-1],[n_s,1]);
l=l+1; syst(l)=list('link','z',[n_s,1],[n_s,1]);
l=l+1; syst(l)=list('link','u',[n_k,1],[n_s,2]);
l=l+1; syst(l)=list('link','y',[n_s,2],[n_k,1]);

// Evaluation call
w=bloc2exp(syst);

In this case the result is a formula equivalent to the usual one:
\[ P11+P12*invr(\text{eye}()-K*P22)*K*P21; \]

SEE ALSO:  bloc2ss 173

AUTHOR: S. S., F. D. (INRIA)

1.3.24 bloc2ss  ______________ block-diagram to state-space conversion

CALLING SEQUENCE:

\[[\text{s1}]=\text{bloc2ss(\text{blocd})}\]\n
Scilab Group  April 1993 173
PARAMETERS:

blocd : list
s1 : list

DESCRIPTION:
Given a block-diagram representation of a linear system, `bloc2ss` converts this representation to a state-space linear system. The first element of the list `blocd` must be the string 'blocd'. Each other element of this list is itself a list of one of the following types:

```
list('transfer','name_of_linear_system')
list('link','name_of_link',
    [number_of_upstream_box,upstream_box_port],
    [downstream_box_1,downstream_box_1_portnumber],
    [downstream_box_2,downstream_box_2_portnumber],
    ...)
```

The strings 'transfer' and 'link' are keywords which indicate the type of element in the block diagram.

Case 1: the second parameter of the list is a character string which may refer (for a possible further evaluation) to the Scilab name of a linear system given in state-space representation (`syslin list`) or in transfer form (matrix of rationals). To each transfer block is associated an integer. To each input and output of a transfer block is also associated its number, an integer (see examples)

Case 2: the second kind of element in a block-diagram representation is a link. A link links one output of a block represented by the pair `[number_of_upstream_box,upstream_box_port]`, to different inputs of other blocks. Each such input is represented by the pair `[downstream_box_i,downstream_box_i_portnumber]`.

The different elements of a block-diagram can be defined in an arbitrary order.

For example

```
[1] S1*S2 with unit feedback.
There are 3 transfers S1 (number n_s1=2), S2 (number n_s2=3) and an adder (number n_add=4) with symbolic transfer function ['1','1'].
There are 4 links. The first one (named 'U') links the input (port 0 of fictitious block -1, omitted) to port 1 of the adder. The second and third one link respectively (output)port 1 of the adder to (input)port 1 of system S1, and (output)port 1 of S1 to (input)port 1 of S2. The fourth link (named 'Y') links (output)port 1 of S2 to the output (port 0 of fictitious block -1, omitted) and to (input)port 2 of the adder.
```

```
//Initialization
syst=list('blocd'); l=1;
//
//Systems
l=l+1;n_s1=l;syst(l)=list('transfer','S1'); //System 1
l=l+1;n_s2=l;syst(l)=list('transfer','S2'); //System 2
l=l+1;n_adder=l;syst(l)=list('transfer',['1','1']); //adder
//
//Links
// Inputs -1 --> input 1
l=l+1;syst(l)=list('link','U1',[-1],[n_adder,1]); // Internal
l=l+1;syst(l)=list('link',' ',[n_adder,1],[n_s1,1]);
// Outputs \-1 --> output 1
l=l+1;syst(l)=list('link','Y',[-1],[n_s2,1],[n_adder,2]);
```

With `s=poly(0,'s');S1=1/(s+1);S2=1/s;` the result of the evaluation call `sl=bloc2ss(syst)` is a state-space representation for \(1/(s^2+s-1)\).
//Initialization
syst=list('blocd'); l=1;
//
//System (2x2 blocks plant)
S = list('transfer',[P11', P12'; P21', P22']);
//
//Controller
l=l+1;n_k=l;syst(l)=list('transfer','k');
//
//Links
l=l+1;syst(l)=list('link','w',[-1],[n_s,1]);
l=l+1;syst(l)=list('link','z',[n_s,1],[-1]);
l=l+1;syst(l)=list('link','u',[n_k,1],[n_s,2]);
l=l+1;syst(l)=list('link','y',[n_s,2],[n_k,1]);

With

P=syslin('c',A,B,C,D);
P11=P(1,1);
P12=P(1,2);
P21=P(2,1);
P22=P(2,2);
K=syslin('c',Ak,Bk,Ck,Dk);

bloc2exp(syst) returns the evaluation the lft of P and K.

SEE ALSO: bloc2exp 172

AUTHOR: S. S., F. D. (INRIA)

1.3.25 calerf computes error functions.

CALLING SEQUENCE:

\[ y = \text{calerf}(x, \text{flag}) \]

PARAMETERS:

\( x \): real vector
\( \text{flag} \): integer indicator
\( y \): real vector (of same size)

DESCRIPTION:

calerf(x, 0) computes the error function:

\[ y = \frac{2}{\sqrt{\pi}} \int_0^x e^{\frac{-f^2}{2}} df \]

calerf(x, 1) computes the complementary error function:

\[ y = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{\frac{-f^2}{2}} df \]

\[ y = 1 - \text{erf}(x) \]
calerf(x,2) computes the scaled complementary error function:

\[
y = \exp(x^2) \text{erfc}(x) \frac{1}{x \sqrt{\pi}} \text{for large } x
\]

EXAMPLE:

deff('y=f(t)','y=exp(-t^2)');
calerf(1,0)
2/sqrt(%pi)*intg(0,1,f)

SEE ALSO: erf 182, erfc 183, calerf 175

1.3.26 ceil -------------------------------------------------------- rounding up

CALLING SEQUENCE:

[y]=ceil(x)

PARAMETERS:

x : a real matrix
y : integer matrix

DESCRIPTION:

ceil(x) returns an integer matrix made of rounded up elements

EXAMPLE:

ceil([1.9 -2.5])-[2,-2]
ceil(-%inf)
x=rand()'*10^20;ceil(x)-x

SEE ALSO: round 209, floor 185, int 188

1.3.27 cmb_lin ----------------------------- symbolic linear combination

CALLING SEQUENCE:

[x]=cmb_lin(alfa,x,beta,y)

DESCRIPTION:

Evaluates \( alfa \times x - beta \times y \). alfa, beta, x, y are character strings. (low-level routine)

SEE ALSO: mulf 203, addf 164

1.3.28 conj ----------------------------------------- conjugate

CALLING SEQUENCE:

[y]=conj(x)

PARAMETERS:

x, y : real or complex matrix.
**DESCRIPTION:**

conj(x) is the complex conjugate of x.

**EXAMPLE:**

```plaintext
x=[1+%i,-%i;%i,2*%i];
conj(x)
x'-conj(x) // x' is conjugate transpose
```

---

1.3.29 **cos**

### cosine function

**CALLING SEQUENCE:**

```plaintext
[y]=cos(x)
```

**PARAMETERS:**

- x : real or complex vector/matrix

**DESCRIPTION:**

For a vector or a matrix, cos(x) is the cosine of its elements. For matrix cosine use cosm(X) function.

**EXAMPLE:**

```plaintext
x=[0,1,%i]
acos(cos(x))
```

**SEE ALSO:** cos 178

---

1.3.30 **cosh**

### hyperbolic cosine

**CALLING SEQUENCE:**

```plaintext
[t]=cosh(x)
```

**PARAMETERS:**

- x, t : real or complex vectors/matrices

**DESCRIPTION:**

The elements of t are the hyperbolic cosine of the corresponding entries of vector x.

**EXAMPLE:**

```plaintext
x=[0,1,%i]
acosh(cosh(x))
```

**SEE ALSO:** cos 177, acosh 162
1.3.31  coshm  ------------------------------------- matrix hyperbolic cosine

CALLING SEQUENCE :

\[ t = \coshm(x) \]

PARAMETERS :

\(x, t\) : real or complex square matrix

DESCRIPTION :

\(\coshm\) is the matrix hyperbolic cosine of the matrix \(x\). \(t = (\expm(x) + \expm(-x))/2\). Result may be inaccurate for nonsymmetric matrix.

EXAMPLE :

\[ A = [1, 2; 2, 4] \]
\[ \text{acoshm(coshm(A))} \]

SEE ALSO:  \(\cosh\) 177,  \(\expm\) 509

1.3.32  cosm  ------------------------------------- matrix cosine function

CALLING SEQUENCE :

\[ t = \cosm(x) \]

PARAMETERS :

\(x\) : real or complex square matrix

DESCRIPTION :

\(\cosm(x)\) is the matrix cosine of the \(x\) matrix. \(t = 0.5*(\expm(\%i*x) + \expm(-\%i*x))\).

EXAMPLE :

\[ A = [1, 2; 3, 4] \]
\[ \cosm(A) - 0.5*(\expm(\%i*A) + \expm(-\%i*A)) \]

SEE ALSO:  \(\cos\) 177,  \(\expm\) 509

1.3.33  cotg  ------------------------------------- cotangent

CALLING SEQUENCE :

\([t] = \cotg(x)\)

PARAMETERS :

\(x, t\) : real or complex vectors/matrices

DESCRIPTION :

The elements of \(t\) are the cotangents of the corresponding entries of \(x\). \(t = \cos(x) ./ \sin(x)\)

EXAMPLE :

\[ x = [1, \%i] ; \]
\[ \cotg(x) - \cos(x) ./ \sin(x) \]

SEE ALSO:  \(\tan\) 225
1.3.34  **coth**  hyperbolic cotangent

**CALLING SEQUENCE :**

\[ t = \text{coth}(x) \]

**DESCRIPTION :**

The elements of vector \( t \) are the hyperbolic cotangent of elements of the vector \( x \).

**EXAMPLE :**

\[
\begin{align*}
  x &= \begin{bmatrix} 1, 2*%i \end{bmatrix} \\
  t &= \exp(x) \\
  (t - \text{ones}(x) ./ t) .\backslash (t + \text{ones}(x) ./ t) \\
  \text{coth}(x)
\end{align*}
\]

**SEE ALSO :**  cotg 178

1.3.35  **cothm**  matrix hyperbolic cotangent

**CALLING SEQUENCE :**

\[ t = \text{cothm}(x) \]

**DESCRIPTION :**

\( \text{cothm}(x) \) is the matrix hyperbolic cotangent of the square matrix \( x \).

**EXAMPLE :**

\[
\begin{align*}
  A &= \begin{bmatrix} 1, 2; 3, 4 \end{bmatrix} \\
  \text{cothm}(A)
\end{align*}
\]

**SEE ALSO :**  coth 179

1.3.36  **cumprod**  cumulative product

**CALLING SEQUENCE :**

\[
\begin{align*}
  y &= \text{cumprod}(x) \\
  y &= \text{cumprod}(x, 'r') \text{ or } y = \text{cumprod}(x, 1) \\
  y &= \text{cumprod}(x, 'c') \text{ or } y = \text{cumprod}(x, 2)
\end{align*}
\]

**PARAMETERS :**

\( x \) : vector or matrix (real or complex)

\( y \) : vector or matrix (real or complex)

**DESCRIPTION :**

For a vector or a matrix \( x \), \( y = \text{cumprod}(x) \) returns in \( y \) the cumulative product of all the entries of \( x \) taken columnwise.

\( y = \text{cumprod}(x, 'c') \) (or, equivalently, \( y = \text{cumprod}(x, 2) \)) returns in \( y \) the cumulative elementwise product of the columns of \( x \): \( y(i,:) = \text{cumprod}(x(i,:)) \).

\( y = \text{cumprod}(x, 'r') \) (or, equivalently, \( y = \text{cumprod}(x, 2) \)) returns in \( y \) the cumulative elementwise product of the rows of \( x \): \( y(:,i) = \text{cumprod}(x(:,i)) \).

**EXAMPLE :**
A=[1,2;3,4];
cumprod(A)
cumprod(A,'r')
cumprod(A,'c')
rand('seed',0);
a=rand(3,4);
[m,n]=size(a);
w=zeros(a);
w(1,:)=a(1,:);
for k=2:m;w(k,:)=w(k-1,:).*a(k,:);end;w-cumprod(a,'r')

SEE ALSO: cumprod 179, sum 222

### 1.3.37 cumsum ----------------------------------------------- cumulative sum

**CALLING SEQUENCE:**

- `y=cumsum(x)`
- `y=cumsum(x,'r')` or `y=cumsum(x,1)`
- `y=cumsum(x,'c')` or `y=cumsum(x,2)`

**PARAMETERS:**

- `x`: vector or matrix (real or complex)
- `y`: vector or matrix (real or complex)

**DESCRIPTION:**

For a vector or a matrix `x`, `y=cumsum(x)` returns in `y` the cumulative sum of all the entries of `x` taken columnwise.

- `y=cumsum(x,'c')` (or, equivalently, `y=cumsum(x,2)`) returns in `y` the cumulative sum of the columns of `x`: `y(i,:)=cumsum(x(i,:))`
- `y=cumsum(x,'r')` (or, equivalently, `y=cumsum(x,1)`) returns in `y` the cumulative sum of the rows of `x`: `y(:,i)=cumsum(x(:,i))`

**EXAMPLE:**

A=[1,2;3,4];
cumsum(A)
cumsum(A,'r')
cumsum(A,'c')
a=rand(3,4)+%i;
[m,n]=size(a);
w=zeros(a);
w(1,:)=a(1,:);
for k=2:m;w(k,:)=w(k-1,:)+a(k,:);end;w-cumsum(a,'r')

SEE ALSO: cumprod 179, sum 222

### 1.3.38 delip --------------------------------------------- elliptic integral

**CALLING SEQUENCE:**

- `[r]=delip(x,ck)`

**PARAMETERS:**

Scilab Group

April 1993

180
dlgamma Scilab Function

**DESCRIPTION:**
returns the value of the elliptic integral with parameter \( ck \):

\[
 r = \int_0^x \frac{1}{\sqrt{(1-t^2)(1-c^2t^2)}} dt 
\]

\( x \) is real and positive. When called with \( x \) a real vector \( r \) is evaluated for each entry of \( x \).

**EXAMPLE:**

\( ck=0.5; \)
\( \text{delip}([1,2],ck) \)
\( \text{deff('y=f(t)','y=1/sqrt((1-t^2)*(1-ck^2*t^2)))') } \)
\( \text{intg}(0,1,f) \quad \text{//OK since real solution!} \)

**SEE ALSO:** amell 165, %asn 445, %sn 446

1.3.39  \text{diag} \quad \text{diagonal including or extracting}

**CALLING SEQUENCE:**
\[ [y]=\text{diag}(vm, [k]) \]

**PARAMETERS:**
- \( vm \): vector or matrix (full or sparse storage)
- \( k \): integer (default value 0)
- \( y \): vector or matrix

**DESCRIPTION:**
- For a vector or column \( n \)-vector \( \text{diag}(vm) \) returns a diagonal matrix with entries of \( vm \) along the main diagonal.
- \( \text{diag}(vm,k) \) is a \((n+\text{abs}(k)) \times (n+\text{abs}(k))\) matrix with the entries of \( vm \) along the \( k \)th diagonal.
- \( k=0 \) is the main diagonal \( k>0 \) is for upper diagonals and \( k<0 \) for lower diagonals.
- For a matrix \( vm \), \( \text{diag}(vm,k) \) is the column vector made of entries of the \( k \)th diagonal of \( vm \). \( \text{diag}(vm) \) is the main diagonal of \( vm \). \( \text{diag}(\text{diag}(x)) \) is a diagonal matrix.
- If \( vm \) is a sparse matrix \( \text{diag}(vm,k) \) returns a sparse matrix.
- To construct a diagonal linear system, use \text{sysdiag}.
- Note that \( \text{eye}(A) \ast A \) returns a diagonal matrix made with the diagonal entries of \( A \). This is valid for any matrix (constant, polynomial, rational, state-space linear system, ...).

**EXAMPLE:**

\( \text{diag}([1,2]) \)
\( A=[1,2;3,4]; \)
\( \text{diag}(A) \quad \text{// main diagonal} \)
\( \text{diag}(A,1) \)
\( \text{diag}(	ext{sparse}(1:10)) \quad \text{// sparse diagonal matrix} \)

// form a tridiagonal matrix of size 2*m+1
\( m=5; \text{diag}(-m:m) + \text{diag}(\text{ones}(2*m,1),1) + \text{diag}(\text{ones}(2*m,1),-1) \)

**SEE ALSO:** sysdiag 224, sparse 214
### 1.3.40  

dlgamma ______________________________ derivative of gammaln function.

**CALLING SEQUENCE:**

\[ y = \text{dlgamma}(x) \]

**PARAMETERS:**

- \( x \): real vector
- \( y \): real vector with same size.

**DESCRIPTION:**

dlgamma \((x)\) evaluates the derivative of gammaln function at all the elements of \(x\). \(x\) must be real.

**EXAMPLE:**

dlgamma(0.5)

**SEE ALSO:**  
gamma 186, gammaln 186

### 1.3.41  

double ______ conversion from integer to double precision representation

**CALLING SEQUENCE:**

\[
\begin{align*}
y &= \text{double}(X) \\
y &= \text{int16}(X) \\
y &= \text{int32}(X) \\
y &= \text{uint8}(X) \\
y &= \text{uint16}(X) \\
y &= \text{uint32}(X)
\end{align*}
\]

**PARAMETERS:**

- \( X \): matrix of floats or integers
- \( y \): matrix of floats

**DESCRIPTION:**

converts data stored using one, two or four bytes integers into double precision floating point representation. If \(X\) entries are already double precision floats, nothing is done.

**EXAMPLES:**

\[
\begin{align*}
x &= \text{int8}([0 \ 12 \ 140]) \\
\text{double}(x)
\end{align*}
\]

**SEE ALSO:**  
int8 188, inttype 54, type 74

### 1.3.42  

erf ________________________________ The error function.

**CALLING SEQUENCE:**

\[ y = \text{erf}(x) \]

**PARAMETERS:**
x : real vector
y : real vector (of same size)

DESCRIPTION :
erf computes the error function:
\[ y = 2/\sqrt{\pi} \int_0^\infty e^{-t^2} \, dt \]

EXAMPLE :
deff('y=f(t)','y=exp(-t^2)');
erf(0.5)-2/sqrt(%pi)*intg(0,0.5,f)

SEE ALSO : erf, erfc, erfcx, calerf

1.3.43 erf The complementary error function.

CALING SEQUENCE :
y = erf(x)

PARAMETERS :
x : real vector
y : real vector (of same size)

DESCRIPTION :
erfc computes the complementary error function:
\[ y = 2/\sqrt{\pi} \int_0^\infty e^{-t^2} \, dt \]
\[ y = 1 - erf(x) \]

EXAMPLE :
erf([0.5,0.2])+erfc([0.5,0.2])

SEE ALSO : erf, erfc, erfcx, calerf

1.3.44 erfcx The scaled complementary error function.

CALING SEQUENCE :
y = erfcx(x)

PARAMETERS :
x : real vector
y : real vector (of same size)

DESCRIPTION :
erfcx computes the scaled complementary error function:
\[ y = \exp(x^2)erfc(x) (1/\sqrt{\pi})1/x \text{ for larger } x \]

SEE ALSO : erf, erfc, erfcx, calerf
1.3.45  eval  evaluation of a matrix of strings

CALLING SEQUENCE :

\[ H = \text{eval}(Z) \]

DESCRIPTION :
returns the evaluation of the matrix of character strings \( Z \).

EXAMPLE :

```plaintext
a=1; b=2; Z=[’a’,’sin(b)’] ; eval(Z)  //returns the matrix [1,0.909];
```

SEE ALSO:  evstr 35, execstr 37

1.3.46  eye  identity matrix

CALLING SEQUENCE :

\[
X = \text{eye}(m,n) \\
X = \text{eye}(A) \\
X = \text{eye}() 
\]

PARAMETERS :

\( A, X \) : matrices or syslin lists  \\
\( m, n \) : integers

DESCRIPTION :
according to its arguments defines an \( m \times n \) matrix with 1 along the main diagonal or an identity matrix of the same dimension as \( A \).  

Caution: \( \text{eye}(10) \) is interpreted as \( \text{eye}(A) \) with \( A=10 \) i.e. 1. (It is NOT a ten by ten identity matrix!). If \( A \) is a linear system represented by a syslin list, \( \text{eye}(A) \) returns an \( \text{eye} \) matrix of appropriate dimension: (number of outputs x number of inputs). \( \text{eye}() \) produces a identity matrix with undefined dimensions. Dimensions will be defined when this identity matrix is added to a matrix with fixed dimensions.

EXAMPLES :

```plaintext
eye(2,3)  \\
A=rand(2,3);eye(A)  \\
s=poly(0,’s’);A=[s,1;s,s+1];eye(A)  \\
A=[1/s,1;s,2];eye(A);  \\
A=ssrand(2,2,3);eye(A)  \\
[1 2 ;3 4]+2*eye()
```

SEE ALSO:  ones 204, zeros 231

1.3.47  fix  rounding towards zero

CALLING SEQUENCE :

\[ y = \text{fix}(x) \]

PARAMETERS :

\( x \) : scalar or vector of doubles
**Scilab Function**

**x**: a real matrix

**y**: integer matrix

**DESCRIPTION**:

fix(x) returns an integer matrix made of nearest rounded integers toward zero, i.e., \( y = \text{sign}(x) \times \text{floor}(\text{abs}(x)) \). Same as int.

**SEE ALSO**: round 209, floor 185, ceil 176

1.3.48 floor __________________________ rounding down

**CALLING SEQUENCE**:

\[ [y]=\text{floor}(x) \]

**PARAMETERS**:

\( x \): a real matrix

\( y \): integer matrix

**DESCRIPTION**:

floor(x) returns an integer matrix made of nearest rounded down integers.

**EXAMPLE**:

floor([1.9, -2.5]) - [1, -3]
floor(-\infty)
x = rand() * 10^20; floor(x) - x

**SEE ALSO**: round 209, fix 184, ceil 176

1.3.49 frexp ___ dissect floating-point numbers into base 2 exponent and mantissa

**CALLING SEQUENCE**:

\[ [f,e]=\text{frexp}(x) \]

**PARAMETERS**:

\( x \): real vector or matrix

\( f \): array of real values, usually in the range \( 0.5 \leq \text{abs}(f) < 1 \).

\( e \): array of integers that satisfy the equation: \( x = f \times 2^e \)

**DESCRIPTION**:

This function corresponds to the ANSI C function frexp(). Any zeros in \( x \) produce \( f=0 \) and \( e=0 \).

**EXAMPLE**:

\[ [f,e]=\text{frexp}([1, \pi, -3, \text{eps}]) \]

**SEE ALSO**: log 195, hat 47, ieee 50, log2 196
1.3.50  full  __________________________ sparse to full matrix conversion

CALLING SEQUENCE :

X=full(sp)

PARAMETERS :

sp : real or complex sparse (or full) matrix
X : full matrix

DESCRIPTION :

X=full(sp) converts the sparse matrix sp into its full representation. (If sp is already full then X equals sp).

EXAMPLE :

sp=sparse([1,2;5,4;3,1],[1,2,3]);
A=full(sp)

SEE ALSO:  sparse 214, sprand 218, speye 216

1.3.51  gamma  __________________________ The gamma function.

CALLING SEQUENCE :

y = gamma(x)

PARAMETERS :

x : real vector
y : real vector with same size.

DESCRIPTION :

gamma(x) evaluates the gamma function at all the elements of x. x must be real.

\[ y = \int_0^\infty t^{x-1}e^{-t}dt \]

 gamma(n+1) = n!

EXAMPLE :

gamma(0.5)
gamma(6)-prod(1:5)

SEE ALSO:  gammaln 186, dlngamma 182

1.3.52  gammaln  __________________________ The logarithm of gamma function.

CALLING SEQUENCE :

y = gammaln(x)

PARAMETERS :

Scilab Group  September 1997
x  : real vector
y  : real vector with same size.

DESCRIPTION :
\texttt{gammaln(x)} evaluates the logarithm of gamma function at all the elements of \( x \), avoiding underflow and overflow. \( x \) must be real.

EXAMPLE :

\texttt{gammaln(0.5)}

SEE ALSO : \texttt{gamma} 186, \texttt{dlgamma} 182

1.3.53 \hspace{1em} \texttt{gsort} \hspace{1em} \begin{tabular}{c}
\begin{tabular}{c}
\end{tabular}
\end{tabular} \hspace{1em} \begin{tabular}{c}
\begin{tabular}{c}
\end{tabular}
\end{tabular} \hspace{1em} decreasing order sorting

CALLING SEQUENCE :

\begin{verbatim}
[s, [k]]=gsort(v )
[s, [k]]=gsort(v,flag1)
[s, [k]]=gsort(v,flag1,flag2)
\end{verbatim}

PARAMETERS :

\begin{itemize}
  \item \( v, s \) : real vector/matrix; character string vector/matrix
  \item \( \text{flag1} \) : a string \('r','c','g','lr'\) and \('lc'\).
  \item \( \text{flag2} \) : a string \('i'\) for increasing and \('d'\) for decreasing order.
  \item \( \text{k} \) : vector or matrix of integers
\end{itemize}

DESCRIPTION :

\texttt{gsort} is similar to \texttt{sort} with additional properties. The third argument can be used to chose between increasing or decreasing order. The second argument can be used for lexical orders.

\texttt{[s,k]=gsort(a,'g')} and \texttt{[s,k]=gsort(a,'g','d')} are the same as \texttt{[s,k]=gsort(a)}.

\texttt{[s,k]=gsort(a,'g','i')} performs the same operation but in increasing order.

\texttt{[s,k]=gsort(a,'lr')} sort the rows of the matrix \texttt{int(a)} (if \( a \) is a real matrix) or \( a \) (if \( a \) is a character string matrix) in lexical decreasing order. \( s \) is obtained by a permutation of the rows of matrix \texttt{int(a)} (or \( a \)) given by the column vector \( k \) in such a way that the rows of \( s \) verify \( s(i,:)/s(j,:)) if i<j. \texttt{[s,k]=gsort(a,'lr','i')} performs the same operation for increasing lexical order.

\texttt{[s,k]=gsort(a,'lc')} sort the columns of the matrix \texttt{int(a)} (if \( a \) is a real matrix) or \( a \) (if \( a \) is a character string matrix) in lexical decreasing order. \( s \) is obtained by a permutation of the columns of matrix \texttt{int(a)} (or \( a \)) given by the row vector \( k \) in such a way that the columns of \( s \) verify \( s(:,i)/s(:,j)) if i<j. \texttt{[s,k]=gsort(a,'lc','i')} performs the same operation for increasing lexical order.

EXAMPLE :

\begin{verbatim}
alr=[1,2,2; 1,2,1; 1,1,2; 1,1,1];
[alr1,k]=gsort(alr,'lr','i')
[alr1,k]=gsort(alr,'lc','i')
\end{verbatim}

SEE ALSO : \texttt{find} 41
### 1.3.54 `imag` imaginary part

**Calling Sequence:**

```
[y] = imag(x)
```

**Parameters:**
- `x`: real or complex vector or matrix.
- `y`: real vector or matrix.

**Description:**
- `imag(x)` is the imaginary part of `x`. (See `%i` to enter complex numbers).

**See Also:** `real` 208

### 1.3.55 `int` integer part

**Calling Sequence:**

```
[y] = int(X)
```

**Parameters:**
- `X`: real matrix
- `y`: integer matrix

**Description:**
- `int(X)` returns the integer part of the real matrix `X`. Same as `fix`.

**See Also:** `round` 209, `floor` 185, `ceil` 176

### 1.3.56 `int8` conversion to one byte integer representation

- `int16` - conversion to 2 bytes integer representation
- `int32` - conversion to 4 bytes integer representation
- `uint8` - conversion to one byte unsigned integer representation
- `uint16` - conversion to 2 bytes unsigned integer representation
- `uint32` - conversion to 4 bytes unsigned integer representation

**Calling Sequence:**

```
y = int8(X)
y = int16(X)
y = int32(X)
y = uint8(X)
y = uint16(X)
y = uint32(X)
```

**Parameters:**
- `X`: matrix of floats or integers
- `y`: matrix of integers coded on one, two or four bytes.

**Description:**
- Converts and stores data two one, two or four bytes integers. These data types are specially useful to store big objects such as images, long signals,...
y=int8(X) : return numbers in the range [-128,127]
y=uint8(X) : return numbers in the range [0,255]
y=int16(X) : return numbers in the range [-32768,32767]
y=uint16(X) : return numbers in the range [0, 65535]
y=int32(X) : return numbers in the range [-2147483648,2147483647]
y=uint32(X) : return numbers in the range [0, 4294967295]

EXAMPLE :
int8([1 -120 127 312])
uint8([1 -120 127 312])
x=int32(-200:100:400)
int8(x)

SEE ALSO: double 182 , inttype 54 , iconvert 49

1.3.57 integrate integration by quadrature

CALLING SEQUENCE :
[x]=integrate(expr,v,x0,x1 [,ea [,er]])

PARAMETERS :
expr : external Scilab
v : string (integration variable)
x0,x1 : real numbers (bounds of integration)
ea,er : real numbers (absolute error bound) Default value: 0
er : real number, (relative error bound) Default value: 1.d-8

DESCRIPTION :
computes : 
\[ x = \int_{x_0}^{x_1} f(v)dv \]

The evaluation hopefully satisfies following claim for accuracy: \[ \text{abs}(I-x) \leq \max(\text{ea,er} \times \text{abs}(I)) \]
where I stands for the exact value of the integral.

EXAMPLE :
integrate('sin(x)','x',0,%pi)
integrate(['if x==0 then 1,','else sin(x)/x,end'],'x',0,%pi)

SEE ALSO: intg 425 , inttrap 192 , intsplin 191 , ode 431

1.3.58 interp interpolation

CALLING SEQUENCE :
[f0 [,f1 [,f2 [,f3]]]]=interp(xd,x,f,d)

PARAMETERS :
xd : real vector

Scilab Group April 1993 189
x, f, d : real vectors from spline
fi : vectors (derivatives)

DESCRIPTION:
given three vectors (x, f, d) defining a spline function (see splin) with fi = S’(xi), di = S’’(xi)
this function evaluates S (resp. S’, S’’, S’’) at xd(i).

x : vector of xi (x(1) < x(2) < ...)
f : vector of S(xi)
d : vector of S’(xi)
f0 : vector [S(xd(1)), S(xd(2)), S(xd(3)), ...]
f1 2 3 : vector of first, second, third derivative of S at xd=[xd(1), xd(2), ...] i.e.
f1 = [S’(xd(1)), S’(xd(2)), ...]
f2 = [S’’(xd(1)), S’’(xd(2)), ...]

SEE ALSO: splin 217, smooth 212, interpln 190

1.3.59  interpln ______________________________ linear interpolation

CALLING SEQUENCE:
[y]=interpln(xyd,x)

PARAMETERS:
xyd : 2 row matrix (xy coordinates of points)
x : vector (abscissae)
y : vector (y-axis values)

DESCRIPTION:
given xyd a set of points in the xy-plane which increasing abscissae and x a set of abscissae, this function computes y the corresponding y-axis values by linear interpolation.

EXAMPLE:
x=[1 10 20 30 40];
y=[1 30 -10 20 40];
plot2d(x',y',[-3],"011","",[-10,-40,50,50]);
yi=interpln([x;y],-4:45);
plot2d((-4:45)’,yi’,[3],"000");

SEE ALSO: splin 217, interp 189, smooth 212

1.3.60  intersect __________ returns the vector of common values of two vectors

CALLING SEQUENCE:
[v, [ka,kb]]=intersect(a,b)

PARAMETERS:
a : vector of real numbers or strings
b : vector of real numbers or strings
**intsplin**

**Scilab Function**

\[ v : \text{row vector of real numbers or strings} \]
\[ ka : \text{row vector of integers} \]
\[ kb : \text{row vector of integers} \]

**DESCRIPTION:**
\[ \text{intersect}(a,b) \] returns a sorted row vector of common values of two vectors of \( a \) and \( b \).
\[ [v,ka,kb]=\text{intersect}(a,b) \] also returns index vectors \( ka \) and \( kb \) such that \( v=a(ka) \) and \( v=b(kb) \).

**EXAMPLE:**

\[ \begin{align*}
A&=\text{round}(5*\text{rand}(10,1)); \\
B&=\text{round}(5*\text{rand}(7,1)); \\
\text{intersect}(A,B) \\
[N,ka,kb]=\text{intersect}(A,B) \\
\text{intersect('a'+\text{string}(A),'a'+\text{string}(B))}
\end{align*} \]

**SEE ALSO:** unique 230, sort 213, union 230

### 1.3.61 intsplin ______ integration of experimental data by spline interpolation

**CALLING SEQUENCE:**

\[ v = \text{intsplin}([x,] s) \]

**PARAMETERS:**

\[ x : \text{vector of increasing x coordinate data. Default value is } 1:\text{size}(y,'*') \]
\[ s : \text{vector of y coordinate data} \]
\[ v : \text{value of the integral} \]

**DESCRIPTION:**

computes:

\[ v = \int_{x_0}^{x_1} f(x)dx \]

Where \( f \) is a function described by a set of experimental value:

\[ s(i) = f(x(i)) \]

and

\[ x_0 = x(1), x_1 = x(n) \]

Between mesh points function is interpolated using spline’s.

**EXAMPLE:**

\[ \begin{align*}
t&=0:0.1:6\pi \\
\text{intsplin}(t,\sin(t))
\end{align*} \]

**SEE ALSO:** intg 425, integrate 189, inttrap 192, splin 217

Scilab Group

April 1993
1.3.62  **inttrap** integration of experimental data by trapezoidal interpolation

**CALLING SEQUENCE :**

\[ v = \text{inttrap}([\mathbf{x},] \mathbf{s}) \]

**PARAMETERS :**

- **\( \mathbf{x} \)**: vector of increasing x coordinate data. Default value is \( 1: \text{size}(\mathbf{y},'\ast') \)
- **\( \mathbf{s} \)**: vector of y coordinate data
- **\( \mathbf{v} \)**: value of the integral

**DESCRIPTION :**

computes:

\[
\frac{a}{b} \int_{c}^{d} f(x)dx
\]

Where \( f \) is a function described by a set of experimental value:

\[ s(i) = f(x(i)) \]

and

\[ x_0 = x(1), x_1 = x(n) \]

Between mesh points function is interpolated linearly.

**EXAMPLE :**

```scilab
t=0:0.1:%pi
inttrap(t,sin(t))
```

**SEE ALSO :**  intg 425, intc 425, intl 426, integrate 189, intsplin 191, splin 217

1.3.63  **isdef** check variable existence

**CALLING SEQUENCE :**

\[ \text{isdef(name [,where])} \]

**PARAMETERS :**

- **name**: a character string
- **where**: an optional character string with default value ‘all’

**DESCRIPTION :**

\( \text{isdef(name)} \) returns %T if the variable ‘var-name’ exists and %F otherwise.

\( \text{isdef(name,'local')} \) returns %T if the variable ‘var-name’ exists in the local environment of the current function and %F otherwise.

**EXAMPLE :**

```scilab
A=1;
isdef('A')
clear A
isdef('A')
```

**SEE ALSO :**  exists 37, whereis 77, type 74, typeof 229, clear 30
1.3.64 isinf ................................. check for infinite entries

CALLING SEQUENCE:
r = isinf(x)

PARAMETERS:
  x : real or complex vector or matrix
  r : boolean vector or matrix

DESCRIPTION:
isinf(x) returns a boolean vector or matrix which contains true entries corresponding with infinite
x entries and false entries corresponding with finite x entries.

EXAMPLE:
isinf([1 0.01 -%inf %inf])

SEE ALSO: isnan 193

1.3.65 isnan ................................. check for ”Not a Number” entries

CALLING SEQUENCE:
r = isnan(x)

PARAMETERS:
  x : real or complex vector or matrix
  r : boolean vector or matrix

DESCRIPTION:
isnan(x) returns a boolean vector or matrix which contains true entries corresponding with ”Not a
Number” x entries and false entries corresponding with regular x entries.

EXAMPLE:
isnan([1 0.01 -%nan %inf-%inf])

SEE ALSO: isinf 193

1.3.66 isreal ............................... check if a variable as real or complex entries

CALLING SEQUENCE:
t = isreal(x)
t = isreal(x,eps)

PARAMETERS:
  x : vector or matrix with floating point entries or coefficients
  t : a boolean

DESCRIPTION:
isreal(x) returns true if x is stored as a real variable and false if x stores complex numbers.
isreal(x,eps) returns true if x is stored as a real variable or if maximum absolute value of imaginary
floating points if less or equal than eps.

EXAMPLE:
isreal([1 2])
isreal(1+0*%i)
isreal(1+0*%i,0)
isreal(1+%s)
isreal(sprand(3,3,0.1))
1.3.67   **kron**  

**Kronecker product (.*.)**

**CALLING SEQUENCE:**

\[\text{kron}(x,y)\]

\[x.*.y\]

**DESCRIPTION:**

Kronecker tensor product of two matrices \(x\) and \(y\). Same as \(x.*.y\) \(x\) and \(y\) can be sparse matrices.

**EXAMPLE:**

\[A=[1,2;3,4]\];
\[\text{kron}(A,A)\]
\[A.*.A\]
\[\text{sparse}(A).*\text{sparse}(A)\]
\[A(1,1)=%i;\]
\[\text{kron}(A,A)\]

1.3.68   **ldivf**  

**left symbolic division**

**CALLING SEQUENCE:**

\[\text{ldivf}('d','c')\]

**DESCRIPTION:**

returns the string ‘c\(\backslash\)d’ Trivial simplifications such as ‘1\(\backslash\)c’ = ‘c’ are performed.

**EXAMPLE:**

\[\text{ldivf}('1','1')\]
\[\text{ldivf}('a','0')\]
\[\text{ldivf}('a','x')\]
\[\text{ldivf}('2','4')\]

SEE ALSO:  **rdivf 208**,  **addf 164**,  **mult 203**,  **evstr 35**

1.3.69   **lex_sort**  

**lexicographic matrix rows sorting**

**CALLING SEQUENCE:**

\[\text{[N, [k]]=lex_sort(M [,sel] [,’unique’])}\]

**PARAMETERS:**

\(M\) : real matrix
\(N\) : real matrix
\(k\) : column vector of integers

**DESCRIPTION:**

\(N=\text{lex_sort}(M)\) sorts the rows (as a group) of the matrix \(M\) in ascending order. If required the output argument \(k\) contains the ordering: \([N,k]=\text{lex_sort}(M)\) returns \(k\) such as \(N\) is equal to \(M(k,:)\).

\(N=\text{lex_sort}(M,\text{sel} [,’unique’])\) produces the same result as the following sequence of instructions:
[N,k]=lex_sort(M(:,sel) [,,'unique']);
N=M(k,:)

The ‘unique’ flag has to be given if one wants to retain only unique rows in the result. Note that
lex_sort(M,sel,‘unique’) retains only rows such that M(:,sel) are unique.

EXAMPLE :
M=round(2*rand(20,3));
lex_sort(M)
lex_sort(M,‘unique’)
[N,k]=lex_sort(M,[1 3],‘unique’)

SEE ALSO: sort 213

1.3.70 linspace ................................. linearly spaced vector

CALLING SEQUENCE :
[v]=linspace(x1,x2 [,n])

PARAMETERS :
 x1,x2 : real or complex scalars
 n : integer (number of values) (default value = 100)
 v : real or complex row vector

DESCRIPTION :
Linearly spaced vector. linspace(x1, x2) generates a row vector of n (default value=100) linearly
equally spaced points between x1 and x2.

EXAMPLE :
linspace(1,2,10)

SEE ALSO: logspace 197

1.3.71 log ............................................. natural logarithm

CALLING SEQUENCE :
y=log(x)

PARAMETERS :
 x : constant vector or constant matrix

DESCRIPTION :
log(x) is the "element-wise" logarithm. y(i,j)=log(x(i,j)). For matrix logarithm see logm.

EXAMPLE :
exp(log([1,%i,-1,-%i]))

SEE ALSO: exp 508, logm 196, ieee 50
### 1.3.72 log10

**log10** Scilab Function

**Calling Sequence:**

\[ y = \log_{10}(x) \]

**Parameters:**

- \( x \): vector or matrix

**Description:**

Decimal logarithm. If \( x \) is a vector \( \log_{10}(x) = [\log_{10}(x_1), ..., \log_{10}(x_n)] \).

**Example:**

\[ 10^\log_{10}([1, \%i, -1, -\%i]) \]

**See Also:** \log_{195}, \hat_{47}, ieee_{50}

### 1.3.73 log2

**log2** base 2 logarithm

**Calling Sequence:**

\[ y = \log_{2}(x) \]

**Parameters:**

- \( x \): vector or matrix

**Description:**

Decimal logarithm. If \( x \) is a vector \( \log_{2}(x) = [\log_{2}(x_1), ..., \log_{2}(x_n)] \).

**Example:**

\[ 2^\log_{2}([1, \%i, -1, -\%i]) \]

**See Also:** \log_{195}, \hat_{47}, ieee_{50}, \log_{196}, \frexp_{185}

### 1.3.74 logm

**logm** square matrix logarithm

**Calling Sequence:**

\[ y = \logm(x) \]

**Parameters:**

- \( x \): square matrix

**Description:**

\( \logm(x) \) is the matrix logarithm of \( x \). The result is complex if \( x \) is not positive or definite positive. If \( x \) is a symmetric matrix, then calculation is made by schur form. Otherwise, \( x \) is assumed diagonalizable.

**Example:**

\[
\begin{align*}
A &= [1, 2; 3, 4]; \\
\logm(A) & \\
\expm(\logm(A)) & \\
A1 &= A * A'; \\
\logm(A1) & \\
\expm(\logm(A1)) & \\
A1(1,1) &= \%i; \\
\expm(\logm(A1)) &
\end{align*}
\]

**See Also:** \expm_{509}, \log_{195}
1.3.75  logspace  __________________________ logarithmically spaced vector

CALLING SEQUENCE:

logspace(d1,d2, [n])

PARAMETERS:

d1, d2 : real or complex scalar (special meaning for %pi)
n : integer (number of values) (default value = 50)

DESCRIPTION:

returns a row vector of n logarithmically equally spaced points between 10^d1 and 10^d2. If d2=%pi then the points are between 10^d1 and pi.

EXAMPLE:

logspace(1,2,10)

SEE ALSO:  linspace

1.3.76  max  __________________________ maximum

CALLING SEQUENCE:

[m [,k]]=max(A)
[m [,k]]=max(A,'c') or [m [,k]]=max(A,'r')
[m [,k]]=max(A1,A2,...,An)
[m [,k]]=max(list(A1,A2,...,An))

PARAMETERS:

A : real vector or matrix.
A1,...,An : a set of real vectors or matrices, all of the same size or scalar.

DESCRIPTION:

For A, a real vector or matrix, max(A) is the largest element A. [m,k]=max(A) gives in addition the index of the maximum. A second argument of type string ’r’ or ’c’ can be used: ’r’ is used to get a row vector m such that m(j) contains the maximum of the j th column of A (A(:,j)), k(j) gives the row indice which contain the maximum for column j. ’c’ is used for the dual operation on the rows of A.

m=max(A1,A2,...,An) where all the Aj are matrices of the same sizes,returns a vector or a matrix m of size size(m)=size(A1) such that m(i)= max( Aj(i)), j=1,...,n. [m,k]=max(A1,A2,...,An) gives in addition the vector or matrix k. for a fixed i, k(i) is the number of the first Aj(i) achieving the maximum.

[m,k]=max(list(A1,...,An)) is an equivalent syntax of [m,k]=max(A1,A2,...,An)

EXAMPLE:

[m,n]=max([1,3,1])
[m,n]=max([3,1,1],[1,3,1],[1,1,3])
[m,n]=max([[3,-2,1],1])
[m,n]=max(list([3,1,1],[1,3,1],[1,1,3]))
[m,n]=max(list(1,3,1))

SEE ALSO:  sort 213,  find 41,  mini 200
**1.3.77 maxi**

**CALLING SEQUENCE:**

\[ [m[,k]]=\text{maxi}(A) \]
\[ [m[,k]]=\text{maxi}(A,'c') \text{ or } [m[,k]]=\text{maxi}(A,'r') \]
\[ [m[,k]]=\text{maxi}(A1,A2,...,An) \]
\[ [m[,k]]=\text{maxi}(\text{list}(A1,A2,...,An)) \]

**PARAMETERS:**

A : real vector or matrix.
A1,...,An : a set of real vectors or matrices, all of the same size or scalar.

**DESCRIPTION:**

For A, a real vector or matrix, maxi(A) is the largest element A. [m,k]=maxi(A) gives in addition the index of the maximum. A second argument of type string 'r' or 'c' can be used: 'r' is used to get a row vector m such that m(j) contains the maximum of the jth column of A (A(:,j)), k(j) gives the row indice which contain the maximum for column j. 'c' is used for the dual operation on the rows of A.

\[ m=\text{maxi}(A1,A2,...,An) \], where all the Aj are matrices of the same sizes,returns a vector or a matrix m of size size(m)=size(A1) such that m(i)= max( Aj(i)), j=1,...,n. [m,k]=maxi(A1,A2,...,An) gives in addition the vector or matrix k. for a fixed i, k(i) is the number of the first Aj(i) achieving the maximum.

\[ [m,k]=\text{maxi}(\text{list}(A1,...,An)) \] is an equivalent syntax of [m,k]=maxi(A1,A2,...,An)

**EXAMPLE:**

\[ [m,n]=\text{maxi}([1,3,1]) \]
\[ [m,n]=\text{maxi}([[3,1,1],[1,3,1],[1,1,3]]) \]
\[ [m,n]=\text{maxi}([[3,-2,1],1]) \]
\[ [m,n]=\text{maxi}(\text{list}([[3,1,1],[1,3,1],[1,1,3]])) \]
\[ [m,n]=\text{maxi}(\text{list}(1,3,1)) \]

**SEE ALSO:** sort 213, find 41, mini 200

**1.3.78 mean**

**mean (row mean, column mean) of vector/matrix entries**

**CALLING SEQUENCE:**

\[ y=\text{mean}(x) \]
\[ y=\text{mean}(x,'r') \]
\[ y=\text{mean}(x,'c') \]

**PARAMETERS:**

x : real vector or matrix
y : scalar or vector

**DESCRIPTION:**

For a vector or a matrix x, y=mean(x) returns in the scalar y the mean of all the entries of x.
\[ y=\text{mean}(x,'r') \] (or, equivalently, y=mean(x,1)) is the rowwise mean. It returns in each entry of the column vector y the mean of each row of x.
\[ y=\text{mean}(x,'c') \] (or, equivalently, y=mean(x,2)) is the columnwise mean. It returns in each entry of the row vector y the mean of each column of x.

**EXAMPLE:**

Scilab Group January 1998
min Scilab Function

A=[1,2,10;7,7.1,7.01];
mean(A)
mean(A,'r')
mean(A,'c')

SEE ALSO: sum 222, median 199, st_deviation 221

1.3.79 median ___ median (row median, column median) of vector/matrix entries

CALLING SEQUENCE :

y=median(x)
y=median(x,'r')
y=median(x,'c')

PARAMETERS :

x : real vector or matrix
y : scalar or vector

DESCRIPTION :
For a vector or a matrix x, y=median(x) returns in the scalar y the median of all the entries of x.
y=median(x,'r') (or, equivalently, y=median(x,1)) is the rowwise median. It returns in each entry of the column vector y the median of each row of x.
y=median(x,'c') (or, equivalently, y=median(x,2)) is the columnwise median. It returns in each entry of the row vector y the median of each column of x.

EXAMPLE :
A=[1,2,10;7,7.1,7.01];
median(A)
median(A,'r')
median(A,'c')

SEE ALSO: sum 222, mean 198

1.3.80 min __________________________________________________________________ minimum

CALLING SEQUENCE :

[m [,k]]=min(A)
[m [,k]]=min(A,'c') or [m [,k]]=min(A,'r')
[m [,k]]=min(A1,A2,...,An)
[m [,k]]=min(list(A1,A2,...,An))

PARAMETERS :

A : real vector or matrix.
A1,...,An : a set of real vectors or matrices, all of the same size or scalar.

DESCRIPTION :

For A, a real vector or matrix, min(A) is the smallest element A. [m, k]=min(A) gives in addition the index of the minimum. A second argument of type string 'r' or 'c' can be used: 'r' is used to get a row vector m such that m(j) contains the minimum of the j th column of A (A(:,j)), k(j)
gives the row indice which contain the minimum for column j. ‘c’ is used for the dual operation on the rows of A.

\[ m = \min(A_1, A_2, \ldots, A_n) \]

where all the \( A_j \) are matrices of the same sizes, returns a vector or a matrix \( m \) of size \( \text{size}(m) = \text{size}(A_1) \) such that \( m(i) = \min( A_j(i) ) \), \( j=1, \ldots, n \). \([m,k]=\min(A_1,A_2,\ldots,A_n)\) gives in addition the vector or matrix \( k \). for a fixed \( i \), \( k(i) \) is the number of the first \( A_j(i) \) achieving the minimum.

\([m,k]=\min(\text{list}(A_1,\ldots,A_n))\) is an equivalent syntax of \([m,k]=\min(A_1,A_2,\ldots,A_n)\)

**EXAMPLE:**

\[
\begin{align*}
[m,n]=&\min([1,3,1]) \\
[m,n]=&\min([3,1,1],[1,3,1],[1,1,3]) \\
[m,n]=&\min(\text{list}([3,1,1],[1,3,1],[1,1,3])) \\
[m,n]=&\min(\text{list}(1,3,1))
\end{align*}
\]

**SEE ALSO:** sort 213, find 41, max 197

### 1.3.81 mini

**CALLING SEQUENCE:**

\[
\begin{align*}
&m \[,k\] = \text{mini}(A) \\
&m \[,k\] = \text{mini}(A,’c’) \text{ or } & [m \[,k\]] = \text{mini}(A,’r’) \\
&m \[,k\] = \text{mini}(A_1,A_2,\ldots,A_n) \\
&m \[,k\] = \text{mini}(\text{list}(A_1,A_2,\ldots,A_n))
\end{align*}
\]

**PARAMETERS:**

A : real vector or matrix.
A1, \ldots, An : a set of real vectors or matrices, all of the same size or scalar.

**DESCRIPTION:**

For \( A \), a real vector or matrix, \( \text{mini}(A) \) is the smallest element \( A \). \([m,k]=\text{mini}(A)\) gives in addition the index of the minimum. A second argument of type string ‘r’ or ‘c’ can be used: ‘r’ is used to get a row vector \( m \) such that \( m(j) \) contains the minimum of the \( j \)th column of \( A \) \( (A(:,j)) \). \( k(j) \) gives the row indice which contain the minimum for column \( j \). ‘c’ is used for the dual operation on the rows of \( A \).

\[ m = \text{mini}(A_1,A_2,\ldots,A_n) \]

where all the \( A_j \) are matrices of the same sizes, returns a vector or a matrix \( m \) of size \( \text{size}(m) = \text{size}(A_1) \) such that \( m(i) = \text{mini}( A_j(i) ) \), \( j=1, \ldots, n \). \([m,k]=\text{mini}(A_1,A_2,\ldots,A_n)\) gives in addition the vector or matrix \( k \). for a fixed \( i \), \( k(i) \) is the number of the first \( A_j(i) \) achieving the minimum.

\([m,k]=\text{mini}(\text{list}(A_1,\ldots,A_n))\) is an equivalent syntax of \([m,k]=\text{mini}(A_1,A_2,\ldots,A_n)\)

**EXAMPLE:**

\[
\begin{align*}
[m,n]=&\text{mini}([1,3,1]) \\
[m,n]=&\text{mini}([3,1,1],[1,3,1],[1,1,3]) \\
[m,n]=&\text{mini}(\text{list}([3,1,1],[1,3,1],[1,1,3])) \\
[m,n]=&\text{mini}(\text{list}(1,3,1))
\end{align*}
\]

**SEE ALSO:** sort 213, find 41, maxi 198, min 199
### 1.3.82 minus ———————————— - substraction operator, sign changes

**CALLING SEQUENCE:**

\[ X - Y \]
\[ -Y \]

**PARAMETERS:**

\[ X \]: scalar or vector or matrix of numbers, polynomials or rationals. It may also be a `syslin` list

\[ Y \]: scalar or vector or matrix of numbers, polynomials or rationals. It may also be a `syslin` list

**DESCRIPTION:**

Substraction

For numeric operands substraction as its usual meaning. If one of the operands is a matrix and the other one a scalar the the operation is performed element-wise. if \[ Y == [ ] \] \( X \) is returned; if \( X == [ ] \) \( -Y \) is returned.

Substraction may also be defined for other data types through "soft-coded" operations.

**EXAMPLE:**

\[ [1,2] - 1 \]
\[ [] - 2 \]
\[ \%s - 2 \]
\[ 1/\%s - 2 \]
\[ "cat" + "enate" \]

**See Also:** `addf`, `mtlb_mode`

### 1.3.83 modulo ———————————— symetric arithmetic remainder modulo m

pmodulo - positive arithmetic remainder modulo m

**CALLING SEQUENCE:**

\[ i = \text{modulo}(n,m) \]
\[ i = \text{pmodulo}(n,m) \]

**PARAMETERS:**

\[ n,m \]: integers

**DESCRIPTION:**

**modulo** computes \( i = n \mod m \) i.e. remainder of \( n \) divided by \( m \) (\( n \) and \( m \) integers).

\[ i = n - m \times \text{int}(n / m). \] Here the answer may be negative if \( n \) or \( m \) are negative.

**pmodulo** computes \( i = n - m \times \text{floor}(n \div m) \) the answer is positive or zero

**EXAMPLE:**

\[ n=[1,2,10,15]; m=[2,2,3,5]; \]
\[ \text{modulo}(n,m) \]
\[ \text{modulo}(-3,9) \]
\[ \text{pmodulo}(-3,9) \]
1.3.84  **mps2linpro**  convert lp problem given in MPS format to linpro format

**CALLING SEQUENCE:**

```plaintext
lp = mps2linpro(mps)
[p,C,b,ci,cs,mi] = mps2linpro(mps)
```

**PARAMETERS:**

- `mps` : either a character string, path of the MPS file, or an mps data structure returned by `readmps`
- `lp` : a linpro data tlist with following fields:
  - `p` : real (column) vector (dimension `n`)
  - `C` : real matrix (dimension `(mi + md) x n`) (If no constraints are given, you can set `C = []`)  
  - `b` : RHS vector (dimension `1 x (mi + md)`)  
  - `ci` : (column) vector of lower-bounds (dimension `n`). If there are no lower bound constraints, put `ci = []`. If some components of `x` are bounded from below, set the other (unconstrained) values of `ci` to a very large negative number (e.g. `ci(j) = -(eps)^(-1)`).  
  - `cs` : (column) vector of upper-bounds. (Same remarks as above).  
  - `mi` : number of equality constraints (i.e. `C(1:mi,:) x = b(1:mi)`)  

**DESCRIPTION:**

`mps2linpro` forms Linear programing data compatible with linpro out of MPS data format.

**SEE ALSO:**  `linpro 429`, `readmps 256`

1.3.85  **mtlb_sparse**  convert sparse matrix

**CALLING SEQUENCE:**

```plaintext
Y=mtlb_sparse(X)
```

**PARAMETERS:**

- `X` : sparse matrix
- `Y` : sparse matrix in Matlab format

**DESCRIPTION:**

`Y=mtlb_sparse(X)` is used to convert `X`, a Scilab sparse matrix, to Matlab format. `Y` is the a variable with type 7, i.e. `type(Y)` is equal to 7. This function should be used in mexfiles (a Matlab mexfile containing sparse matrices can be used only if the Scilab sparse matrices are converted to that format). The functions `full` and `spget` work with this format. Other operations and functions using this format can be overloaded with Scilab functions using the prefix "%msp". For instance the function `%msp_p(x)` (see SCIDIR/macros/percent directory) is used to display such "type 7" objects.

**EXAMPLE:**

```plaintext
X=sparse(rand(2,2)); Y=mtlb_sparse(X);
Y, full(Y), [ij,v,mn]=spget(Y)
```

**SEE ALSO:**  `full 186`, `spget 216`
1.3.86  mulf  symbolic multiplication

CALLING SEQUENCE:
mulf('d','c')

DESCRIPTION:
returns the string 'c*d'  Trivial simplifications such as '1*c' = 'c' are performed.

EXAMPLE:
mulf('1','a')
mulf('0','a')
'a'+'b'  //Caution...

SEE ALSO:  rdivf 208, addf 164, subf 222

1.3.87  nnz  number of non zero entries in a matrix

CALLING SEQUENCE:
n=nnz(X)

PARAMETERS:
X : real or complex sparse (or full) matrix
n : integer, the number of non zero elements in X

DESCRIPTION:
nnz counts the number of non zero entries in a sparse or full matrix

EXAMPLE:
sp=sparse([1,2;4,5;3,10],[1,2,3]);
nnz(sp)
a=[1 0 0 0 2];
nnz(a)

SEE ALSO:  spget 216

1.3.88  norm  matrix norms

CALLING SEQUENCE:
[y]=norm(x [,flag])

PARAMETERS:
x : real or complex vector or matrix (full or sparse storage)
flag : string (type of norm) (default value =2)

DESCRIPTION:
For matrices

norm(x)  : or norm(x,2)  is the largest singular value of x  (max(svd(x))).
norm(x,1)  : The L_1 norm x  (the largest column sum : maxi(sum(abs(x),'r'))) .
norm(x,'inf'), norm(x,%inf) : The infinity norm of x (the largest row sum : \( \max_i \sum_j |x_{ij}| \)).
norm(x,'fro') : Frobenius norm i.e. \( \sqrt{\sum_{ij} x_{ij}^2} \).

For vectors
norm(v,p) : \( \|v\|_p \) norm \( \left( \sum_i |v_i|^p \right)^{1/p} \).
norm(v) = norm(v,2) : l_2 norm
norm(v,'inf') : max(|v_i|).

EXAMPLE :
A=[1,2,3];
norm(A,1)
norm(A,'inf')
A=[1,2;3,4]
max(svd(A))-norm(A)
A=sparse([1 0 0 33 -1])
norm(A)

SEE ALSO: h_norm 382, dhnorm 376, h2norm 381, abs 162

1.3.89 not ------------------------------- - logical not

CALLING SEQUENCE:
~A

DESCRIPTION:
~A gives the element-wise negation of the elements of the boolean matrix A.

EXAMPLES:
~[%t %t %f]

SEE ALSO: and 165, or 205, find 41

1.3.90 ones ---------------------------------------- matrix made of ones

CALLING SEQUENCE:
y=ones(m1,m2,...)
y=ones(x)
y=ones()

PARAMETERS:
x, y : matrices
m1, m2, .. : integers

DESCRIPTION:
Returns a matrix made of ones.
ones(m1,m2) returns a (m1,m2) matrix full of ones.
ones(m1,m2,...,mn) : creates a (m1,m2,...,mn) matrix full of ones.
ones(x) returns a matrix full of ones with the same size that x.
ones(x) is also valid for x a syslin list.

Note that ones(3) is ones(a) with a=3 i.e it is NOT a 3x3 matrix!
one() is equivalent to ones(1,1).

EXAMPLE:
ones(3)
one(3,3)
one(2,3,2)

SEE ALSO: eye 184, zeros 231

### 1.3.91 or - logical or

**CALLING SEQUENCE:**
- or(A), or(A,"*")
- or(A,"r"), or(A,1)
- or(A,"c"), or(A,2)

**DESCRIPTION:**
or(A) gives the or of the elements of the boolean matrix A. or(A) is true (\%t) iff at least one entry of A is \%t.
y=or(A,"r") (or, equivalently, y=or(A,1)) is the rowwise or. It returns in each entry of the row vector y the or of the rows of x (The or is performed on the row index: y(j)= or(A(i,j),i=1,m)).
y=or(A,"c") (or, equivalently, y=or(A,2)) is the columnwise or. It returns in each entry of the column vector y the or of the columns of x (The or is performed on the column index: y(i)= or(A(i,j),j=1,n)).
A\|B gives the element-wise logical or of the boolean matrices A and B. A and B must be matrices with the same dimensions or one from them must be a single boolean.

**EXAMPLES:**
or([\%t \%t \%f])
[\%t \%f] \| [\%f \%t \%t]
[\%t \%t \%f] \| \%f

SEE ALSO: and 165, not 204, find 41

### 1.3.92 pen2ea pencil to E,A conversion

**CALLING SEQUENCE:**
- [E, A]=pen2ea(Fs)

**PARAMETERS:**
- Fs: matrix pencil s*E-A
- E, A: two matrices such that Fs=s*E-A

**DESCRIPTION:**
Utility function. Given the pencil Fs=s*E-A, returns the matrices E and A.

**EXAMPLE:**
E=[1,0];A=[1,2];s=poly(0,’s’);
[E, A]=pen2ea(s*E-A)
1.3.93 pertrans

CALLING SEQUENCE:

[Y]=pertrans(X)

PARAMETERS:

X : real or complex matrix
Y : real or complex matrix

DESCRIPTION:

Y=pertrans(X) returns the pertranspose of X, i.e. the symmetric of X w.r.t the second diagonal (utility function).

EXAMPLE:

A=[1,2;3,4]
pertrans(A)

1.3.94 prod

CALLING SEQUENCE:

y=prod(x)
y=prod(x,'r') or y=prod(x,1)
y=prod(x,'c') or y=prod(x,2)

PARAMETERS:

x : real or complex vector or matrix
y : real or complex scalar or matrix

DESCRIPTION:

For a vector or a matrix x, y=prod(x) returns in the scalar y the prod of all the entries of x, e.g. prod(1:n) is n!
y=prod(x,'r') (or, equivalently, y=prod(x,1)) computes the rows elementwise product of x. y is the row vector: y(i)=prod(x(i,:)).
y=prod(x,'c') (or, equivalently, y=prod(x,2)) computes the columns elementwise product of x. y is the column vector: y(i)=prod(x(:,i)).

prod is not implemented for sparse matrices.

EXAMPLE:

A=[1,2;0,100];
prod(A)
prod(A,'c')
prod(A,'r')

SEE ALSO: sum 222, cumprod 179
### 1.3.95 rand

**random number generator**

**CALLING SEQUENCE:**

```plaintext
call rand(m1,m2,.. [,key])
call rand(x [, key])
call rand()
call rand(key)
call rand("seed" [,n])
call rand("info")
```

**PARAMETERS:**

- `mi`: integers
- `key`: character string with value in "uniform", "normal"
- `x`: a matrix. Only its dimensions are taken into account.

**DESCRIPTION:**

random matrix generator.

Without key argument the syntaxes below produce random matrices with the current random generator (default is "uniform")

- `rand(m1,m2)` is a random matrix of dimension `m1` by `m2`.
- `rand(m1,m2,..,mn)` is a random matrix of dimension `m1` by `m2`... by `mn`.
- `rand(a)` is a random matrix of same size as `a`. `rand(a)` is complex if `a` is a complex matrix
- `rand()` : with no arguments gives a scalar whose value changes each time it is referenced.

If present, the key argument allows to specify an other random generator.

- `rand('uniform')`: The current random generator is set to a uniform random number generator. Random numbers are uniformly distributed in the interval (0,1).
- `rand('normal')`: The current random generator is set to a Gaussian (with mean 0 and variance 1) random number generator.

- `str=rand('info')` return the type of the default random generator ('uniform' or 'normal')

It is possible to (re-)initialize the seed of the rand generator:

- `rand('seed')` returns the current value of the seed.
- `rand('seed',n)` puts the seed to `n`. (`n=0` at first call).

**EXAMPLE:**

```plaintext
x=rand(10,10,'uniform')
rand('normal')
rand('info')
y=rand(x,'normal');
x=rand(2,2,2)
```

**SEE ALSO:** ssrand

### 1.3.96 rat

**Floating point rational approximation**

**CALLING SEQUENCE:**

Scilab Group

April 1993
[N,D]=rat(x [,tol])
y=rat(x [,tol])

PARAMETERS:

x : real vector or matrix
n : integer vector or matrix
d : integer vector or matrix
y : real vector or matrix

DESCRIPTION:

[N,D] = rat(x,tol) returns two integer matrices so that N./D is close to x in the sense that
abs(N./D - x) <= tol*abs(x). The rational approximations are generated by truncating con-
tinued fraction expansions. tol = 1.e-6*norm(x,1) is the default. y = rat(x,tol) return
the quotient N./D
See Also: int 188, round 209

EXAMPLES:

[n,d]=rat(%pi)
[n,d]=rat(%pi,1.d-12)
n/d-%pi

1.3.97  rdivf _______________________________ right symbolic division

CALLING SEQUENCE:

["r"]=ldivf("d","c")

PARAMETERS:

"d","c","r" : strings

DESCRIPTION:

returns the string "c/d" Trivial simplifications such as "c/1" = "c" are performed.

EXAMPLE:

ldivf('c','d')
ldivf('1','2')
ldivf('a','0')

See Also: ldivf 194

1.3.98  real ________________________________ real part

CALLING SEQUENCE:

[y]=real(x)

PARAMETERS:

x : real or complex vector or matrix
y : real matrix

DESCRIPTION:

real(x) is the real part of x (See %i to enter complex numbers).

See Also: imag 188
1.3.99  round  ____________________________________________ rounding

CALLING SEQUENCE :

y=round(x)

PARAMETERS :

x  : real or complex matrix
y  : integer or complex (with integer real and imag) matrix

DESCRIPTION :

round(x) rounds the elements of x to the nearest integers.

EXAMPLE :

round([1.9 -2.5])=[2,-3]
round(1.6+2.1*%i)=(2+2*%i)
round(-%inf)
x=rand()*10^20;round(x)-x

SEE ALSO:   int 188, floor 185, ceil 176

1.3.100  sign  ____________________________________________ sign function

DESCRIPTION :

X=sign(A) returns the matrix made of the signs of A(i,j). For complex A, sign(A) = A./abs(A).

EXAMPLE :

sign(rand(2,3))
sign(1+%i)

SEE ALSO:   abs 162

1.3.101  signm  ____________________________________________ matrix sign function

DESCRIPTION :

For square and Hermitian matrices X=sign(A) is matrix sign function.

EXAMPLE :

A=rand(4,4);B=A+A';X=sign(B);spec(X)

SEE ALSO:  sign 209
1.3.102  \textbf{sin} \hspace{2cm} \text{sine function}

\textbf{CALLING SEQUENCE :}

\[ [t] = \text{sin}(x) \]

\textbf{PARAMETERS :}

\( x \): real or complex vector or matrix

\textbf{DESCRIPTION :}

For a vector or a matrix, \( \text{sin}(x) \) is the sine of its elements. For matrix sine use \( \text{sinm}(X) \) function.

\textbf{EXAMPLE :}

\[ \text{asin}(\text{sin}([1,0,\%i])) \]

\textbf{SEE ALSO :} \text{sinm} 211

1.3.103  \textbf{sinh} \hspace{2cm} \text{hyperbolic sine}

\textbf{CALLING SEQUENCE :}

\[ [t] = \text{sinh}(x) \]

\textbf{PARAMETERS :}

\( x, t \): real or complex vectors/matrices

\textbf{DESCRIPTION :}

the elements of vector \( t \) are the hyperbolic sine of elements of vector \( x \).

\textbf{EXAMPLE :}

\[ \text{asinh}(\text{sinh}([0,1,\%i])) \]

\textbf{SEE ALSO :} \text{asinh} 166

1.3.104  \textbf{sinhm} \hspace{2cm} \text{matrix hyperbolic sine}

\textbf{CALLING SEQUENCE :}

\( t = \text{sinhm}(x) \)

\textbf{PARAMETERS :}

\( x, t \): real or complex square matrix

\textbf{DESCRIPTION :}

\( \text{sinhm} \) is the matrix hyperbolic sine of the matrix \( x \). \( t = (\text{expm}(x) - \text{expm}(-x)) / 2 \)

\textbf{EXAMPLE :}

\( A = [1,2;2,3] \)
\( \text{asinhm} (\text{sinhm}(A)) \)
\( A(1,1) = \%i; \text{sinhm}(A) - (\text{expm}(A) - \text{expm}(-A))/2 \)  //Complex case

\textbf{SEE ALSO :} \text{sinh} 210

Scilab Group  
April 1993  
210
1.3.105  sinm  .......................................................... matrix sine function

CALLING SEQUENCE :

t=sinm(x)

PARAMETERS :

x : real or complex square matrix

DESCRIPTION :

sinm(x) is matrix sine of x matrix.

EXAMPLE :

A=[1,2;2,4];
sinm(A)+0.5*i*(expm(%i*A)-expm(-%i*A))

SEE ALSO: sin 210, asinm 167

1.3.106  size  ........................................................... size of objects

CALLING SEQUENCE :

y=size(x [,sel])

[nc]=size(x)

PARAMETERS :

x : matrix (including transfer matrix) or list or linear system (syslin)
y : 1x2 integer vector or integer number
sel : a scalar or a character string
nr,nc : two integers

DESCRIPTION :

Applied to:

a matrix (constant, polynomial, string, boolean, rational) x, with only one lhs argument
size returns a 1x2 vector [number of rows, number of columns].

Called with LHS=2, returns nr,nc = [number of rows, number of columns].

sel may be used to specify what dimension to get:

1 or 'r' : to get the number of rows
2 or 'c' : to get the number of columns
'*' : to get the product of rows and column numbers

Applied to:

a list it returns the number of elements. In this case only y=size(x) syntax can be used

Applied to:

a linear system, y=size(x) returns in y the (row) vector [number of outputs, number of inputs] i.e. the dimension of the corresponding transfer matrix. The syntax [nr,nc]=size(x) is also valid (with (nr,nc)=(y(1),y(2))).

If x is a linear system in state-space form, then [nr,nc,nx]=size(x) returns in addition the dimension nx of the A matrix of x.

Applied to:

an hypermatrix y=size(x) returns the vector of hypermatrix dimensions. [n1,n2,...nn]=size(x) returns the hypermatrix dimensions. ni=size(x,i) returns the ith dimension and size(x,'*') returns the product of dimensions.

EXAMPLES :

Scilab Group  April 1993 211
[n,m]=size(rand(3,2))
[n,m]=size(["a","b";'c','d'])
x=ssrand(3,2,4);[ny,nu]=size(x)
[ny,nu]=size(ss2tf(x))
[ny,nu,nx]=size(x)

SEE ALSO: length 281, syslin 224

1.3.107 smooth ------------------------------------- smoothing by spline functions

CALLING SEQUENCE:

[pt]=smooth(ptd [,step])

PARAMETERS:

ptd : (2xn) real vector
step : real (discretization step of abscissae) (default=0.01*magnitude(v))
pt : (2xn) real vector

DESCRIPTION:

this function computes interpolation by spline functions for a given set of points in the plane. The coordinates are (ptd(1,i),ptd(2,i)). The components ptd(1,:) must be in ascending order. The default value for the step is abs(maxi(ptd(1,:))-mini(ptd(1,:)))/100

EXAMPLE:

x=[1 10 20 30 40];
y=[1 30 -10 20 40];
plot2d(x',y',[3],"011","",[-10,-40,50,50]);
yi=smooth([x;y],0.1);
plot2d(yi(1,:)',yi(2,:)',[1],"000");

SEE ALSO: splin 217, interp 189, interpln 190

1.3.108 solve --------------------------------------- symbolic linear system solver

CALLING SEQUENCE:

[x]=solve(A,b)

PARAMETERS:

A,b,c : matrix (resp. vectors) of character strings

DESCRIPTION:

solves A*x = b when A is an upper triangular matrix made of character strings.

EXAMPLE:

A=["1","a","0","2"];  //Upper triangular
b=["x";'y"];
w=solve(A,b)
a=1;x=2;y=5;
evstr(w)
inv([1,1;0,2])*[2;5]

SEE ALSO: trianfml 228

Scilab Group  April 1993  212
1.3.109  sort ________________________ decreasing order sorting

CALLING SEQUENCE :

\[ [s, [k]] = \text{sort}(v) \]
\[ [s, [k]] = \text{sort}(v,'r') \]
\[ [s, [k]] = \text{sort}(v,'c') \]

PARAMETERS :

\( v \) : real or complex vector/matrix; sparse vector; character string vector/matrix
\( s \) : real or complex vector or matrix; sparse vector; character string vector/matrix
\( k \) : vector or matrix of integers

DESCRIPTION :

\( s = \text{sort}(v) \) sorts \( v \) in decreasing order. If \( v \) is a matrix, sorting is done columnwise, \( v \) being seen as the stacked vector \( v(:) \). \([s, k] = \text{sort}(v) \) gives in addition the indices of entries of \( s \) in \( v \), i.e. \( v(k(:)) \) is the vector \( s \).
\( s = \text{sort}(v,'r') \) sorts the rows of \( v \) in decreasing order i.e. each column of \( s \) is obtained from each column of \( v \) by reordering it in decreasing order. \([s, k] = \text{sort}(v,'r') \) returns in addition in each column of \( k \) the indices such that \( v(k(:,i),i) = s(:,i) \) for each column index \( i \).
\( s = \text{sort}(v,'c') \) sorts the columns of \( v \) in decreasing order i.e. each row of \( s \) is obtained from each row of \( v \) by reordering it in decreasing order. \([s, k] = \text{sort}(v,'c') \) returns in addition in each row of \( k \) the indices such that \( v(i,k(i,:)) = s(i,:) \) for each row index \( i \).
Complex matrices or vectors are sorted w.r.t their magnitude.
\( y = \text{sort}(A) \) is valid when \( A \) is a sparse vector. Column/row sorting is not implemented for sparse matrices.

EXAMPLE :

\[ [s, p] = \text{sort}\left(\text{rand}(1,10)\right); \] //\( p \) is a random permutation of 1:10
\[ A = [1, 2, 5; 3, 4, 2]; \]
\[ [\text{Asorted}, q] = \text{sort}(A); A(q(:)) - \text{Asorted}(:) \]
\[ v = 1:10; \]
\[ \text{sort}(v) \]
\[ \text{sort}(v') \] //Does nothing for row vectors
\[ \text{sort}(v,'c') \]

SEE ALSO:  \text{find}^{41}

1.3.110  \text{sp2adj} ________________________ converts sparse matrix into adjacency form

CALLING SEQUENCE :

\[ [xadj, \text{adjncy}, anz] = \text{sp2adj}(A) \]

PARAMETERS :

\( A \) : real or complex sparse matrix (nz non-zero entries)
\( xadj \) : integer vector of length \((n+1)\).
sparse Scilab Function

adjncy
: integer vector of length nz containing the row indices for the corresponding elements in anz

anz
: column vector of length nz, containing the non-zero elements of A

DESCRIPTION:

\texttt{sp2adj} converts a sparse matrix into its adjacency form (utility function).

\texttt{VA} = n \times m \texttt{sparse matrix. } \texttt{Vadj, adjncy, anz} = adjacency representation of \texttt{VA} i.e:

-LP
\texttt{Vadj}(j+1)-\texttt{adj}(j) = number of non zero entries in row j.
\texttt{Vadjncy} = column index of the non zeros entries in row 1, row 2,..., row n.
\texttt{Vanz} = values of non zero entries in row 1, row 2,..., row n.
\texttt{Vadj} is a (column) vector of size n+1 and \texttt{Vadjncy} is an integer (column) vector of size \texttt{Vnz=nnz(A)}.
\texttt{Vanz} is a real vector of size \texttt{Vnz=nnz(A)}.

EXAMPLE:

A = sprand(100,50,.05);
[xadj,adjncy,anz]= sp2adj(A);
[n,m]=size(A);
p = adj2sp(xadj,adjncy,anz,[n,m]);
A-p,

SEE ALSO: adj2sp 164, sparse 214, spcompack 215, spget 216

1.3.111 sparse ----------------------------- sparse matrix definition

CALLING SEQUENCE:

sp=sparse(X)
sp=sparse(ij,v [,mn])

PARAMETERS:

X : real or complex full (or sparse) matrix
ij : two columns integer matrix (indices of non-zeros entries)
mn : integer vector with two entries (row-dimension, column-dimension)
sp : sparse matrix

DESCRIPTION:

\texttt{sparse} is used to build a sparse matrix. Only non-zero entries are stored.
\texttt{sp = sparse(X)} converts a full matrix to sparse form by squeezing out any zero elements. (If \texttt{X} is already sparse \texttt{sp} is \texttt{X}).
\texttt{sp=sparse(ij,v [,mn])} builds an \texttt{mn(1)}-by-\texttt{mn(2)} sparse matrix with \texttt{sp(ij(k,1),ij(k,2))=v(k)}.
\texttt{ij} and \texttt{v} must have the same column dimension. If optional \texttt{mn} parameter is not given the \texttt{sp} matrix dimensions are the max value of \texttt{ij(:,1)} and \texttt{ij(:,2)} respectively.
Operations (concatenation, addition, etc.) with sparse matrices are made using the same syntax as for full matrices. Elementary functions are also available (abs, maxi, sum, diag,...) for sparse matrices. Mixed operations (full-sparse) are allowed. Results are full or sparse depending on the operations.

**EXAMPLE:**

\[
sp=sparse([1,2;4,5;3,10],[1,2,3])
\]

\[
size(sp)
\]

\[
x=rand(2,2);abs(x)-full(abs(sparse(x)))
\]

**SEE ALSO:** full 186, spget 216, sprand 218, speye 216, lufact 520

### 1.3.112 spcompack

**Utility function** \fVspcompak\fR is used to convert a compressed adjacency representation into standard adjacency representation.

**CALLING SEQUENCE:**

\[
adjncy = spcompak(xadj,xlindx,lindx)
\]

**PARAMETERS:**

- `xadj`: integer vector of length \((n+1)\).
- `xlindx`: integer vector of length \(n+1\) (pointers).
- `lindx`: integer vector
- `adjncy`: integer vector

**DESCRIPTION:**

Utility function \fVspcompak\fR is used to convert a compressed adjacency representation into standard adjacency representation.

**EXAMPLE:**

// A is the sparse matrix:
\[
A=[1,0,0,0,0,0,0;
   0,1,0,0,0,0,0;
   0,0,1,0,0,0,0;
   0,0,1,1,0,0,0;
   0,0,1,1,1,0,0;
   0,0,1,1,0,1,0;
   0,0,1,1,0,1,1];
\]

\[
A=sparse(A);
\]

//For this matrix, the standard adjacency representation is given by:
\[
xadj=[1,2,3,8,12,13,15,16];
\]

\[
adjncy=[1, 2, 3,4,5,6,7, 4,5,6,7, 5, 6,7, 7];
\]

// see sp2adj).

// increments in vector xadj give the number of non zero entries in each column

// ie there is 2-1=1 entry in the column 1
// there is 3-2=1 entry in the column 2
// there are 8-3=5 entries in the column 3
// 12-8=4
// etc
// The row index of these entries is given by the adjncy vector
// for instance,
// adjncy (3:7)=adjncy(xadj(3):xadj(4)-1)=[3,4,5,6,7]
// says that the 5=xadj(4)-xadj(3) entries in column 3 have row
// indices 3,4,5,6,7.
// In the compact representation, the repeated sequences in adjncy
// are eliminated.
// Here in adjncy the sequences 4,5,6,7 and 7 are eliminated.
// The standard structure (xadj,adjncy) takes the compressed form (lindx,xlindx)
lindx=[1, 2, 3,4,5,6,7, 5, 6,7];
xlindx=[1,2,3,8,9,11];
// (Columns 4 and 7 of A are eliminated).
// A can be reconstructed from (xadj,xlindx,lindx).
[xadj,adjncy,anz]= sp2adj(A);
adjncy = spcompack(xadj,xlindx,lindx)

SEE ALSO:  sp2adj 213,  adj2sp 164,  spget 216

1.3.113  speye ___________________________ sparse identity matrix

CALING SEQUENCE :
Isp=speye(nrows,ncols)
Isp=speye(A)

PARAMETERS :
nrows : integer (number of rows)
ncols : integer (number of columns)
A : sparse matrix
sp : sparse identity matrix

DESCRIPTION :
Isp=speye(nrows,ncols) returns a sparse identity matrix Isp with nrows rows, ncols columns. (Non square identity matrix have a maximal number of ones along the main diagonal).
Isp=speye(A) returns a sparse identity matrix with same dimensions as A. If [m,n]=size(A), speye(m,n) and speye(A) are equivalent. In particular speye(3) is not equivalent to speye(3,3).

EXAMPLE :
eye(3,3)-full(speye(3,3))

SEE ALSO:  sparse 214,  full 186,  eye 184,  spzeros 218,  spones 218

1.3.114  spget __________________________ retrieves entries of sparse matrix

CALLING SEQUENCE :
[ij,v,mn]=spget(sp)

PARAMETERS :
Scilab Group January 1995 216
sp : real or complex sparse matrix
ij : two columns integer matrix (indices of non-zeros entries)
mn : integer vector with two entries (row-dimension, column-dimension)
v : column vector

DESCRIPTION :
spget is used to convert the internal representation of sparse matrices into the standard ij, v representation.
Non zero entries of sp are located in rows and columns with indices in ij.

EXAMPLE :
sp=sparse([1,2;4,5;3,10],[1,2,3])
[ij,v,mn]=spget(sp);

SEE ALSO : sparse 214, sprand 218, speye 216, lufact 520

1.3.115 s lin .......................... spline function

CALLING SEQUENCE :
d=splin(x,f [,"periodic"])

PARAMETERS :
x : real vector
f : real vector of same size as x
"periodic" : string flag (a periodic spline is looked for)

DESCRIPTION :
Given values fi of a function f at given points xi (fi=f(xi)) this primitive computes a third order spline function S which interpolates the function f. The components of x must be in increasing order. For a periodic spline f(1) must equal f(n); S is defined through the triple (x,f,d) where d=spline(x,f) is the vector of the estimated derivatives of S at xi (fi=S(xi),di=S'(xi)). This function should be used in conjunction with interp.
In the case "periodic" n must be chosen >= 3. In the non periodic case, n must be >= 4 (but n=3 gives some kind of results) and the boundary/end conditions for the spline are of type "not-a-knot conditions" and prescribe (if x1, x2, ..., xn are the interpolation nodes):

S'''(x2-) = S'''(x2+)
S'''(x(n-1)-) = S'''(x(n-1)+)

so the first cubic polynomial p1 is equal to the second p2 and the same is valid for the 2 last ones: p[n-2]=p[n-1].

EXAMPLE :
x=0:0.5:10;f=sin(x);
d=splin(x,f);
S=interp(0:0.1:10,x,f,d);
plot2d(x',f',-1);
plot2d((0:0.1:10)',S',2,'000')

SEE ALSO : interp 189, smooth 212
1.3.116  **spones**  

**sparse matrix**

**SYNTAX:**
sp = spones(A)

**PARAMETERS:**
A : sparse matrix  
sp : sparse matrix  

**DESCRIPTION:**
sp = spones(A) generates a matrix with the same sparsity structure as A, but with ones in the nonzero positions;

**EXAMPLE:**
A = sprand(10, 12, 0.1);  
sp = spones(A)  
B = A' = 0  
bool2s(B)

**SEE ALSO:**  sparse 214,  full 186,  eye 184,  speye 216,  spzeros 218

1.3.117  **sprand**  

**sparse random matrix**

**CALLING SEQUENCE:**
sp = sprand(nrows, ncols, fill [,typ])

**PARAMETERS:**

- nrows : integer (number of rows)  
- ncols : integer (number of columns)  
- fill : filling coefficient (density)  
- typ : character string ('uniform' (default) or 'normal')  
- sp : sparse matrix

**DESCRIPTION:**
sp = sprand(nrows, ncols, fill) returns a sparse matrix sp with nrows rows, ncols columns and approximately fill*nrows*ncols non-zero entries.  
If typ = 'uniform' uniformly distributed values are generated. If typ = 'normal' normally distributed values are generated.

**EXAMPLE:**
W = sprand(100, 1000, 0.001);

**SEE ALSO:**  sparse 214,  full 186,  rand 207,  speye 216

1.3.118  **spzeros**  

**sparse zero matrix**

**SYNTAX:**
sp = spzeros(nrows, ncols)  
sp = spzeros(A)

**PARAMETERS:**

Scilab Group  January 1995  218
**nrows** : integer (number of rows)
**ncols** : integer (number of columns)
**A** : sparse matrix
**sp** : sparse zero matrix

**DESCRIPTION**:

- `sp=spzeros(nrows,ncols,fill)` returns a sparse zero matrix `sp` with `nrows` rows, `ncols` columns. (Equivalent to `sparse([],[],[nrow,ncols])`)
- `sp=spzeros(A)` returns a sparse zero matrix with same dimensions as `A`. If `[m,n]=size(A).spzeros(m,n)` and `spzeros(A)` are equivalent. In particular `spzeros(3)` is not equivalent to `spzeros(3,3)`.

**EXAMPLE**:

```
sum(spzeros(1000,1000))
```

**SEE ALSO**:
- `sparse 214`, `full 186`, `eye 184`, `speye 216`, `spones 218`

### 1.3.119 `sqrt` .................................................................................. square root

**CALLING SEQUENCE**:

```
y=sqrt(x)
```

**PARAMETERS**:

- **x** : real or complex scalar or vector

**DESCRIPTION**:

`sqrt(x)` is the vector of the square root of the `x` elements. Result is complex if `x` is negative.

**EXAMPLE**:

```
sqrt([2,4])
sqrt(-1)
```

**SEE ALSO**:
- `hat 47`, `sqrtm 219`

### 1.3.120 `sqrtm` .................................................................................. matrix square root

**CALLING SEQUENCE**:

```
y=sqrtm(x)
```

**PARAMETERS**:

- **x** : real or complex square matrix

**DESCRIPTION**:

`y=sqrtm(x)` is the matrix square root of the `x` matrix `(x=y^2)` Result may not be accurate if `x` is not symmetric.

**EXAMPLE**:

```
x=[0 1;2 4]
w=sqrtm(x);
规范(w*w-x)
x(1,2)=%i;
w=sqrtm(x);norm(w*w-x,1)
```

**SEE ALSO**:
- `expm 509`, `sqroot 537`
1.3.121 squarewave ____________ generates a square wave with period 2*%pi

CALLING SEQUENCE:

\[ x = \text{squarewave}(t [,\%]) \]

PARAMETERS:

t : real vector, time discretization
x : real vector, the wave value at each time point in set (-1,+1)
\% : real positive scalar, the percent of the period in which the signal is positive. Default value is 50

DESCRIPTION:

\( \text{squarewave}(t) \) generates the vector of the values of the square wave with period 2*%pi at each date given in the \( t \) vector.
\( \text{squarewave}(t,\%) \) generates a square wave such that \( \% \) is the percent of the period in which the signal is positive.

EXAMPLE:

\[
t=(0:0.1:5*%pi)';
plot2d1('onn',t,[2*sin(t),1.5*squarewave(t),squarewave(t,10)])
\]

SEE ALSO: \sin 210, \cos 177

1.3.122 ssprint ............................ pretty print for linear system

CALLING SEQUENCE:

\[ \text{ssprint}(s1 [,\text{out}]) \]

PARAMETERS:

s1 : list(syslin list)
out : output (default value out=%io(2))

DESCRIPTION:

pretty print of a linear system in state-space form \( s1=(A,B,C,D) \) syslin list.

EXAMPLE:

\[
a=[1 1;0 1];b=[0 1;1 0];c=[1,1];d=[3,2];
ssprint(syslin('c',a,b,c,d))
ssprint(syslin('d',a,b,c,d))
\]

SEE ALSO: texprint 665

1.3.123 ssrand ________________________ random system generator

CALLING SEQUENCE:

\[ s1=\text{ssrand}(\text{nout},\text{nin},\text{nstate}) \]
\[ [s1,U]=\text{ssrand}(\text{nout},\text{nin},\text{nstate},\text{flag}) \]

PARAMETERS:

Scilab Group April 1993 220
stdeviation (Scilab Function)

nout : integer (number of output)
nin : integer (number of input)
nstate : integer (dimension of state-space)
flag : list made of one character string and one or several integers
sl : list (syslin list)
U square (nstate x nstate) nonsingular matrix

DESCRIPTION:
sl=ssrand(nout,nin,nstate) returns a random strictly proper (D=0) state-space system of size [nout,nint] represented by a syslin list and with nstate state variables.
[sl,U]=ssrand(nout,nin,nstate,flag) returns a test linear system with given properties specified by flag. flag can be one of the following:

flag=list('co',dim_cont_subs)
flag=list('uo',dim_unobs_subs)
flag=list('ncno',dim_cno,dim_ncno,dim_co,dim_nco)
flag=list('st',dim_cont_subs,dim_stab_subs,dim_stab0)
flag=list('dt',dim_inst_unob,dim_instb0,dim_unobs)
flag=list('on',nr,ng,ng0,nv,rk)
flag=list('ui',nw,nwui,nwuis,rk)

The complete description of the Sys is given in the code of the ssrand function (in SCIDIR/macros/util). For example with flag=list('co',dim_cont_subs) a non-controllable system is returned and dim_cont_subs is the dimension of the controllable subspace of Sys. The character strings 'co','uo','ncno','st','dt','on','ui' stand for "controllable", "unobservable", "non-controllable-non-observable", "stabilizable", "detectable", "output-nulling", "unknown-input".

EXAMPLE:
//flag=list('st',dim_cont_subs,dim_stab_subs,dim_stab0)
//dim_cont_subs<dim_stab_subs<dim_stab0
//pair (A,B) U-similar to:
// [*,*,*,*; [*;
// [0,s,*,*; [0;
// [0,0,i,*; B=[0;
// [0,0,0,u] [0]
//
// (A11,B1) controllable s=stable matrix i=neutral matrix u=unstable matrix
[sl,U]=ssrand(2,3,8,list('st',2,5,5));
w=ss2ss(sl,inv(U)); //undo the random change of basis => form as above
[n,nc,u,sl]=st_ility(sl);n,nc

SEE ALSO: syslin 224

1.3.124 stdeviation standard deviation (row or column-wise) of vector/matrix entries

CALLING SEQUENCE:

y=stdeviation(x)
y=stdeviation(x,'r')
y=stdeviation(x,'c')

PARAMETERS:

x : real vector or matrix
**DESCRIPTION:**

`st_deviation` computes the "sample" standard deviation, that is, it is normalized by N-1, where N is the sequence length.

For a vector or a matrix `x`, `y = st_deviation(x)` returns in the scalar `y` the standard deviation of all the entries of `x`.

`y = st_deviation(x,'r')` (or, equivalently, `y = st_deviation(x,1)`) is the rowwise standard deviation. It returns in each entry of the column vector `y` the standard deviation of each row of `x`.

`y = st_deviation(x,'c')` (or, equivalently, `y = st_deviation(x,2)`) is the columnwise standard deviation. It returns in each entry of the row vector `y` the standard deviation of each column of `x`.

**EXAMPLE:**

```plaintext
A = [1, 2, 10; 7, 7.1, 7.01];
st_deviation(A)
st_deviation(A,'r')
st_deviation(A,'c')
```

**SEE ALSO:** `sum`, `median`, `mean` 198

1.3.125 **subf** ________________________________ symbolic subtraction

**CALLING SEQUENCE:**

```plaintext
[c] = subf("a", "b")
```

**PARAMETERS:**

- "a", "b", "c" : strings

**DESCRIPTION:**

returns the character string `c = "a-b"` Trivial simplifications such as `subf("0", "a")` or `subf("1", "2")` are performed.

**EXAMPLE:**

```plaintext
subf('0','a')
subf('2','1')
subf('a','0')
```

**SEE ALSO:** `mulf`, `ldivf`, `rdivf`, `eval`, `evstr` 35

1.3.126 **sum** ______________ sum (row sum, column sum) of vector/matrix entries

**CALLING SEQUENCE:**

```plaintext
y = sum(x)
y = sum(x,'r') or y = sum(x,1)
y = sum(x,'c') or y = sum(x,2)
```

**PARAMETERS:**

- `x` : vector or matrix (real, complex, sparse or polynomial)
- `y` : scalar or vector
DESCRIPTION:
For a vector or a matrix $x$, $y = \text{sum}(x)$ returns in the scalar $y$ the sum of all the entries of $x$.
$y = \text{sum}(x,'r')$ (or, equivalently, $y = \text{sum}(x,1)$) is the rowwise sum. It returns in each entry of the row vector $y$ the sum of the rows of $x$. (The sum is performed on the row indice: $y(j) = \text{sum}(x(i,j), i=1,m)$).
$y = \text{sum}(x,'c')$ (or, equivalently, $y = \text{sum}(x,2)$) is the columnwise sum. It returns in each entry of the column vector $y$ the sum of the columns of $x$. (The sum is performed on the column indice: $y(i) = \text{sum}(x(i,j), j=1,n)$).

EXAMPLE:
```scilab
A=[1,2;3,4];
trace(A)-sum(diag(A))
sum(A,'c')-A*ones(2,1)
sum(A+%i)
A=sparse(A);sum(A,'c')-A*ones(2,1)
s=poly(0,'s');
M=[s,%i+s;s^2,1];
sum(M),sum(M,2)
```

SEE ALSO: cumsum 180, prod 206

1.3.127 sysconv system conversion

CALLING SEQUENCE:
```scilab```
```
[s1,s2]=sysconv(s1,s2)
```
```
PARAMETERS:
s1,s2: list (linear syslin systems)

DESCRIPTION:
Converts $s1$ and $s2$ into common representation in order that system interconnexion operations can be applied. Utility function for experts. The conversion rules in given in the following table.

"c" : continuous time system
"d" : discrete time system
n : sampled system with sampling period n
[] : system with undefined time domain

For mixed systems $s1$ and $s2$ are put in state-space representation.
```
<table>
<thead>
<tr>
<th>s1\s2</th>
<th>&quot;c&quot;</th>
<th>&quot;d&quot;</th>
<th>n2</th>
<th>[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;c&quot;</td>
<td>nothing</td>
<td>incompatible</td>
<td>c2e(s1,n2)</td>
<td>c(s2)</td>
</tr>
<tr>
<td>&quot;d&quot;</td>
<td>incompatible</td>
<td>nothing</td>
<td>e(s1,n2)</td>
<td>d(s2)</td>
</tr>
<tr>
<td>n1</td>
<td>c2e(s2,n1)</td>
<td>e(s2,n1)</td>
<td>n1&lt;&gt;n2</td>
<td>e(s2,n1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>n1=n2 nothing</td>
<td></td>
</tr>
<tr>
<td>[]</td>
<td>c(s1)</td>
<td>d(s1)</td>
<td>e(s1,n2)</td>
<td>nothing</td>
</tr>
</tbody>
</table>
```

With the following meaning:
n1, n2 : sampling period
c2e(s, n) : the continuous-time system s is transformed into a sampled system with sampling period n.
c(s) : conversion to continuous (time domain is "c")
```
syslin Scilab Function

d(s) : conversion to discrete (time domain is "d")
e(s, n) : conversion to samples system with period n

EXAMPLE:

s1=ssrand(1,1,2);
s2=ss2tf(s1);
[s1,s2]=sysconv(s1,s2);

SEE ALSO: syslin 224, ss2tf 363, tf2ss 367

1.3.128 sysdiag ________________ block diagonal system connection

CALLING SEQUENCE:

r=sysdiag(a1,a2,...,an)

DESCRIPTION:
Returns the block-diagonal system made with subsystems put in the main diagonal
ai : subsystems (i.e. gains, or linear systems in state-space or transfer form)
Used in particular for system interconnections.

REMARK:
At most 17 arguments.

EXAMPLES:

s=poly(0,'s')
sydsdiag(rand(2,2),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
sydsdiag(tf2ss(1/s),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])

s=poly(0,'s')
sydsdiag(rand(2,2),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])
sydsdiag(tf2ss(1/s),1/(s+1),[1/(s-1);1/((s-2)*(s-3))])

SEE ALSO: brackets 28, insertion 51, feedback 335

1.3.129 syslin __________________________ linear system definition

CALLING SEQUENCE:

[sl]=syslin(dom,A,B,C [,D [,x0] ])
[sl]=syslin(dom,N,D)
[sl]=syslin(dom,H)

PARAMETERS:

dom : character string (‘c’, ‘d’), or [], or a scalar.
A, B, C, D : matrices of the state-space representation (D optional with default value zero matrix). For
improper systems D is a polynomial matrix.
x0 : vector (initial state; default value is 0)
N, D : polynomial matrices
H : rational matrix or linear state space representation
sl : tlist (”syslin” list) representing the linear system
**DESCRIPTION:**

`systlin` defines a linear system as a list and checks consistency of data.

`dom` specifies the time domain of the system and can have the following values:
- `dom='c'` for a continuous time system,
- `dom='d'` for a discrete time system,
- `n` for a sampled system with sampling period `n` (in seconds).

`dom=[]` if the time domain is undefined.

State-space representation:

```plaintext
sl=systlin(dom,A,B,C [,D [,x0]])
```

represents the system:

\[
\begin{align*}
    \dot{x} &= Ax + Bu \\
    y &= Cx + Du \\
    x(0) &= x0
\end{align*}
\]

The output of `systlin` is a list of the following form: `sl=tlist(['lss','A','B','C','D','X0','dt'],A,B,C,...)`.

Note that `D` is allowed to be a polynomial matrix (improper systems).

Transfer matrix representation:

```plaintext
sl=systlin(dom,N,D)  
sl=systlin(dom,H)
```

The output of `systlin` is a list of the following form:

```plaintext
sl=tlist(['r','num','den','dt'],N,D,dom)
```

or

```plaintext
sl=tlist(['r','num','den','dt'],H(2),H(3),dom)
```

Linear systems defined as `systlin` can be manipulated as usual matrices (concatenation, extraction, transpose, multiplication, etc) both in state-space or transfer representation.

Most of state-space control functions receive a `systlin` list as input instead of the four matrices defining the system.

**EXAMPLES:**

```plaintext
A=[0,1;0,0];B=[1;1];C=[1,1];
S1=systlin('c',A,B,C)  //Linear system definition
S1("A")  //Display of A-matrix
S1("X0") , S1("dt")  // Display of X0 and time domain
s=poly(0,'s');
D=s;
S2=systlin('c',A,B,C,D)
H1=(1+2*s)/s^2, S1bis=systlin('c',H1)
H2=(1+2*s+s^3)/s^2, S2bis=systlin('c',H2)
S1+S2
[S1,S2]
ss2tf(S1)-S1bis
S1bis+S2bis
S1*S2bis
size(S1)
```

**SEE ALSO:**  tlist 73,  lsslist 57,  rlist 69,  ssrand 220,  ss2tf 363,  tf2ss 367,  dscr 333,  abcd 318

---

**1.3.130**  \( \tan \)  \( \text{tangent} \)

**CALLING SEQUENCE:**

```plaintext
[t]=tan(x)
```

Scilab Group  April 1993  225
PARAMETERS:

\( x \) : vector or matrix
\( t \) : vector or matrix

DESCRIPTION:
The elements of \( t \) are the tangent of the elements of \( x \).

EXAMPLE:

\[
\begin{align*}
\text{x} &= \{1, \%i, -1, -\%i\} \\
\text{tan}(\text{x}) &\quad \text{sin}(\text{x}) \div \text{cos}(\text{x}) \\
\end{align*}
\]

SEE ALSO: \( \text{atan} \ 167, \ \text{tanh} \ 227 \)

1.3.131  tanh ------------------------------------------ hyperbolic tangent

CALLING SEQUENCE:

t=tanh(x)

DESCRIPTION:
the elements of \( t \) are the hyperbolic tangents of the elements of \( x \)

EXAMPLE:

\[
\begin{align*}
\text{x} &= \{1, \%i, -1, -\%i\} \\
\text{tanh}(\text{x}) &\quad \text{sinh}(\text{x}) \div \text{cosh}(\text{x}) \\
\end{align*}
\]

SEE ALSO: \( \text{atanh} \ 168, \ \text{tan} \ 225, \ \text{tanhm} \ 226 \)

1.3.132  tanhm ---------------------------------------- matrix hyperbolic tangent

CALLING SEQUENCE:

t=tanhm(x)

PARAMETERS:

\( x, t \) : real or complex square matrix

DESCRIPTION:
tanhm is the matrix hyperbolic tangent of the matrix \( x \).

SEE ALSO: \( \text{tanh} \ 226 \)
### tanm: Matrix Tangent

**Calling Sequence:**

```plaintext
[t] = tanm(x)
```

**Parameters:**

- `x`: square real or complex matrix
- `t`: square matrix

**Description:**

`tanm(x)` is the matrix tangent of the square matrix `x`.

**Example:**

```plaintext
A = [1, 2; 3, 4];
tanm(A)
```

**See Also:** `tan`, `expm`, `sinm`, `atanm`

### toeplitz: Toeplitz Matrix

**Calling Sequence:**

```plaintext
A = toeplitz(c [,r])
```

**Parameters:**

- `a, c, r`: constant, polynomial or character chain matrices

**Description:**

returns the Toeplitz matrix whose first row is `r` and first column is `c`. `c(1)` must be equal to `r(1)`.

`toeplitz(c)` returns the symmetric Toeplitz matrix.

**Example:**

```plaintext
A = toeplitz(1:5);
//
T = toeplitz(1:5, 1:2:7); T1 = [1 3 5 7; 2 1 3 5; 3 2 1 3; 4 3 2 1; 5 4 3 2];
T - T1
//
s = poly(0, 's');
t = toeplitz([s, s+1, s^2, 1-s]);
t1 = [s, 1+s, s*s, 1+s, s, 1+s, s*s, s, 1+s, 1-s, s*s, 1+s, s];
t - t1
//
t = toeplitz(['1', '2', '3', '4']);
t1 = ['1', '2', '3', '4', '2', '1', '2', '3', '2', '1', '2', '4', '3', '2', '1']
```

**See Also:** `matrix`
1.3.135 trfmod  

CALLING SEQUENCE:  

\[ [h_m] = \text{trfmod}(h [, \text{job}]) \]

DESCRIPTION:  
To visualize the pole-zero structure of a SISO transfer function \( h \).

job='p': visualization of polynomials (default)
job='f': visualization of natural frequencies and damping

Interactive simplification of \( h \). \text{trfmod} opens a dialog window.

SEE ALSO: \text{poly 65, simp 497}

1.3.136 trianfml  

CALLING SEQUENCE:  

\[ [f [, \text{sexp}]] = \text{trianfml}(f [, \text{sexp}]) \]

DESCRIPTION:  
Triangularization of the symbolic matrix \( f \); triangularization is performed by elementary row operations;
\text{sexp} is a set of common expressions stored by the algorithm.

EXAMPLE:  
\begin{verbatim}
A=['1','2';'a','b']
W=trianfml([A,string(eye(2,2))])
U=W(:,3:4)
a=5;b=6;
A=evstr(A)
U=evstr(U)
U*A
evstr(W(:,1:2))
\end{verbatim}

SEE ALSO: \text{addf 164, mulf 203, solve 212, trisolve 229}

1.3.137 tril  

CALLING SEQUENCE:  

\[ \text{tril}(x [,k]) \]

PARAMETERS:  
\( x \): matrix (real, complex, polynomial, rational)
\( k \): integer (default value 0)

DESCRIPTION:  
Lower triangle part of a matrix. \text{tril}(x,k) is made by entries below the kth diagonal: \( k \geq 0 \) (upper diagonal) and \( k < 0 \) (diagonals below the main diagonal).

EXAMPLE:  
\begin{verbatim}
s=poly(0,'s');
tril([s,s;s,1])
tril([1/s,1/s;1/s,1])
\end{verbatim}

SEE ALSO: \text{triu 229, ones 204, eye 184, diag 181}
1.3.138  trisolve  

symbolic linear system solver

CALLING SEQUENCE :

\[ x [,sexp] ] = trisolve(A, b [,sexp]) \]

PARAMETERS :

A, b  : matrices of strings

DESCRIPTION :

symbolically solves \( A * x = b \) , \( A \) being assumed to be upper triangular.

sexp  is a vector of common subexpressions in \( A, b, x \).

EXAMPLE :

\[
A=['x','y';'0','z'];b=['0';'1'];
\]

\[
w=trisolve(A,b)\]

\[
x=5; y=2; z=4;\]

\[
evstr(w)\]

\[
inv(evstr(A))*evstr(b)\]

SEE ALSO:  trianfml 228,  solve 212

1.3.139  triu  

upper triangle

DESCRIPTION :

Upper triangle. See tril.

1.3.140  typeof  

object type

CALLING SEQUENCE :

\[ [t]=typeof(object) \]

PARAMETERS :

object  : Scilab object

\[ t \]  : string

DESCRIPTION :

t typeof(object) returns one of the following strings:

"constant" if object is a real or complex constant matrix
"polynomial" if object is a polynomial matrix
"function" if object is a function
"string" if object is a matrix made of character strings
"boolean" if object is a boolean matrix
"list" if object is a list
"rational" if object is a rational matrix (transfer matrix)
"state-space" if object is a state-space model (see syslin)
"sparse" if object is a (real) sparse matrix.
"boolean sparse" if object is a boolean sparse matrix.
EXAMPLE :

typeof(1)
typeof(poly(0,'x'))
typeof(1/poly(0,'x'))
typeof(%t)
w=sprand(100,100,0.001);
typeof(w)
typeof(w==w)
deff('y=f(x)','y=2*x');
typeof(f)

SEE ALSO:  type 74, strings 284, syslin 224, poly 65

### 1.3.141  union  
extract union components of a vector

**CALLING SEQUENCE:**

[v, [ka, kb]]=union(a,b)

**PARAMETERS:**

- `a` : vector of real numbers or strings
- `b` : vector of real numbers or strings
- `v` : row vector of real numbers or strings
- `ka` : row vector of integers
- `kb` : row vector of integers

**DESCRIPTION:**

union(a,b) returns a sorted row vector which retains the unique entries of `[a(:);b(:)]`. 

[v,ka,kb]=union(a,b) also returns index vectors ka and kb such that v is a sorted combination of the entries a(ka) and b(kb).

**EXAMPLE** :

A=round(5*rand(10,1));
B=round(5*rand(7,1));

union(A,B)
[N,ka,kb]=union(A,B)

union('a'+string(A),'b'+string(B))

SEE ALSO:  unique 230, sort 213

### 1.3.142  unique  
extract unique components of a vector

**CALLING SEQUENCE:**

[N, [k]]=unique(M)

**PARAMETERS:**

Scilab Group  April 1999
zeros: vector of real numbers or strings
N: vector of real numbers or strings
k: vector of integers

**DESCRIPTION:**
unique(M) returns a vector which retains the unique entries of M in ascending order. If required the output argument k contains the position of the first encountered unique entries.

**EXAMPLE:**

```scilab
M=round(2*rand(20,1));

unique(M)
[N,k]=unique(M)

unique(string(M))
[N,k]=unique(string(M))
```

**SEE ALSO:** union 230, sort 213, lex_sort 194

### 1.3.143 zeros

**matrix made of zeros**

**CALLING SEQUENCE:**

```
y=zeros()
y=zeros(x)
y=zeros(m1,m2,..)
```

**PARAMETERS:**

x, y: matrices
m1, m2,..: integers

**DESCRIPTION:**
Creates matrix of zeros (same as 0*ones).

```
zeros(m1,m2) : for an (m1,m2) matrix.
zeros(m1,m2,..,mn) : creates a (m1,m2,..,mn) matrix filled with zeros
zeros(A) : for a matrix of same size of A.
zeros(3) : is zeros(a) with a=3 i.e it is NOT a 3x3 matrix!
zeros() : returns a single zero
```

If x is a syslin list (linear system in state-space or transfer form), zeros(x) is also valid and returns a zero matrix.

**EXAMPLE:**

```
zeros(3)
zeros(3,3)
zeros(2,3,2)
```

**SEE ALSO:** eye 184, ones 204, spzeros 218
1.4 Input/Output functions
1.4.1 diary  
diary of session  

CALLING SEQUENCE:

diary('file-name')

DESCRIPTION:
diary creates a file which contains a copy of the current Scilab session. diary(0) interrupts the diary.

SEE ALSO: exec 36, unix 310

1.4.2 disp  
displays variables  

CALLING SEQUENCE:

disp(xi,[x2,...xn])

DESCRIPTION:
disp displays xi with the current format. xi's are arbitrary objects (matrices of constants, strings, functions, lists, ...)
Display of objects defined by tlist may be overloaded by the definition of a function. This function must have no output argument a single input argument ant it's name is formed as follow $%$<tlist_type> $ where $%$<tlist_type> stands for the first entry of the tlist type component.

SEE ALSO: write 262, read 254, print 251, string 284, tlist 73

EXAMPLES:

disp([1 2],3)
deff('[]=t_p(l)').'disp(l(3),l(2))')
disp(tlist(‘t’,1,2))

1.4.3 dispfile  
display opened files properties  

CALLING SEQUENCE:

dispfiles([units])

PARAMETERS:

units : a vector of numbers, the file's logical units. By default all opened files.

DESCRIPTION:
dispfiles displays properties of currently opened files.

EXAMPLE:

dispfiles()

SEE ALSO: file 234, mopen 244

AUTHOR: S. Steer
1.4.4 file

CALLING SEQUENCE:

[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format])
file(action,unit)
[units [,typ [,nams [,mod [,swap]]] ] ] = file([unit])

PARAMETERS:

file-name : string, file name of the file to be opened
status : string, The status of the file to be opened
"new" : file must not exist new file (default)
"old" : file must already exists.
"unknown" : unknown status
"scratch" : file is to be deleted at end of session
access : string, The type of access to the file
"sequential" : sequential access (default)
"direct" : direct access.
format : string,
"formatted" : for a formatted file (default)
"unformatted" : binary record.
recl : integer, is the size of records in bytes when access="direct"
unit : integer, logical unit descriptor of the opened file
units : integer vector, logical unit descriptor of the opened files. Units 1 5 and 6 are reserved by the
system for history file, input and output devices.
typs : Character string vector, type (C or Fortran) of opened files.
nams : Character string vector, pathnames of opened files.
mod : file opening mode. Formed by three digits abc
Fortran fields stands for formatted and 1 for unformatted (binary)
b : 0 stands for sequential acces and 1 for direct access
b : 0 stands for "new", 1 for "old", 2 for "scratch" and 3 for "unknown"
C files are 1 if file has been opened with a "b" (binary) mode
b : is 1 if file has been opened with a "t" (upating) mode
b : 1 stands for "r" (read), 2 stands for "w" (write) and 3 for "a" (append)
swap : automatic swap switch. swap=1 if automatic swap is on. swap is always 0 for Fortran files.
err : integer, error message number (see error), if open fails. If err is omitted an error message is issued.
action : is one of the following strings:
"close" : closes the file(s) given by the logical unit descriptors given in units
"rewind" : puts the pointer at beginning of file
"backspace" : puts the pointer at beginning of last record.
"last" : puts the pointer after last record.

DESCRIPTION:

selects a logical unit unit and manages the file file-name.
[unit [,err]]=file('open', file-name [,status] [,access [,recl]] [,format])
allows to open a file with specified properties and to get the associated unit number unit. This unit number
may be used for further actions on this file or as file descriptor in read, write, readb, writb, save, load function calls.
file(action,unit) allows to close the file, or move the current file pointer.
file() returns the logical unit descriptors of the opened files. So file('close',file() ) closes
all user opened files (C or Fortran type).

EXAMPLE:

Scilab Group

April 1993

234
fprintf Scilab Function

```matlab
u=file('open',TMPDIR+'/foo','unknown')
for k=1:4
    a=rand(1,4)
    write(u,a)
end
file('rewind',u)
x=read(u,2,4)
file('close',u)
```

```
//
ui=file('open',TMPDIR+'/foo','unknown')
u2=mopen(TMPDIR+'/foo1','wb')
[unit,typs,nams]=file()
```

See Also: save 258, load 239, write 262, read 254, writb 261, readb 255, xgetfile 263, mopen 244, mclose 240

1.4.5  `fileinfo`  Provides information about a file

**Calling Sequence:**

```
[x,ierr]=fileinfo(file)
```

**Parameters:**

- `file`: a character string, the file pathname
- `x`: an integer vector of size 6 containing information or an empty matrix if file does not exist.
- `ierr`: error indicator, 0, if no error has occurred

**Description:**

`x=fileinfo(file)` returns

- `x(1)`: The file size
- `x(2)`: The file mode
- `x(3)`: The user id
- `x(4)`: the group id
- `x(5)`: The device number
- `x(6)`: The date of last modification

**Examples:**

```matlab
w=fileinfo(SCI+'/scilab.star')
getdate(w(6))
```

See Also: getdate 314, file 234, dispfiles ??, newest 250

1.4.6  `fprintf`  Emulator of C language fprintf function

**Calling Sequence:**

```
fprintf(file,format,value_1,..,value_n)
```

**Parameters:**

Scilab Group  May 1994  235
format : a Scilab string. Specifies a character string combining literal characters with conversion specifications.
value : Specifies the data to be converted according to the format parameter.
str : column vector of character strings
file : a Scilab string specifying a file name or a logical unit number (see file)

DESCRIPTION :
The fprintf function converts, formats, and writes its value parameters, under control of the format parameter, to the file specified by its file parameter.

The format parameter is a character string that contains two types of objects:

Literal characters : which are copied to the output stream.
Conversion specifications : each of which causes zero or more items to be fetched from the value parameter list. See printf_conversion for details

If any values remain after the entire format has been processed, they are ignored.

EXAMPLES :

u=file('open','results','unknown') //open the result file
t=0:0.1:2*%pi;
for tk=t
    fprintf(u,'time = %6.3f value = %6.3f',tk,sin(tk)) // write a line
end
file('close',u) //close the result file

SEE ALSO : string284, print251, write262, format42, disp233, file234, printf252, sprintf260

1.4.7 fprintfMat ---------------------------------------- print a matrix in a file.

CALLING SEQUENCE :

fprintfMat(fil,M,format)

PARAMETERS :

fil : a string, path of the file
format : a character string, a C like format.
M : A matrix of real numbers.

DESCRIPTION :

The fprintfMat function prints a matrix in a formated file. Each row of the matrix give a line in the file.

EXAMPLE :

n=50;
a=rand(n,n,'u');
fprintfMat(TMPDIR+'/Mat',a,'%5.2f');
a1=fscanfMat(TMPDIR+'/Mat');

SEE ALSO : mclose240, meof241, mfprintf245, mfscanf248, fscanfMat237, mget242, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell

Scilab Group May 1994 236
1.4.8 \texttt{fscanf} \hspace{1cm} \textit{Converts formatted input read on a file}

**CALLING SEQUENCE:**

\[ [v_1, \ldots, v_n] = \texttt{fscanf} \ (\text{file}, \text{format}) \]

**PARAMETERS:**

- \texttt{format}: Specifies the format conversion.
- \texttt{file}: Specifies the input file name or file number.

**DESCRIPTION:**

The \texttt{fscanf} functions read character data on the file specified by the \texttt{file} argument, interpret it according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input. The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except \% (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see \texttt{scanf} \texttt{conversion} for details.

**SEE ALSO:** \texttt{printf} 252, \texttt{read} 254, \texttt{scanf} 259, \texttt{sscanf} 260

1.4.9 \texttt{fscanfMat} \hspace{1cm} \textit{Reads a Matrix from a text file.}

**CALLING SEQUENCE:**

\[ M = \texttt{fscanfMat}(\text{filename}); \]

**PARAMETERS:**

- \texttt{filename}: a character string giving the name of the file to be scanned.
- \texttt{M}: Output variable. A matrix of real numbers.

**DESCRIPTION:**

The \texttt{fscanfMat} function is used to read a scalar matrix from a text file. The first non-numeric lines of the file are ignored and all the remaining lines must have the same number of columns (column separator are assumed to be white spaces or tab characters). The number of columns of the matrix will follow the number of columns found in the file and the number of lines is fetched by detecting e0f in the input file. This function can be used to read back numerical data saved with the \texttt{fprintfMat}.

**EXAMPLE:**

```matlab
fd=mopen(TMPDIR+’/Mat’,’w’);
mfprintf(fd,’Some text.....\n’);
mfprintf(fd,’Some text again\n’);
a=rand(6,6);
for i=1:6 ,
  for j=1:6, mfprintf(fd,’%5.2f ’,a(i,j));end;
mfprintf(fd,’\n’);
end
mclose(fd); 
A1=fscanfMat(TMPDIR+’/Mat’) 
```

**SEE ALSO:** \texttt{mclose} 240, \texttt{meof} 241, \texttt{mfprintf} 245, \texttt{fprintfMat} 236, \texttt{mfscanf} 248, \texttt{fscanfMat} 237, \texttt{mget} 242, \texttt{mgetstr} 244, \texttt{mopen} 244, \texttt{mprintf} 245, \texttt{mput} 246, \texttt{mputstr} 247, \texttt{mscanf} 248, \texttt{mseek} 248, \texttt{mtell} 249

Scilab Group \hspace{1cm} December 2000
1.4.10 getio .......................... get Scilab input/output logical units

CALLING SEQUENCE:

ios=getio()

PARAMETERS:

ios : a vector [rio rte wio wte]
rio : current logical unit for reading instructions
rte : logical unit assigned for input in main scilab window
wio : logical unit relative to the diary file if any. wio=0 stands for no diary file opened
wte : logical unit assigned for output in main scilab window

DESCRIPTION:

getio returns logical units assigned for main scilab input and output

SEE ALSO: %io ??, file 234, exec 36

1.4.11 input ............................. prompt for user input

CALLING SEQUENCE:

[x]=input(message, ["string"])

PARAMETERS:

message : character string
"string" : the character string "string" (may be abbreviated to "s")
x : real number (or character string if "string" is in the calling sequence)

DESCRIPTION:

input(message) gives the user the prompt in the text string and then waits for input from the keyboard. The input can be expression which is evaluated by evstr. Invoked with two arguments, the output is a character string which is the expression entered at keyboard.

EXAMPLE:

//x=input("How many iterations?")
//x=input("What is your name?","string")

SEE ALSO: file 234, read 254, write 262, evstr 35, x_dialog 292, x_mdiallog 293

1.4.12 lines .............................. rows and columns used for display

CALLING SEQUENCE:

lines([nl [,nc]])
nlc=lines()

PARAMETERS:

nl : an integer, the number of lines for vertical paging control. If 0 no vertical paging control is done.
nc : an integer, the number of column of output. Used for formatting output
nlc : a 1x2 vector [nl,nc]

DESCRIPTION :
lines handles Scilab display paging.
   lines() returns the vector [# columns, # rows] currently used by Scilab for displaying the results.
lines(nl) sets the number of displayed lines (before user is asked for more) to nl.
lines(0) disables vertical paging
lines(nl,nc) changes also the size of the output to nc columns.
   When Scilab is launched without -nw option, the lines parameters are automatically set according
to the output window size.
SEE ALSO: disp 233, print 251

1.4.13  load  load saved variable

CALLING SEQUENCE :
load(filename [,x1,...,xn])
load(fd [,x1,...,xn])

PARAMETERS :
filename : character string containing the path of the file
fd : a file descriptor given by a call to mopen
xi : arbitrary Scilab variable name(s) given as strings.

DESCRIPTION :
The load command can be used to reload in the Scilab session variables previously saved in a file with
the save command.
load(filename) loads the variables saved in file given by its path filename.
load(fd) loads the variables saved in file given by its descriptor fd.
load(filename,'x','y') or load(fd,'x','y') loads only variables x,y.

COMPATIBILITY :
Even if the binary file format has changed with 2.5 version, load(filename,...) is able to read
old format files. Previous file format can be accessed for a while using function oldsave ans oldload.

EXAMPLES :
a=eye(2,2);b=ones(a);
save('vals.dat',a,b);
clear a
clear b
load('vals.dat','a','b');

SEE ALSO: save 258, getf 272, mopen 244

1.4.14  manedit  editing a manual item

CALLING SEQUENCE :
manedit(manitem ,[editor])

PARAMETERS :
**manitem** : character string (usually, name of a function)
**editor** : character string

**DESCRIPTION :**
`edit(manitem ,[editor])` opens the file `manitem` in the editor given by `editor`. Default editor is Emacs. This function should be customized according to your needs.

**EXAMPLE :**

```plaintext
//manedit('lqg')
```

**SEE ALSO :** `whereis 77`, `edit 267`

### 1.4.15 mclearerr ................................. reset binary file access errors

**CALLING SEQUENCE :**

```plaintext
mclearerr([fd])
```

**PARAMETERS :**

- `fd` : scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.

**DESCRIPTION :**
The function `clearerr` is used to resets the error indicator and EOF indicator to zero.

**SEE ALSO :** `mclose 240`, `mopen 244`, `mput 246`, `mget 242`, `mgetstr 244`, `mputstr 247`, `meof 241`, `mseek 248`, `mtell 249`, `?? file, read, write, save, load`

### 1.4.16 mclose ..................................... close an opened file

**CALLING SEQUENCE :**

```plaintext
err=mclose([fd])
mclose('all')
```

**PARAMETERS :**

- `fd` : scalar. The `fd` parameter returned by the function `mopen` is used as a file descriptor (it’s a positive integer).
- `err` : a scalar. Error indicator : vector

**DESCRIPTION :**
`mclose` must be used to close a file opened by `mopen`. If `fd` is omitted `mclose` closes the last opened file. `mclose('all')` closes all files opened by `file('open',..)` or `mopen`

**SEE ALSO :** `meof 241`, `mfprintf 245`, `fprintfMat 236`, `mfscanf 248`, `fscanfMat 237`, `mget 242`, `mgetstr 244`, `mopen 244`, `?? mprintf, mput, mputstr, mscanf, mseek, mtell, file`
1.4.17  meof  ________________  check if end of file has been reached

CALLING SEQUENCE :

err=meof(fd)

PARAMETERS :

fd : scalar. The fd parameter returned by the function mopen. -1 stands for last opened file. Default value is -1.
err : scalar. Error indicator

DESCRIPTION :

The function meof will return a non null value if end of file has been reached in a previous call to mget or mgetstr. The function clearerr is used to reset the error flag and EOF flag to zero.

SEE ALSO : mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell

1.4.18  mscanf  ________________  interface to the C scanf function

mscanf - interface to the C fscanf function
msscanf - interface to the C sscanf function

CALLING SEQUENCE :

[n,v_1,...v_n]=mfscanf(fd,format)
L=mfscanf(fd,format)

[n,v_1,...v_n]=mscanf(format)
L=mscanf(format)

[n,v_1,...v_m]=msscanf(format,str)
L=msscanf(format)

PARAMETERS :

format : a Scilab string describing the format to use to write the remaining operands. The format operand follows, as close as possible, the C printf format operand syntax.
fd : The fd parameter returned by the function mopen is used as a file descriptor (it’s a positive integer). When specifying the fd parameter, the value -1 refers to the default file (i.e. the last opened file).
str : a Scilab string.
n : an integer, the number of data read or -1 if EOL has been encountered before any datum has been read.
v_i : Each function reads characters, interprets them according to a format, and stores the results in its output arguments. If more than $n$ output arguments are provided, the last ones $v_{n+1},...v_m$ are set to empty matrices.
L : a matrix of strings or numbers if data read are homogeneous or an mlist of type (cblock) containing a sequence of homogeneous matrices

DESCRIPTION :

The mfscanf function reads characters from the stream fd.
The mscanf function reads characters from Scilab window.
The msscanf function reads characters from the Scilab string str.

EXAMPLES :
s='1 1.3'
[n,a,b]=msscanf(s,"%i %e")
msscanf(s,"%i %e")

msscanf(" 1\n",'%c%c%c%c') //scan characters
msscanf('0xabc','%x') //scan with hexadecimal format
msscanf('012345abczoo','[%0-9abc]s') //[] notation

//create a file with data
u=mopen(TMPDIR+'/foo','w');
t=0.5;mfprintf(u,"%6.3f %6.3f\n",t,sin(t))
t=0.6;mfprintf(u,"%6.3f %6.3f\n",t,sin(t))
mclose(u);
//read the file
u=mopen(TMPDIR+'/foo','r');
[n,a,b]=mfscanf(u,'%e %e')
l=mfscanf(u,'%e %e')
mclose(u);

SEE ALSO: mclose, meof, mfprintf, fprintfMat, mscanf, fscanfMat, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell

1.4.19 mget — reads byte or word in a given binary format and convert to double
mgeti - reads byte or word in a given binary format return an int type

CALLING SEQUENCE:

x=mget([n,type,fd])
x=mgeti([n,type,fd])

PARAMETERS:

n :a positive scalar: The number of items to be read.
f d : scalar. The fd parameter returned by the function mopen. -1 stands for last opened file. Default value is -1.
type : a string. Give the binary format used to write all the entries of x.
x : a vector of floating point or integer type numbers

DESCRIPTION:
The mget function reads data in the input specified by the stream parameter fd and returns a vector of floating point data. The mgeti function reads data in the input specified by the stream parameter fd and returns a vector of integer data.

Data is read at the position at which the file pointer is currently pointing and advances the indicator appropriately.
The type parameter is a conversion specifier which may be set to any of the following flag characters (with default value "l"):
"l", "i", "s", "ul", "ui", "us", "d", "f", "e", "c", "uc" : for reading respectively a long, an int, a short, an unsigned long, an unsigned int, an unsigned short, a double, a float, a char and an unsigned char. The bytes which are read are automatically swapped if necessary (by checking little-endian status). This default swapping mode can be suppressed by adding a flag in the mopen function. Format "l", "d" and "f" are valid only with the mget function.
"..l" or ".b": It is also possible to read in little-endian or big-endian mode by adding a 'l' or 'b' character at the end of a type specification. For example "db" will read a double in big-endian mode.

**EXAMPLE:**

```plaintext
cfile1 = 'test1.bin';
cfile2 = 'test2.bin';
cfd1=mopen(cfile1,'wb');
cfd2=mopen(cfile2,'wb');
mput(1996,'ull',cfd1);
mput(1996,'ull',cfd2);
cmclose(cfd1);
cmclose(cfd2);
cfd1=mopen(cfile1,'rb');
if 1996<>mget(1,'ull',cfd1);write(%io(2),'Bug');end;
cfd2=mopen(cfile2,'rb');
if 1996<>mget(1,'ull',cfd2);write(%io(2),'Bug');end;
cmclose(cfd1);
cmclose(cfd2);

See Also: mclose 240, meof 241, mfprintf 245, fprintfMat 236, mfscanf 248, fscanfMat 237, mget 242, mgetstr 244, mopen 244, mprintf 245, mput 246, mputstr 247, mscanf 248, mseek 248, mtell 249
```

### 1.4.20 mgetl----------------------------------------------- read lines from an ascii file

**CALLING SEQUENCE:**

```plaintext
txt=mgetl(file_desc [,m])
```

**PARAMETERS:**

- `file_desc`: a character string giving the file name or a logical unit returned by mopen
- `m`: an integer scalar. Default value is -1.
- `txt`: a column vector of string

**DESCRIPTION:**

The `mgetl` function allows to read a lines from an ascii file.

If `m` is omitted or is -1 all lines till end of file occurs are read.

If `m` is given `mgetl` tries to read exactly `m` lines, if an end of file occurs before `m` lines are read an error is issued. This option is useful to sequentially read part of a file as follow `mgetl` allows to read files coming from Unix, Windows, or Mac operating systems.

**EXAMPLE:**

```plaintext
mgetl('SCI/scilab.star',5)
mgetl SCI/macros/elema/and.sci

fd=mopen('SCI/scilab.star','r')
mgetl(fd,10)
cmclose(fd)
```

See Also: mputl 247, mclosel 240, mfsclang 248, mget 242, mgetstr 244, mopen 244, read 254

**AUTHOR:** S. Steer

Scilab Group April 1993 243
### 1.4.21 mgetstr  read a character string

**CALLING SEQUENCE:**

```plaintext
str=mgetstr([n,fd])
```

**PARAMETERS:**

- `n`: a positive scalar. The number of character to read.
- `fd`: scalar. The `fd` parameter returned by the function `mopen`. -1 stands for last opened file. Default value is -1.
- `str`: a character string

**DESCRIPTION:**

The `mgetstr` function allows to read a character string in a binary file. If EOF is reached before read completion only the properly read values will be returned.

**SEE ALSO:** `mclose`, `meof`, `mprintf`, `fprintfMat`, `mfscanf`, `fscanfMat`, `mget`, `mgetstr`, `mopen`, `mprintf`, `mput`, `mputstr`, `mscanf`, `mseek`, `mtell`

### 1.4.22 mopen open a file

**CALLING SEQUENCE:**

```plaintext
[fd,err]=mopen(file [, mode, swap ])
```

**PARAMETERS:**

- `file`: a character string. The pathname of the file to open.
- `mode`: a character string that controls whether the file is opened for reading (r), writing (w), or appending (a) and whether the file is opened for updating (+). The mode can also include a b parameter to indicate a binary file.
- `swap`: a scalar. If `swap` is present and `swap=0` then automatic bytes swap is disabled.
- `err`: a scalar. Error indicator
- `fd`: scalar. The `fd` parameter returned by the function `mopen` is used as a file descriptor (it’s a positive integer).

**DESCRIPTION:**

The `mopen` function may be used to open a file in a way compatible with the C `fopen` procedure. Without swap argument the file is supposed to be coded in "little endian IEEE format” and data are swaped if necessary to match the IEEE format of the processor.

The mode parameter controls the access allowed to the stream. The parameter can have one of the following values. In this list of values, the b character indicates a binary file:

- `r` or `rb`: Opens the file for reading.
- `w` or `wb`: Creates a new file for writing, or opens and truncates a file to zero length.
- `a` or `ab`: Appends (opens a file for writing at the end of the file, or creates a file for writing).
- `r+` or `r+b`: Opens a file for update (reading and writing).
- `w+` or `w+b`: Truncates to zero length or creates a file for update.
- `a+` or `a+b`: Appends (opens a file for update, writing at the end of the file, or creates a file for writing).

When you open a file for update, you can perform both input and output operations on the resulting stream. However, an output operation cannot be directly followed by an input operation without a file-positioning operation (mseek() function). Also, an input operation cannot be directly followed...
by an output operation without an intervening file positioning operation, unless the input operation
encounters the end of the file.

When you open a file for append (that is, when the mode parameter is a or a+), it is impossible to
overwrite information already in the file. You can use the fseek() function to reposition the file pointer
to any position in the file, but when output is written to the file, the current file pointer is ignored. All
output is written at the end of the file and the file pointer is repositioned to the end of the output.

To open files in a way compatible with Fortran like functions use function file.

SEE ALSO: mclose 240, meof 241, mfprintf 245, fprintfMat 236, mfscanf 248,
fscanfMat 237, mget 242, mgetstr 244, mopen 244, mprintf 245, mput 246, mputstr
247, mscanf 248, mseek 248, mtell 249

1.4.23 mfprintf _______________ converts, formats, and writes data to a file

mfprintf - converts, formats, and writes data to the main scilab window
msprintf - converts, formats, and writes data in a string

CALLING SEQUENCE :

mfprintf(fd, format, a1, ..., an);
mprintf(format, a1, ..., an);
str=msprintf(format, a1, ..., an);

PARAMETERS :

fd : scalar, file descriptor given by mopen (it’s a positive integer). The value -1 refers to the default
file (i.e the last opened file).
format : a Scilab string describing the format to use to write the remaining operands. The format operand
follows, as close as possible, the C printf format operand syntax.
str : a character string, string to be scanned.
a1, ..., an : Specifies the data to be converted and printed according to the format parameter.

DESCRIPTION :
The mfprintf, mfprintf, and msprintf functions are interface for C-coded version of printf,
fprintf and sprintf functions.
The mfprintf function writes formatted operands to the standard Scilab output (i.e the Scilab window).
The argument operands are formatted under control of the format operand.
The mfprintf function writes formatted operands to the file specified by the file descriptor fd. The
argument operands are formatted under control of the format operand.
The msprintf writes formatted operands in its returned value (a Scilab string). The argument operands
are formatted under control of the format operand. Note that, in this case, the escape sequences (", ,...")
are treated as a normal sequence of characters.
All these functions may be used to output column vectors of numbers and string vectors without an explicit
loop on the elements. In that case these functions iterates on the rows. The shortest vector gives the number
of time the format has to be iterated.
An homogeneous sequence of identical type parameters can be replaced by a matrix

EXAMPLE :

mfprintf(’At iteration %i, Result is:\nalpha=\n%f’,33,0.535)

msprintf(’%5.3f %5.3f’,123,0.732)
msprintf(’%5.3f\n%5.3f’,123,0.732)
A=rand(5,2);
// vectorized forms: the format directive needs
// two operand, each column of A is used as an operand.
// and the mprintf function is applied on each row of A
mprintf('%5.3f\t%5.3f\n',A)

colors=['red';'green';'blue';'pink';'black'];
RGB=[1 0 0;0 1 0;1 0.75 0.75;0 0 0];
mprintf('%d\t%s\t%f\t%f\t%f\n',(1:5)',colors,RGB)

SEE ALSO: mclose 240, meof 241, mfprintf 245, fprintfMat 236, mfscanf 248, fscanfMat 237, mget 242, mgetstr 244, mopen 244, mprintf 245, mput 246, mputstr 247, mscanf 248, mseek 248, mtell 249

1.4.24 mput ________________ writes byte or word in a given binary format

CALLING SEQUENCE :

mput(x [,type,fd])

PARAMETERS :

x : a vector of floating point or integer type numbers
fd : scalar. The fd parameter returned by the function. Default value is -1 which stands for the last
(mopen) opened file.
type : a string. Give the binary format used to write all the entries of x.

DESCRIPTION :

The mput function writes data to the output specified by the stream parameter fd. Data is written at the
position at which the file pointer is currently pointing and advances the indicator appropriately.
The type parameter is a conversion specifier which may be set to any of the following flag characters (with
default value 'l'):

"l", "i", "s", "ul", "ui", "us", "d", "f", "c", "uc" : for writing respectively a long, an int, a
short, an unsigned long, an unsigned int, an unsigned short, a double, a float, a char and an unsigned
char. The bytes which are wrote are automatically swapped if necessary (by checking little-endian
status) in order to produce machine independent binary files (in little-endian mode). This default
swapping mode can be suppressed by adding a flag in the mopen function.
"..l" or "..b" : It is also possible to write in little-endian or big-endian mode by adding a 'l' or
'b' character at the end of a type specification. For example "db" will write a double in big-endian
mode.

EXAMPLE :

filen = 'test.bin';
mopen(filen,'wb');
mput(1996,'l');mput(1996,'i');mput(1996,'s');mput(98,'c');
// force little-endian
mput(1996,'ll');mput(1996,'il');mput(1996,'sl');mput(98,'cl');
// force big-endian
mput(1996,'lb');mput(1996,'ib');mput(1996,'sb');mput(98,'cb');
//
mclose();
mopen(filen,'rb');
if 1996<>mget(1,'l') then pause,end
if 1996<>mget(1,'i') then pause,end
if 1996<>mget(1,'s') then pause,end
if 98<>mget(1,’c’) then pause,end
// force little-endian
if 1996<>mget(1,’ll’) then pause,end
if 1996<>mget(1,’il’) then pause,end
if 1996<>mget(1,’sl’) then pause,end
if 98<>mget(1,’cl’) then pause,end
// force big-endian
if 1996<>mget(1,’lb’) then pause,end
if 1996<>mget(1,’ib’) then pause,end
if 1996<>mget(1,’sb’) then pause,end
if 98<>mget(1,’cb’) then pause,end
//
mclose();

SEE ALSO: mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr, mscanf, mseek, mtell

1.4.25 mputl ___________________________ writes strings in an ascii file

CALLING SEQUENCE:

mputl(txt [,file_desc])

PARAMETERS:

file_desc : a character string giving the file name or a logical unit returned by mopen. If omitted lines
are written in the last file opened by mopen.
txt : a vector of strings.

DESCRIPTION:

mputl function allows to write a vector of strings as a sequence of lines in an ascii file.

SEE ALSO: mputl, mclose, meof, mfprintf, fprintfMat, mput, mputstr, mopen, mputstr, mput, mputstr,

1.4.26 mputstr __________________________ write a character string in a file

CALLING SEQUENCE:

mputstr(str [, fd]);

PARAMETERS:

fd : scalar. The fd parameter returned by the function mopen. -1 stands for last opened file. Default
value is -1.
str : a character string

DESCRIPTION:

mputstr function allows to write a character string in a binary file.

SEE ALSO: mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr, mopen, mprintf, mput, mputstr,
mseek, mtell

Scilab Group Aug 1999 247
1.4.27 mscanf

mscanf - scan data from input
msscanf - scan data from string

CALLING SEQUENCE:

\[ [n, a1, \ldots, am] = mfscanf(fd, format); \]
\[ data = mfscanf(fd, format); \]
\[ [n, a1, \ldots, am] = mscanf(format); \]
\[ data = mscanf(format); \]
\[ [n, a1, \ldots, am] = msscanf(str, format); \]
\[ data = msscanf(str, format); \]

PARAMETERS:

\[ fd \] : scalar, file descriptor given by \texttt{mopen} (it’s a positive integer). The value \(-1\) refers to the default file (i.e, the last opened file).
\[ format \] : a character string, a C like format.
\[ str \] : a character string, string to be scanned.
\[ n \] : a scalar integer, the number or data really read.
\[ a1, \ldots, am \] : Output variables. if \( m > n \) the \( n+1:n \) last \( ai \) are set to \([]\).
\[ data \] : a list formed by the data really read.

DESCRIPTION:

The mscanf(), mfscanf(), and msscanf() functions are interface for C-coded version of scanf, sscanf and scanf functions.

The mscanf(), mfscanf(), and msscanf() functions read character data, interpret it according to a format (see \texttt{cformat}), and store the converted results into variables. The format parameter contains conversion specifications used to interpret the input.

These functions read their input from the following sources:

\texttt{mscanf()} : Reads from the Scilab input.
\texttt{mfscanf()} : Reads from the file given by the file descriptor \( fd \).
\texttt{msscanf()} : Reads from the character string specified by the \texttt{str} parameter.

EXAMPLE:

\[ [n, a1, a2] = msscanf('123 456', '\%i \%s') \]
\[ [n, a1, a2, a3] = msscanf('123 456', '\%i \%s') \]
\[ data = msscanf('123 456', '\%i \%s') \]

\[ fd = \texttt{mopen} (\texttt{SCI}+/\texttt{scilab.star}', \texttt{'r'}) \]
\[ mfscanf(fd, '\%s \%s \%s') \]
\[ \texttt{mclose}(fd) \]

SEE ALSO: \texttt{mclose}, \texttt{meof}, \texttt{mfprintf}, \texttt{fprintfMat}, \texttt{mfscanf}, \texttt{fscanfMat}, \texttt{mget}, \texttt{mgetstr}, \texttt{mopen}, \texttt{mprintf}, \texttt{mput}, \texttt{mputstr}, \texttt{mseek}, \texttt{mtell}

1.4.28 mseek

set current position in binary file.

CALLING SEQUENCE:
mseek(n [,fd, flag])

PARAMETERS:

n : a positive scalar: The offset from origin in number of bytes.
fd : scalar. The fd parameter returned by the function mopen. -1 stands for last opened file. Default value is -1.
flag : a string. specifies the origin. Default value 'set'.

DESCRIPTION:
The function mseek() sets the position of the next input or output operation on the stream fd. The new position is at the signed distance given by n bytes from the beginning, from the current position, or from the end of the file, according to the flag value which can be 'set', 'cur' or 'end'.
mseek() allows the file position indicator to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap will return zero until data is actually written into the gap. mseek(), by itself, does not extend the size of the file.

EXAMPLE:

```scilab
file3='test3.bin'
fd1= mopen(file3,'wb');
for i=1:10, mput(i,'d'); end
mseek(0);
mput(678,'d');
mseek(0,fd1,'end');
mput(932,'d');
mclose(fd1)
fd1= mopen(file3,'rb');
res=mget(fd1);
res1=[1:11]; res1(1)=678; res1($)=932;
if res1<>res ;write(%io(2),'Bug');end;
mseek(0,fd1,'set');
// trying to read more than stored data
res1=mget(100,'d',fd1);
if res1<>res ;write(%io(2),'Bug');end;
meof(fd1)
mclearerr(fd1)
mclose(fd1);
```

SEE ALSO: mclose 240, meof 241, mfprintf 245, fprintfMat 236, mfscanf 248, fscanfMat 237, mget 242, mgetstr 244, mopen 244, mprintf 245, mput 246, mputstr 248, mscanf 248, mseek 248, mtell 249

1.4.29 mtell ____________________________________________ binary file management

CALLING SEQUENCE:

mtell([fd])

PARAMETERS:

fd : scalar. The fd parameter returned by the function mopen. -1 stands for last opened file. Default value is -1.

DESCRIPTION:
The function mtell() returns the offset of the current byte relative to the beginning of the file associated with the named stream fd.

SEE ALSO: mclose 240, meof 241, mfprintf 245, fprintfMat 236, mfscanf 248, fscanfMat 237, mget 242, mgetstr 244, mopen 244, mprintf 245, mput 246, mputstr 248, mscanf 248, mseek 248, mtell 249
1.4.30 newest ____________________________ returns newest file of a set of files

CALLING SEQUENCE :

k=newest (paths)
k=newest (path1,path2,...,pathn)

PARAMETERS :

k : the index of the newest file
paths : a character string vector (or list), paths(i) is the pathname of ith file
pathi : a character string, the pathname of ith file

DESCRIPTION :

Given a set of pathnames newest returns the index of the newest one. Non existant files are supposed to be the oldest.

EXAMPLE :

newest(’SCI/macros/xdess/bode.sci’,’SCI/macros/xdess/bode.bin’)
newest(’SCI/macros/xdess/bode.’+['sci','bin'])

SEE ALSO:  fileinfo 235

1.4.31 oldload ____________ load saved variable in 2.4.1 and previous formats

CALLING SEQUENCE :

oldload(’file-name’ [,x1,...,xn])

PARAMETERS :

file-name : character string
xi : arbitrary Scilab variable name(s) given as strings.

DESCRIPTION :

The oldload function is obsolete and is retained only for compatibility purpose.
The oldload command can be used to reload in the Scilab session variables previously saved in a file with the save command.
oldload(’file-name’) loads the variables saved in file ’file-name’.
oldload(’file-name’,’x’,’y’,’z’) loads only variables x,y,...,z stored in file ’file-name’.

EXAMPLES :

a=eye(2,2);b=ones(a);
oldsave(’TMPDIR/vals.dat’,a,b);
clear a
clear b
oldload(’TMPDIR/vals.dat’,’a’,’b’);

SEE ALSO:  save 258, getf 272

Scilab Group  April 1999  250
1.4.32 oldsave ______________ saving variables in 2.4.1 and previous format

CALLING SEQUENCE:

oldsave(filename [,x1,x2,...,xn])

PARAMETERS:

filename : character string or a logical unit returned by file('open',...)
xi : arbitrary Scilab variable(s)

DESCRIPTION:

The oldsave function is obsolete and is retained only for compatibility purpose.

The oldsave(filename) saves all current variables in the file defined by filename.
oldsave(file-name,x,y) saves only named variables x and y.
Saved variables can be reloaded by the load or oldload command.

EXAMPLES:

a=eye(2,2);b=ones(a);
oldsave('TMPDIR/val.dat',a,b);
clear a
clear b
oldload('TMPDIR/val.dat','a','b');

SEE ALSO: load 239, file 234

1.4.33 print __________________________ prints variables in a file

CALLING SEQUENCE:

print('file-name',x1,[x2,...xn])

DESCRIPTION:

prints xi on file ‘file-name’ with the current format, i.e. the format used by scilab to display the
variables. All types of variables may be ‘print’ed
Note: xi must be a named variable, with expressions variable name part of the display is unpredictable.
print(%io(2),...) prints on Scilab’s window. this syntax may be used to display variables within
a macro.

EXAMPLES:

a=rand(3,3);p=poly([1,2,3],'s');l=list(1,'asdf',[1 2 3]);
print(%io(2),a,p,l)
write(%io(2),a)

SEE ALSO: write 262, read 254, format 42, printf 252, disp 233
1.4.34 printf
Emulator of C language printf function

CALLING SEQUENCE:

printf(format, value_1, .., value_n)

PARAMETERS:

format: a Scilab string. Specifies a character string combining literal characters with conversion specifications.
value_i: Specifies the data to be converted according to the format parameter.
str: column vector of character strings
file: a Scilab string specifying a file name or a logical unit number (see file)

DESCRIPTION:
The printf function converts, formats, and writes its value parameters, under control of the format parameter, to the standard output.

The format parameter is a character string that contains two types of objects:

Literal characters: which are copied to the output stream.
Conversion specifications: each of which causes zero or more items to be fetched from the value parameter list. See printf_conversion for details.

If any values remain after the entire format has been processed, they are ignored.

EXAMPLES:

printf('Result is:\nalpha=%f", 0.535)

SEE ALSO: string 284, print 251, write 262, format 42, disp 233, file 234, fprintf 235, sprintf 260

1.4.35 printf_conversion
printf, sprintf, fprintf conversion specifications

DESCRIPTION:
Each conversion specification in the printf, sprintf, fprintf format parameter has the following syntax:

- A % (percent) sign.
- Zero or more options, which modify the meaning of the conversion specification. The following list contains the option characters and their meanings:
  - : Left align, within the field, the result of the conversion.
  + : Begin the result of a signed conversion with a sign (+ or -).
  "space": Prefix a space character to the result if the first character of a signed conversion is not a sign. If both the (space) and + options appear, the (space) option is ignored.
  # : Convert the value to an alternate form. For c, d, i, s, and u conversions, the # option has no effect. For o conversion, # increases the precision to force the first digit of the result to be a 0 (zero). For x and X conversions, a nonzero result has 0x or 0X prefixed to it. For e, E, f, g, and G conversions, the result always contains a decimal point, even if no digits follow it. For g and G conversions, trailing zeros are not removed from the result.
  0 : Pad to the field width, using leading zeros (following any indication of sign or base) for d, i, o, u, x, X, e, E, f, g, and G conversions; no space padding is performed. If the 0 and -- (dash) flags both appear, the 0 flag is ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag is also ignored.
An optional decimal digit string that specifies the minimum field width. If the converted value has fewer characters than the field width, the field is padded on the left to the length specified by the field width. If the left-adjustment option is specified, the field is padded on the right.

An optional precision. The precision is a . (dot) followed by a decimal digit string. If no precision is given, the parameter is treated as 0 (zero). The precision specifies:

- The minimum number of digits to appear for d, u, o, x, or X conversions
- The number of digits to appear after the decimal point for e, E, and f conversions
- The maximum number of significant digits for g and G conversions
- The maximum number of characters to be printed from a string in an s conversion
- A character that indicates the type of conversion to be applied:

% : Performs no conversion. Displays %.

d, i : Accepts an integer value and converts it to signed decimal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as a leading character causes the field width value to be padded with leading zeros.

u : Accepts an integer value and converts it to unsigned decimal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros.

o : Accepts an integer value and converts it to unsigned octal notation. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros.

x, X : Accepts an integer value and converts it to unsigned hexadecimal notation. The letters "abcdef" are used for the x conversion; the letters "ABCDEF" are used for the X conversion. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is a null string. Specifying a field width with a zero as the leading character causes the field width value to be padded with leading zeros.

f : Accepts a float or double value and converts it to decimal notation in the format %[-]ddd.ddd. The number of digits after the decimal point is equal to the precision specification.

- If no precision is specified, six digits are output.
- If the precision is zero, no decimal point appears and the system outputs a number rounded to the integer nearest to value.
- If a decimal point is output, at least one digit is output before it.

e, E : Accepts a real and converts it to the exponential form %[-]d.ddde+/–dd. There is one digit before the decimal point, and the number of digits after the decimal point is equal to the precision specification.

- If no precision is specified, , six digits are output.
- If the precision is zero, , no decimal point appears.
- The E conversion character produces a number with E instead of e before the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.

g, G : Accepts a real and converts it in the style of the e, E, or f conversion characters, with the precision specifying the number of significant digits. Trailing zeros are removed from the result. A decimal point appears only if it is followed by a digit. The style used depends on the value converted. Style e (E, if G is the flag used) results only if the exponent resulting from the conversion is less than -4, or if it is greater or equal to the precision.

c : Accepts and displays an integer value converted to a character.

s : Accepts a string value and displays characters from the string to the end or the number of characters indicated by the precision is reached. If no precision is specified, all characters up to the end are displayed.
A field width or precision can be indicated by an * (asterisk) instead of a digit string. In this case, an integer value parameter supplies the field width or precision. The value parameter converted for output is not fetched until the conversion letter is reached, so the parameters specifying field width or precision must appear before the value to be converted (if any).

If the result of a conversion is wider than the field width, the field is expanded to contain the converted result.

The representation of the plus sign depends on whether the + or (space) formatting option is specified.

**SEE ALSO:** printf 252, fprintf 235, sprintf 260

### 1.4.36 read matrices read

**CALLING SEQUENCE:**

```latex
[x]=read(file-desc,m,n,[format])
[x]=read(file-desc,m,n,k,format)
```

**PARAMETERS:**

- file-desc: character string specifying the file name or integer value specifying logical unit (see file).
- m, n: integers (dimensions of the matrix x). Set m=-1 if you do not know the numbers of rows, so the whole file is read.
- format: character string, specifies a "Fortran" format. This character string must begin with a right parenthesis and end with a left parenthesis. Formats cannot mix floating point or character edition modes.
- k: integer or vector of integer

**DESCRIPTION:**

reads row after row the m\times n matrix x (n=1 for character chain) in the file file-desc (string or integer). Each row of the matrix x begin in a new line of file-desc file. Depending on format, a given row of the x matrix may be read from more than one line of file-desc file.

The type of the result will depend on the specified format. If format contains only (d,e,f,g) descriptors the function tries to read numerical data (the result is matrix of real numbers).

If format contains only a descriptors the function tries to read character strings (the result is a character string column vector). In this case n must be equal to 1.

Examples for format:

- (l1x,e10.3,5x,3(f3.0))
- (10x,a20)

When format is omitted datas are read using numerical free format:

blank, comma and slash may be used as data separators, n\times v may be use to represent n occurrences of value v.

A direct access file can be used if using the parameter \vvm\fr which is the vector of record numbers to be read (one record per row), thus \vvm\fr must be \vvm=prod(size(k))\fr.

To read on the keyboard use \vread(%io(1),...\fr.

**REMARK**

Last line of data files must be terminated by a newline to be taken into account.
.SH EXAMPLE
.if
  if MSDOS then unix('del foo');
  else unix('rm -f foo'); end
  A=rand(3,5); write('foo',A);
  B=read('foo',3,5)
  B=read('foo',-1,5)
  read(%io(1),1,1,'(a)') // waits for user's input

SEE ALSO:  file 234,  readb 255,  write 262,  x_dialog 292,  mscanf 248,  mfscanf 248,  msscanf 248,  fscanfMat 237

1.4.37  read4b  fortran file binary read

CALLING SEQUENCE :
  x=read4b(file-name,m,n [,rec])

PARAMETERS :
  file-name : string or integer
  m, n : integers (dimensions of the matrix x). Set m=-1 if you do not know the numbers of rows, so all the
  file is read
  rec : vector of positive integers. the selected records for direct access. This vector size must be equal to
  the number of rows of desired x.

DESCRIPTION :
  binary read of the matrix x in the file file-name. Matrix entries are supposed to have been stored on 4
  byte words.
  For direct record access, file must have been previously opened using file function to set the record length.
  file-name must be the result of the file function.

SEE ALSO:  file 234,  write 262,  writb 261,  mget 242,  write4b 262

1.4.38  readb  fortran file binary read

CALLING SEQUENCE :
  x=readb(file-name,m,n [,rec])

PARAMETERS :
  file-name : string or integer
  m, n : integers (dimensions of the matrix x). Set m=-1 if you do not know the numbers of rows, so all the
  file is read
  rec : vector of positive integers. the selected records for direct access. This vector size must be equal to
  the number of rows of desired x.

DESCRIPTION :
  binary read of the matrix x in the file file-name. Matrix entries are supposed to have been stored on 8
  byte words.
  For direct record access, file must have been previously opened using file function to set the record length.
  file-name must be the result of the file function.

SEE ALSO:  file 234,  write 262,  writb 261,  mget 242,  read4b 255
1.4.39  **readc_** read a character string

**CALLING SEQUENCE:**

[c]=readc_(unit)
[c]=readc_()

**DESCRIPTION:**

readc_ reads a character string. This function allows one to interrupt an exec file without pause; the exec file stops until carriage return is made.

**SEE ALSO:** read

1.4.40  **readmps** reads a file in MPS format

**CALLING SEQUENCE:**

mps= readmps (file-name, bounds [, maxsizes]);

**PARAMETERS:**

- **file-name**: character string, path of the MPS file
- **bounds**: 2-vector [lowbound, upbound], default lower ans upper bounds
- **maxsizes**: 3-vector [maxm, maxn, maxnza] Maximum number of contraints and variables, maximum number of nonzeros entries in the LP constraint matrix. If omitted readmps reads the file once just to compute these numbers.
- **mps**: list with following fields
  - **irow**: integer (index of the objective row).
  - **namec**: character string (Name of the objective).
  - **nameb**: character string (Name of the right hand side).
  - **namran**: character string (Name of the ranges section).
  - **nambnd**: character string (Name of the bounds section).
  - **name**: character string (Name of the LP problem).
  - **rownames**: character string column vector (Name of the rows).
  - **colnames**: character string row vector (Name of the columns).
  - **rowstat**: integer vector, row types:
    - 1: row type is "="
    - 2: row type is ">="
    - 3: row type is "<="
    - 4: objective row
    - 5: other free row
  - **rowcode**: real matrix [hdrowcd, lnkrow] with
    - **hdrowcd**: real vector (Header to the linked list of rows with the same codes).
    - **lnkrow**: integer vector (Linked list of rows with the same codes).
  - **colcode**: real matrix [hdcolcd, lnkcol] with
    - **hdcolcd**: integer vector (Header to the linked list of columns with the same codes).
    - **lnkcol**: integer vector (Linked list of columns with the same codes).
  - **rownmbs**: integer vector (Row numbers of nonzeros in columns of matrix A.)
  - **colpnts**: integer vector (Pointers to the beginning of columns of matrix A).
  - **acoeff**: real vector (Array of nonzero elements for each column).
  - **rhs**: real vector (Right hand side of the linear program).
  - **ranges**: real vector of constraint ranges.
  - **bounds**: real matrix [lbounds, ubounds] with
    - **lbounds**: full column vector of lower bounds.
    - **ubounds**: full column vector of upper bounds.
lbounds : full column vector of lower bounds
stavar : full column vector of variable status
0 : standard (non negative) variable
1 : upper bounded variable
2 : lower bounded variable
3 : lower and upper bounded variable
4 : minus infinity type variable i.e. -inf < x <= u
5 : plus infinity type variable i.e. l <= x < inf
6 : fixed type variable i.e. l = x = u
-k : free variable

DESCRIPTION:
readmps. Utility function: reads a file containing description of an LP problem given in MPS format. It is an interface with the program rdmps1.f of hopdm (J. Gondzio). For a description of the variables, see the file rdmps1.f.

MPS format is a standard ASCII medium for LP codes. MPS format is described in more detail in Murtagh’s book:

EXAMPLE:

// Let the LP problem:
// objective:
// min XONE + 4 YTWO + 9 ZTHREE
// constraints:
// LIM1: XONE + YTWO <= 5
// LIM2: XONE + ZTHREE >= 10
// MYEQN: - YTWO + ZTHREE = 7
// Bounds
//  0 <= XONE <= 4
// -1 <= YTWO <= 1

// Generate MPS file
txt=[‘NAME TESTPROB’
‘ROWS’
‘ N COST’
‘ L LIM1’
‘ G LIM2’
‘ E MYEQN’
‘COLUMNS’
‘ XONE COST 1 LIM1 1’
‘ XONE LIM2 1’
‘ YTWO COST 4 LIM1 1’
‘ YTWO MYEQN -1’
‘ ZTHREE COST 9 LIM2 1’
‘ ZTHREE MYEQN 1’
‘RHS’
‘ RHS1 LIM1 5 LIM2 10’
‘ RHS1 MYEQN 7’
‘BOUNDS’
‘ UP BND1 XONE 4’
‘ LO BND1 YTWO -1’
‘ UP BND1 YTWO 1’
‘ENDATA’];
u=file(‘open’,’/tmp/test.mps’,’unknown’)
write(u,tex,'(a)');file('close',u)
//Read the MPS file
P=readmps('/tmp/test.mps',[0 10^30])
//Convert it to linpro format
LP=mps2linpro(P)
//Solve it with linpro
[x,lagr,f]=linpro(LP(2:))

SEE ALSO: mps2linpro 202

1.4.41  save ___________________________ saving variables in binary files

CALLING SEQUENCE:

save(filename [,x1,x2,...,xn])  
save(fd [,x1,x2,...,xn])

PARAMETERS:

filename : character string containing the path of the file
fd : a file descriptor given by a call to mopen
xi : arbitrary Scilab variable(s)

DESCRIPTION:

The save command can be used to save Scilab current variables in a binary file. The file can be given either by its paths or by its descriptor previously given by mopen.

save(filename) saves all current variables in the file defined by filename.
save(fd) saves all current variables in the file defined by the descriptor fd.

save(filename,x,y) or save(fd,x,y) saves only named variables x and y.

Saved variables can be reloaded by the load command.

EXAMPLES:

a=eye(2,2);b=ones(a);
save('val.dat',a,b);
clear a
clear b
load('val.dat','a','b');

// sequential save into a file
fd=mopen('TMPDIR/foo','wb')
for k=1:4, x=k^2;save(fd,x,k),end
mclose(fd)

fd=mopen('TMPDIR/foo','rb')
for i=1:4, load(fd,'x','k');x,k,end
mclose(fd)

// appending variables to an old save file
fd=mopen('TMPDIR/foo','r+')
mseek(0,fd,'end')
lst=list(1,2,3)
save(fd,lst)
mclose(fd)

SEE ALSO: load 239, mopen 244
1.4.42 scanf ___________________ Converts formatted input on standard input

CALLING SEQUENCE:

[v_1,...v_n]=scanf (format);

PARAMETERS:

format : Specifies the format conversion.

DESCRIPTION:

The scanf functions get character data on standard input (%io(1)), interpret it according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input.

The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. See scanf_conversion for details.

SEE ALSO: printf 252, read 254, fscanf 237, sscanf 260

1.4.43 scanf_conversion __________ scanf, sscanf, fscanf conversion specifications

DESCRIPTION:

Each conversion specification in the format parameter contains the following elements:

+ The character % (percent sign)
+ The optional assignment suppression character *
+ An optional numeric maximum field width
+ A conversion code

The conversion specification has the following syntax:

[*] [width] [size] convcode.

The results from the conversion are placed in v arguments unless you specify assignment suppression with * (asterisk). Assignment suppression provides a way to describe an input field that is to be skipped. The input field is a string of nonwhite-space characters. It extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion code indicates how to interpret the input field. You should not specify the v parameter for a suppressed field. You can use the following conversion codes:

% : Accepts a single % (percent sign) input at this point; no assignment is done.
d, i : Accepts a decimal integer;
u : Accepts an unsigned decimal integer;
o : Accepts an octal integer;
x : Accepts a hexadecimal integer;
e, f, g : Accepts a floating-point number. The next field is converted accordingly and stored through the corresponding parameter, which should be a pointer to a float. The input format for floating-point numbers is a string of digits, with the following optional characteristics:
+ It can be a signed value.
+ It can be an exponential value, containing a decimal point followed by an exponent field, which consists of an E or an e followed by an (optionally signed) integer.
It can be one of the special values INF, NaN.

- : Accepts a string of characters.
- c : character value is expected. The normal skip over white space is suppressed.

**SEE ALSO:** scanf 259, scanf 259, fscanf 237

### 1.4.44 sprintf

**Emulator of C language sprintf function**

**CALLING SEQUENCE:**

\[ \text{str} = \text{sprintf} (\text{format}, \text{value}_1, \ldots, \text{value}_n) \]

**PARAMETERS:**

- format: A Scilab string. Specifies a character string combining literal characters with conversion specifications.
- value\(_i\): Specifies the data to be converted according to the format parameter.
- str: Column vector of character strings

**DESCRIPTION:**

The `sprintf` function converts, formats, and stores its value parameters, under control of the format parameter. The format parameter is a character string that contains two types of objects:

- **Literal characters:** which are copied to the output stream.
- **Conversion specifications:** each of which causes zero or more items to be fetched from the value parameter list. See `printf_conversion` for details.

If there are not enough items for format in the value parameter list, `sprintf` generates an error. If any values remain after the entire format has been processed, they are ignored.

**EXAMPLES:**

```plaintext
fahr=120
sprintf('%3d Fahrenheit = %6.1f Celsius', fahr, (5/9)*(fahr-32))
```

**SEE ALSO:** string 284, print 251, write 262, format 42, disp 233, file 234, printf 252, fprintf 235, m sprintf 245

### 1.4.45 sscanf

**Converts formatted input given by a string**

**CALLING SEQUENCE:**

\[ [v_1, \ldots v_n] = \text{sscanf} (\text{string}, \text{format}) \]

**PARAMETERS:**

- format: Specifies the format conversion. : Specifies the input file name or file number.
- string: Specifies input to be read.

**DESCRIPTION:**

The sscanf functions interpret character string according to a format, and returns the converted results.

The format parameter contains conversion specifications used to interpret the input. The format parameter can contain white-space characters (blanks, tabs, newline, or formfeed) that, except in the following two cases, read the input up to the next nonwhite-space character. Unless there is a match in the control string, trailing white space (including a newline character) is not read.
write Scilab Function

- Any character except % (percent sign), which must match the next character of the input stream.
- A conversion specification that directs the conversion of the next input field. see scanf, conversion

See Also: printf 252, read 254, scanf 259, fscanf 237

1.4.46 startup ________________________________ startup file

DESCRIPTION:
The startup files .scilab (in your home directory) and .scilab in your working directory are automatically executed (if present) when Scilab is invoked, in addition with the file scilab.star in the Scilab directory.

REMARK:
Last line of startup file must be terminated by a newline to be taken into account.

1.4.47 warning ____________________________ warning messages

CALLING SEQUENCE:

warning('string')

DESCRIPTION:
prints the character string ‘string’ in a warning message

See Also: error 35

1.4.48 writb _______________________________ fortran file binary write

CALLING SEQUENCE:

writb(file-name, a [,rec])

PARAMETERS:

file-name : string or integer
rec : vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of a

DESCRIPTION:
writes in binary format the matrix a in the file ‘filename’. Matrix entries are stored on 4 byte words For direct record access, file must have been previously opened using file function to set the record length.
file-name must be the result of the file function.

See Also: file 234, read 255, write 262, mput 246, write4b 262

Scilab Group April 1993 261
1.4.49 write ___________________________________ write in a formatted file

DESCRIPTION:

write(file-desc,a,[format])
write(file-desc,a,k,format)

PARAMETERS:

file-desc : character string specifying the file name or integer value specifying logical unit (see file).
a : real matrix or column vector of character strings.
format : character string, specifies a "Fortran" format. This character string must begin with a right parenthesis and end with a left parenthesis. Formats cannot mix floating point, integer or character edition modes
k : integer vector

DESCRIPTION:
writes row-by-row a real matrix or a column vector of character strings in a formatted file. Each row of the a argument begin in a new line of file-desc file. Depending on format a given row of the a argument may be written in more than one line of file-desc file.
Format examples: (1x,e10.3,5x,3(f3.0)) , (10x,a20) ;
See a Fortran book for more precision.
Direct access files : x=write(file-desc,a,k,format). Here k is the vector of records (one record by row, i.e. m=prod(size(k))
write(%io(2),....) writes on Scilab's window.

EXAMPLE:

if MSDOS then unix('del asave');
else unix('rm -f asave'); end
A=rand(5,3); write('asave',A); A=read('asave',5,3);
write(%io(2),A,'('' | '',3(f10.3,'' | ''))')
write(%io(2),string(1:10))
write(%io(2),strcat(string(1:10),','))
write(%io(2),1:10,'(10(i2,3x))')
if MSDOS then unix('del foo');
else unix('rm -f foo'); end
write('foo',A)

SEE ALSO: file 234, writb 261, read 254, print 251, string 284, mfprintf 245, mprintf 245, msprintf 245, fprintfMat 236

1.4.50 write4b _____________________________ fortran file binary write

CALLING SEQUENCE:

write4b(file-name,a [,rec])

PARAMETERS:

file-name : string or integer
rec : vector of positive integers. the selected records for direct access. This vector size must be equal to the number of rows of a

Scilab Group  Jan 1996  262
DESCRIPTION:
writes in binary format the matrix a in the file 'filename'. Matrix entries are stored on 8 byte words
For direct record access, file must have been previously opened using file function to set the record length.
file-name must be the result of the file function.
SEE ALSO: file 234, readb 255, write 262, mput 246, read4b 255

1.4.51 xgetfile ________________________________ dialog to get a file path

CALLING SEQUENCE:
path=xgetfile()  
path=xgetfile(file_mask,[title='string'])  
path=xgetfile(file_mask,dir,[title='string'])  
path=xgetfile(file_mask,dir,'string')

PARAMETERS:
  file_mask : a character string which gives the file mask to use for file selection. file_mask is written
              with Unix convention. the default value is '*'.
  dir : a character string which gives the initial directory used for file search. by default xgetfile uses the
        previously selected directory.
  path : is the user selected file path if user answers "Ok" or the " " string if user answers "Cancel"
  title='string' : Optional arguments which gives the title for the xgetfile window.

DESCRIPTION:
Creates a dialog window for file selection

EXAMPLE:
xgetfile()  
xgetfile('*.sci','SCI/macros/xdess')  
xgetfile(title='Choose a file name ')

SEE ALSO: x_dialog 292, file 234, read 254, write 262, exec 36, getf 272
1.5 Handling of functions and libraries
1.5.1 addinter __________ new functions interface incremental linking at run time

CALLING SEQUENCE:

addinter(files, spname, fcts)

PARAMETERS:

files: a character string or a vector of character string contain object files used to define the new Scilab interface routine (interface code, user routines or libraries, system libraries).
spname: a character string. Name of interface routine entry point
fcts: vector of character strings. The name of new Scilab function implemented in the new interface (in the order).

DESCRIPTION:

addinter performs incremental linking of a compiled C or Fortran new Scilab interface routine (see intersci documentation) and define corresponding scilab functions. For machines using dlopen functionality one can unlink an interface with ulink (use the command link('show') to get the number of the shared library). And to reload a new version of an interface a call to ulink is necessary to get rid of the old version. See link for more precision on use.

SEE ALSO: link 303, intersci 303, newfun 274, clearfun 265

1.5.2 argn ________________ number of arguments in a function call

CALLING SEQUENCE:

[lhs [, rhs] ]=argn()
lhs=argn(1)
rhs=argn(2)

DESCRIPTION:

This function is used inside a function definition. It gives the number of actual inputs rhs and output lhs parameters passed to the function when the function is called. It is usually used in function definitions to deal with optional arguments.

SEE ALSO: function 268, varargin 277

1.5.3 clearfun ______________ remove primitive.

CALLING SEQUENCE:

clearfun('name')

DESCRIPTION:

clearfun('name') removes the primitive 'name' from the set of primitives (built-in functions). This function allows to rename a primitive: a Scilab primitive can be replaced by a user-defined function. For experts...

SEE ALSO: newfun 274, funptr 45
1.5.4  comp  ____________________________  scilab function compilation

CALLING SEQUENCE :

comp(function [,opt])

PARAMETERS :

function : an not compiled scilab function (type 11)
opt : integer flag with value 0 (default) or 1.

DESCRIPTION :
comp(function) compiles the function function. Compiled and interpreted functions are equivalent but usually compiled functions are much faster. The functions provided in the standard libraries are compiled.
The command: getf('filename') loads the functions in file 'filename' and compiles them. So comp has to be used in very particular cases.
The opt=1 option is specific to code analysis purpose (see macr2lst)

REMARKS :
commands who, help, what cannot be compiled.

SEE ALSO:  deff 266, getf 272, whereis 77, macr2lst 273, lib 272

1.5.5  deff  ____________________________  on-line definition of function

CALLING SEQUENCE :

deff('[s1,s2,...]=newfunction(e1,e2,...)',text [,opt])

PARAMETERS :

e1,e2,..., : input variables.
s1,s2,..., : output variables.
text : matrix of character strings
opt : optional character string
'c' : function is "compiled" to be more efficient (default)
'n' : function is not "compiled"

DESCRIPTION :
On-line definition of function (user defined function): the name of the created function is newfunction. text is a sequence of instructions usually set as a vector of character strings.
This command can be used inside a function and the new function can be an input or output of any other function.
Usually, functions are defined in a file and loaded into Scilab by getf
Some time, in particular when you want to use define strings within deff text is rather difficult to write. A more tractable way may be to define your function in a file as usual, to load it into Scilab by getf (without 'c' option) and use sci2exp to get corresponding deff instructions.

EXAMPLES :

deff('[x]=myplus(y,z)',"'x=y+z'")
//
deff('[x]=mymacro(y,z)',"'a=3*y+1'; 'x=a*z+y'\")

SEE ALSO:  getf 272, comp 266, exec 36, function 268

Scilab Group  April 1993  266
1.5.6 delbpt ____________________________________________ delete breakpoint

CALLING SEQUENCE:

delbpt('macroname' [,linenumb])

DESCRIPTION:
deletes the breakpoint at line linenumb in the function macroname. If linenumb is omitted all the breakpoints in the function are deleted.

EXAMPLE:

setbpt('foo',1), setbpt('foo',10), delbpt('foo',10), dispbpt()
delbpt('foo',1)

SEE ALSO: setbpt 276, dispbpt 267, pause 64, resume 68

1.5.7 dispbpt ________________________________ display breakpoints

CALLING SEQUENCE:

dispbpt()

DESCRIPTION:
dispbpt() displays all active breakpoints actually inserted in functions.

SEE ALSO: setbpt 276, delbpt 267, pause 64, resume 68

1.5.8 edit ________________________________ function editing

CALLING SEQUENCE:

newname=edit(functionname [, editor])

PARAMETERS:

functionname : character string
editor : character string

DESCRIPTION:
If functionname is the name of a defined scilab function edit(functionname [,editor]) try to open the associated file functionname.sci. If this file can’t be modified edit first create a copy of this file in the TMPDIR directory.

If functionname is the name of a undefined scilab function edit create a functionname.sci file in the TMPDIR directory.

When leaving the editor the modified or defined function is loaded into Scilab under the name newname.

The editor character string can be used to specify your favourite text editor.

Default editor is Emacs. This function should be customized according to your needs.

EXAMPLE:

//newedit=edit('edit')  //opens editor with text of this function
//myfunction=edit('myfunction')  //opens editor for a new function

SEE ALSO: manedit 239
1.5.9  funcprot ____________________ switch scilab functions protection mode

CALLING SEQUENCE:

prot=funcprot ()
funcprot (prot)

PARAMETERS:

prot : integer with possible values 0,1,2

DESCRIPTION:
Scilab functions are variable, funcprot allows the user to specify what scilab do when such variables are redefined.

* If prot==0 nothing special is done
* If prot==1 scilab issues a warning message when a function is redefined (default mode)
* If prot==2 scilab issues an error when a function is redefined

EXAMPLE:

funcprot (1)
deff (’[x]=foo(a)’,’x=a’)«
deff (’[x]=foo(a)’,’x=a+1’)«
foo=33
funcprot (0)
deff (’[x]=foo(a)’,’x=a’)«
deff (’[x]=foo(a)’,’x=a+1’)«
foo=33

1.5.10 function __________________________ opens a function definition

endfunction - closes a function definition

SYNTAX:

function <lhs_arguments>=<function_name><rhs_arguments>
<statements>
endfunction

Where

<function_name> stands for the name of the function
<rhs_arguments> stands for the input argument list. It may be
  - a comma separated sequence of variable names enclosed in parenthesis, like (x1,...,xm). Last variable name can be the key word varargin (see varargin)
  - the sequence () or nothing, if the function has no input argument.
<lhs_arguments> stands for the output argument list. It may be
  - a comma separated sequence of variable names enclosed in brackets, like [y1,...,yn]. Last variable name can be the key word varargout (see varargout)
  - the sequence [], if the function has no input argument. In this case the syntax may also be: function <function_name><rhs_arguments>
<statements> stands for a set of scilab instructions (statements)
DESCRIPTION:
This syntax may be used to define function (see functions) inline or in a script file (see exec). For compatibility with old Scilab versions, functions defined in a script file containing only function definitions can be "loaded" into Scilab using the \texttt{getf} function.

The function \texttt{<lhs_arguments>=<function_name><rhs_arguments>} sequence cannot be split over several lines. This sequence can be followed by statements in the same line if a comma or semi column is added at its end.

function definitions can be nested

EXAMPLE:

//inline definition (see function)
function \([x,y]=myfct(a,b)\)
    x=a+b
    y=a-b
endfunction

\([x,y]=myfct(3,2)\)

//a one line function definition
function y=sq(x),y=x^2,endfunction

sq(3)

//nested functions definition
function y=foo(x)
a=sin(x)
    function y=sq(x), y=x^2,endfunction
    y=sq(a)+1
endfunction

foo(%pi/3)

// definition in an script file (see exec)
exec SCI/macros/elem/asin.sci;

SEE ALSO: functions 269, exec 36, getf 272

1.5.11 functions _______________________ Scilab procedures and Scilab objects

DESCRIPTION:
Functions are Scilab procedures ("macro", "function" and "procedure" have the same meaning).

FUNCTION DEFINITION:
Usually, they are defined in files with an editor and loaded into Scilab by \texttt{getf} or through a library (see \texttt{lib} or \texttt{genlib}). But they can also be defined on-line (see \texttt{deff} or \texttt{function o}.

A function is defined by two components:

- a "syntax definition" part as follows:

\[
[y_1,...,y_n]=\text{foo}(x_1,...,x_m)
\]

\[
[y_1,...,y_n,\text{varargout}]=\text{foo}(x_1,...,x_m,\text{varargin})
\]
- a sequence of scilab instructions.

The "syntax definition" line gives the "full" calling syntax of this function. The \( y_i \) are output variables calculated as functions of input variables \( x_i \) and variables existing in Scilab when the function is executed.

**CALLING FUNCTION:**

Usually function calling syntax is \([y_1, \ldots, y_n] = \text{foo}(x_1, \ldots, x_m)\). Shorter input or output argument list than definition ones may be used. In such cases, only the first variables from the left are used of set. The `argn` function may be used to get the actual number of calling arguments.

It is also possible to use "named argument" to specify input arguments: suppose function `fun1` defined as \( y_1 = \text{fun1}(x_1, x_2, x_3) \) then it can be called with a syntax like \( y = \text{fun1}(x_1=33, x_3=[1 \ 2 \ 3]) \) within `fun1` \( x_2 \) will be undefined. It is possible to check for defined variables with the `exists` function.

When a function has no left hand side argument and is called only with character string arguments, the calling syntax may be simplified: \( \text{fun}('a','toto','a string') \) can be replaced by \( \text{fun} \ a \ \text{toto} \ 'a \text{ string}' \)

**MISCELLANEOUS:**

Functions are Scilab objects (with type numbers 13 or 11). They and can be manipulated (built, saved, loaded, passed as arguments,..) as other variable types.

Collections of functions can be collected in libraries. Functions which begin with %% sign (e.g. %%foo) are often used to overload (see overloading) operations or functions for new data type.

**EXAMPLE:**

```scilab
// inline definition (see function)
function [x,y]=myfct(a,b)
    x=a+b
    y=a-b
endfunction

[x,y]=myfct(3,2)

// inline definition (see deff)
deff('[x,y]=myfct(a,b)',['x=a+b';
    'y=a-b'])

// definition in an ascii file (see exec)
exec SCI/macros/elemd/asin.sci;

// definition in an ascii file (see getf)
getf SCI/macros/elemd/asin.sci;
```

**SEE ALSO:** `function` 268, `deff` 266, `getf` 272, `comp` 266, `lib` 272, `getd` 271, `genlib` 270, `exists` 37, `varargin` 277, `varargout` 277

### 1.5.12 genlib

**build library from all functions in given directory**

**CALLING SEQUENCE:**

```scilab
genlib(lib-name [,dir-name])
```

**PARAMETERS:**

- `lib-name`: Scilab string. The variable name of the library to (re)create.
dir-name: Scilab string. The name of the directory to look for .bin-files; default value is the current directory.

DESCRIPTION:
For each .sci file in dir-name, genlib executes a getf and saves the functions to the corresponding .bin file. The .sci file must not contain anything but Scilab functions. If a .bin file is newer than the associated .sci file, genlib does not translate and save the file.

When all .sci files have been processed, genlib creates a library variable named lib-name and saves it in the file lib in dir-name.

RESTRICTIONS:
Scilab tacitly assumes that file foo.sci defines only a single function named foo.

SEE ALSO: getf 272, save 258, lib 272

1.5.13 get_function_path _______________ get source file path of a library function

CALLING SEQUENCE:

path=get_function_path(fun_name)

PARAMETERS:

fun_name : a string, the name of the function
path : a string, the absolute pathname of the function source file (.sci) or [].

DESCRIPTION:
Given the name of a function get_function_path returns the absolute pathname of the function source file if the function is defined in a Scilab library (see lib) or [] if name does not match any library function.

EXAMPLE:

get_function_path('median')

SEE ALSO: lib 272, string 284

1.5.14 getd _________________ getting all functions defined in a directory

CALLING SEQUENCE:

getd(path)

PARAMETERS:

path : Scilab string. The directory pathname

DESCRIPTION:
loads all .sci files (containing Scilab functions) defined in the path directory.

EXAMPLE:

getd('SCI/macros/auto')

SEE ALSO: getf 272, lib 272, getcwd 67, pwd 67, chdir 297
1.5.15 getf defining a function from a file

CALLING SEQUENCE:

getf(file-name [,opt])

PARAMETERS:

filename : Scilab string.
opt : optional character string
  "c" : loaded functions are "compiled" to be more efficient (default)
  "n" : loaded functions are not "compiled"
  "p" : loaded functions are "compiled" and prepared for profiling (see profile)

DESCRIPTION:

loads one or several functions (see functions) defined in the file 'file-name'. The string opt='n'
means that the functions are not compiled (pre-interpreted) when loaded. This can be useful for some
debugging purpose (see comp). By default, functions are compiled when loaded (i.e. opt='c' is used).
In the file a function must begin by a "syntax definition" line as follows:

function [y1,...,yn]=foo(x1,...,xm)
  followed by a sequence of scilab instructions.
The "syntax definition" line gives the "full" calling syntax of this function. The yi are output variables
calculated as functions of input variables xi and variables existing in Scilab when the function is executed.
Shorter input or output argument list may be used.
Many functions may be written in the same file. A function is terminated by an endfunction keyword.
For compatibility with previous versions a function may also be terminated by the following function
keyword or the EOF mark.

REMARK:

getf is an old way for loading functions into scilab from a file. If functions in a file are terminated by an
endfunction keyword, the file maybe loaded using the exec function instead of getf. In this case
default option opt is used.

EXAMPLE:

getf('SCI/macros/xdess/plot.sci')

getf SCI/macros/xdess/plot.sci

SEE ALSO: functions 269, function 268, genlib 270, getd 271, exec 36, edit 267, comp 266

1.5.16 lib library definition

CALLING SEQUENCE:

xlib = lib('lib-dir')

PARAMETERS:

lib-dir: character string

DESCRIPTION:

lib-dir is a character string defining a directory that contains compiled Scilab function (.bin) files.
In addition to these files lib-dir must have a file called names, that contains the names of the functions
defined in lib-dir. On success, all functions in lib-dir are available from within Scilab. They are loaded on demand when called for the first time.

Binary files can be created from within Scilab with the command save.

Scilab’s standard libraries are defined using lib on the SCIDIR/macros/* subdirectories.

As an example, given the following definitions

```
  deff('z = myplus(x, y)', 'z = x + y')
  deff('z = yourplus(x, y)', 'x = x - y')
  lib_dir = '/home/joeuser/mycidir'
```

myplus and yourplus are compiled into lib_dir with the commands

```
  save(lib_dir + '/myplus.bin', myplus)
  save(lib_dir + '/yourplus.bin', yourplus)
```

A library can now be created from the two .bin files with the command

```
  xlib = lib(lib_path + '/')
```

xlib is a Scilab variable of type "library". A library variable usually is saved for later loading, either on-line or from the user-specific startup file ($HOME/.scilab).

**RESTRICTIONS:**

Scilab tacitly assumes that file foo.bin defines only a single function named foo.

SEE ALSO: genlib 270, save 258, deff 266, getf 272, whereis 77

### 1.5.17 macr2lst __________________________ function to list conversion

**CALLING SEQUENCE:**

```
[txt]=macr2lst(function-name)
```

**DESCRIPTION:**

This primitive converts a compiled Scilab function function-name into a list which codes the internal representation of the function. For use with mac2for.

SEE ALSO: macrovar 274

### 1.5.18 macro __________________________ Scilab procedure and Scilab object

**DESCRIPTION:**

Macros are Scilab procedures (“macro”, “function” and “procedure” have the same meaning). Usually, they are defined in files with an editor and loaded into Scilab by getf or through a library. They can also be defined on-line (see deff). A file which contains a macro must begin as follows:

```
function [y1,...,yn]=foo(x1,...,xm)
```

The yi are output variables calculated as functions of input variables and variables existing in Scilab when the macro is executed. A macro can be compiled for faster execution. Collections of macros can be collected in libraries. Macros which begin with % sign (e.g. %foo) and whose arguments are lists are used to perform specific operations: for example, z=%rmr(x,y) is equivalent to z=x*y when x and z are rationals (i.e. x=list('r',n,d,[])) with n and d polynomials.

SEE ALSO: deff 266, getf 272, comp 266, lib 272
1.5.19  macrovar ____________________________ variables of function

**CALLING SEQUENCE:**

\[ \text{vars}=\text{macrovar} (\text{function}) \]

**PARAMETERS:**

\[
\text{vars} : \text{list list (in, out, globals, called, locals)} \\
\text{function} : \text{name of a function}
\]

**DESCRIPTION:**

Returns in a list the set of variables used by a function. \( \text{vars} \) is a list made of five column vectors of character strings 
- \( \text{in} \) : input variables (\( \text{vars}(1) \))
- \( \text{out} \) : output variables (\( \text{vars}(2) \))
- \( \text{globals} \) : global variables (\( \text{vars}(3) \))
- \( \text{called} \) : names of functions called (\( \text{vars}(4) \))
- \( \text{locals} \) : local variables (\( \text{vars}(5) \))

**EXAMPLE:**

\[
\text{deff} ('y=f(x1,x2)','loc=1;y=a*x1+x2-loc') \\
\text{vars}=\text{macrovar} (f)
\]

**SEE ALSO:** string 284, macr2lst 273

1.5.20  newfun ______________________________ add a name in the table of functions

**CALLING SEQUENCE:**

\[ \text{newfun} ("\text{function-name}", \text{nameptr}) \]

**DESCRIPTION:**

Utility function (for experts only). Adds the name "\text{function-name}" in the table of functions known to the interpreter. \( \text{nameptr} \) is an integer \( 100*\text{fun}+\text{fin} \) where \( \text{fun} \) and \( \text{fin} \) is the internal coding of the primitive "\text{function-name}". This function is useful to associate a primitive to a routine interfaced in "matur.f" (fun=14). Used with \text{funptr} and \text{clearfun} one can redefine a primitive by a function with same name.

**SEE ALSO:** clearfun 265

1.5.21  plotprofile ___ extracts and displays execution profiles of a Scilab function

**CALLING SEQUENCE:**

\[ \text{plotprofile} (\text{fun}) \]

**PARAMETERS:**

\[ \text{fun} : \text{a Scilab function} \]
DESCRIPTION:

To use `plotprofile` the Scilab function must have been prepared for profiling (see `getf`).

For such functions, when such a function is executed the systems counts how many times each line is executed and how much CPU time is spent for each line execution. These data are stored within the function data structure. The `plotprofile` function is an interactive function which displays this result in a graphic window. When a line is clicked the text of the function is displayed with the selected line highlighted.

NOTE: you have to click on the "Exit" item in the graphics window to exit from "plotprofile".

Function text is rebuild with `fun2string`.

EXAMPLE:

```scilab
//define function and prepare it for profiling
def('x=foo(n)',["if n==0 then"
  x=[]
  'else'
  x=0'
  'for k=1:n'
  '  s=svd(rand(n+10,n+10))'
  '  x=x+s(1)'
  '  end'
  'end'],'p')
//call the function
foo(30)
//get execution profiles
plotprofile(foo) // click on Exit to exit
```

SEE ALSO: `profile`, `showprofile`, `fun2string`.
showprofile Scilab Function

showprofile(rand(n+10,n+10))
' s=x+s(1)
end'
end',
//call the function
//get execution profiles
profile(foo)
//call the function
profile(foo) //execution profiles are cumulated

SEE ALSO: getf 272, deff 266, plotprofile 274, showprofile 276

1.5.23 setbpt _____________________________ setting breakpoints

CALLING SEQUENCE:
setbpt(macro-name [, line-num])

PARAMETERS:
macro-name : string
line-num : integer

DESCRIPTION:
setbpt interactively inserts a breakpoint in the line number line-num (default value is 1) of the function macro-name
When reaching the breakpoint, Scilab evaluates the specified line, prints the number of the line and the name of the function. If the function is not compiled (see comp) the line is printed on the screen. Then Scilab goes into a pause mode in which the user can check current values. The pause is exited with resume or abort. Redefining the function does not clear the breakpoints, the user must explicitly delete breakpoints using delbpt. The maximum number of functions with breakpoints enabled must be less than 20 and the maximum number of breakpoints is set to 100.
SEE ALSO: delbpt 267, dispbpt 267, pause 64, resume 68

1.5.24 showprofile ___ extracts and displays execution profiles of a Scilab function

CALLING SEQUENCE:
showprofile(fun)

PARAMETERS:
fun : a Scilab function

DESCRIPTION:
To use showprofile the Scilab function must have been prepared for profiling (see getf).
For such function, when such a function is executed the systems counts how many time each line is executed and how much time is spend for each line execution. These data are stored within the function data structure. The showprofile function outputs profiling results (see rofile) with text of the function lines.
Function text is rebuild with fun2string.

EXAMPLE:

Scilab Group july 2000 276
//define function and prepare it for profiling
def('x=foo(n)', [if n==0 then
    x=[]
else
    x=0
    for k=1:n
        s=svd(rand(n+10,n+10))
        x=x+s(1)
    end
end], 'p')

//call the function
foo(30)
//get execution profiles
showprofile(foo)

See Also: profile 275, plotprofile 274, fun2string 661

1.5.25 varargin ______ variable numbers of arguments in an input argument list

Syntax:
varargin must be the rightmost argument of the function definition input list.

Description:
A function whose input argument list contains varargin can be called with more input arguments than indicated in the input argument list. The calling arguments passed form varargin keyword onwards may then be retrieved within the function in a list named varargin. Suppose that varargin keyword is the n-th argument of the formal input argument list, then if the function is called with less than n-1 input arguments the varargin list is not defined, if the function is called with n-1 arguments then varargin list is an empty list.

\[ y = \text{function ex}(\text{varargin}) \]

may be called with any number of input arguments. Within function ex input arguments may be retrieved in varargin(i), i=1:length(varargin)

Remark:
Named argument syntax like \text{foo}(..., \text{key=value}) is incompatible with the use of varargin.

Example:
\text{deff('exampl(a, varargin)', ['[lhs, rhs]=argn(0)'
    'if rhs>1 then disp(varargin), end'])}
exampl(1)
exampl()
exampl(1,2,3)
l=list('a','%s','%t');
exampl(1,l(2:3))

See Also: function 268, varargin 277, list 57

1.5.26 varargout ______ variable numbers of arguments in an output argument list

Syntax:
varargout must be the rightmost argument of the function definition output list.

Description:
A function whose output argument list contains varargout must be called with more output arguments than indicated in the output argument list. The calling arguments passed form varargout keyword onwards are extracted out of the varargout list defined in the function.
The `varargout` Scilab keyword function `ex` can be called with any number of output arguments. Within function `ex`, output arguments may be stored in `varargout(i)`.

**EXAMPLE:**

```scilab
def('varargout=exampl()','varargout=list(1,2,3,4)')
x=exampl()
[x,y]=exampl()
[x,y,z]=exampl()
```

**SEE ALSO:** function 268, `varargin` 277, `list` 57
1.6 Strings manipulations
1.6.1  
code2str ______ returns character string associated with Scilab integer codes.

CALLING SEQUENCE :
str=code2str(c)

PARAMETERS :
str : a character string
c : vector of character integer codes

DESCRIPTION :
Returns character string associated with Scilab integer codes. str is such that c(i) is the Scilab integer code of part(str,i))

EXAMPLE :
code2str([-28 12 18 21 10 11])

SEE ALSO :  str2code 282

1.6.2  
convstr ________________________________ case conversion

CALLING SEQUENCE :
[y]=convstr(str-matrix, ["flag"])  

PARAMETERS :
str-matrix, y : matrices of strings
"flag" : string ("u" for upper or "l" for lower (default value))

DESCRIPTION :
converts the matrix of strings str-matrix into lower case (for "l" ;default value) or upper case (for "u").

EXAMPLE :
A=['this','is';'my','matrix'];
convstr(A,'u')

1.6.3  
emptystr ________________________________ zero length string

CALLING SEQUENCE :
s=emptystr()
s=emptystr(a)
s=emptystr(m,n)

PARAMETERS :
a : any type of matrix
s : character string matrix
m, n : integers
**DESCRIPTION:**
Returns a matrix of zero length character strings
With no input argument returns a zero length character string.
With a matrix for input argument returns a zero length character strings matrix of the same size.
With two integer arguments returns a mxn zero length character strings matrix

**EXAMPLE:**

```plaintext
x=emptystr();for k=1:10, x=x+','+string(k);end
```

**SEE ALSO:** part 282, length 281, string 284

1.6.4 **grep** find matches of a string in a vector of strings

**CALLING SEQUENCE:**

```plaintext
row=grep(str1,str2)
[row,which]=grep(str1,str2)
```

**PARAMETERS:**

- **str1**: a vector of strings.
- **str2**: a character string or character string vector. The string(s) to search in str1
- **row**: vector of indices: row where a match has been found or an empty matrix if no match found.
- **which**: vector of indices: index of str2 string found or an empty matrix if no match found.

**DESCRIPTION:**
Foreach entry of str1, grep searches if at least a string in str2 matches a substring. str1 entries
index where at least a match has been found are returned in the row argument. while optionnal which argument gives the index of first string of str2 found.

**EXAMPLE:**

```plaintext
txt=['find matches of a string in a vector of strings'
     'search position of a character string in an other string'
     'Compare Strings'];

grep(txt,'strings')
grep(txt,['strings' 'Strings'])

[r,w]=grep(txt,['strings' 'Strings'])
```

**SEE ALSO:** strindex 283

1.6.5 **length** length of object

**CALLING SEQUENCE:**

```plaintext
n=length(M)
```

**PARAMETERS:**

- **M**: matrix (usual or polynomial or character string) or list
- **n**: integer or integer matrix
DESCRIPTION:
For usual or polynomial matrix \( n \) is the integer equal to number of rows times number of columns of \( M \).
(Also valid for \( M \) a boolean matrix)
For matrices made of character strings (and in particular for a character string) \( \text{length} \) returns in \( n \) the length of entries of the matrix of character strings \( M \).
The length of a list is the number of elements in the list (also given by \( \text{size} \)).
\( \text{length}('123') \) is 3. \( \text{length}([1,2;3,4]) \) is 4.
SEE ALSO: \( \text{size} 211 \)

1.6.6 part _________________________________ extraction of strings

CALLING SEQUENCE:
\[
[c]=\text{part}(mp,v)
\]

PARAMETERS:
- \( mp \), \( c \): string matrices
- \( v \): integer vector.

DESCRIPTION:
Let \( s[k] \) stands for the \( k \) character of string \( s \) (or the empty character if \( k \geq \text{length}(s) \)).
\( \text{part} \) returns \( c \), a matrix of character strings, such that \( c(i,j) \) is the string "\( s[v(1)]...s[v(n)] \)"
\( s=mp(i,j) \).

EXAMPLE:
\[
c=\text{part}(['a','abc','abcd'],[1,1,2])
\]
SEE ALSO: \( \text{string} 284 \), \( \text{length} 281 \)

1.6.7 str2code ______ return scilab integer codes associated with a character string

CALLING SEQUENCE:
\[
c=\text{str2code}(str)
\]

PARAMETERS:
- \( str \): a character string
- \( c \): vector of character integer codes

DESCRIPTION:
Return \( c \) such that \( c(i) \) is the scilab integer code of \( \text{part}(str,i) \).

EXAMPLE:
\[
\text{str2code('Scilab')}
\]
SEE ALSO: \( \text{code2str} \) 280
1.6.8  **strcat**  

**catenate character strings**

**CALLING SEQUENCE:**

\[ \text{txt}=\text{strcat(vstr [,strp])} \]

**PARAMETERS:**

- **vstr**: vector of strings
- **strp**: string, default value is the emptystr 
  
**TXT**: string

**DESCRIPTION:**

\[ \text{txt}=\text{strcat(vstr)} \]

concatenates character strings:
\[ \text{txt}=\text{vstr}(1)+...+\text{vstr}(n) \]

\[ \text{txt}=\text{strcat(vstr,strp)} \]

returns \[ \text{txt}=\text{strs}(1)+\text{strp}+...+\text{strp}+\text{strs}(n) \]. The plus symbol does the same: \[ "a"+"b" \] is the same as \[ \text{strcat}(["a","b"]) \]

**EXAMPLE:**

\[ \text{strcat(string(1:10),',',')} \]

**SEE ALSO:**  string 284,  strings 284

1.6.9  **strindex**  

**search position of a character string in an other string.**

**CALLING SEQUENCE:**

\[ \text{ind}=\text{strindex(str1,str2)} \]

**PARAMETERS:**

- **str1**: a character string. The string where to search occurrences of str2
- **str2**: a character string or character string vector. The string(s) to search in str1

**TXT**: vector of indexes

**DESCRIPTION:**

\[ \text{strindex} \] searches indexes where \( \text{str2}(i) \) is found in \( \text{str1} \) for each \( k \) it exist a isuch that \( \text{part} \left(\text{str1},\text{ind}(k)+(0:\text{length}(\text{str2}(i))-1)\right) \) is the same string than \( \text{str2}(i) \)

When str2 is a vector and str2

**EXAMPLE:**

\[ \text{k}=\text{strindex(}'SCI/demos/scicos','/'\text{)} \]
\[ \text{k}=\text{strindex(}'SCI/demos/scicos','SCI'\text{)} \]
\[ \text{k}=\text{strindex(}'SCI/demos/scicos',!'\text{)} \]
\[ \text{k}=\text{strindex(}'aaaaa','aa'\text{)} \]
\[ \text{k}=\text{strindex(}'SCI/demos/scicos',[SCI',sci']\text{)} \]

**SEE ALSO:**  string 284,  strings 284

Scilab Group  April 1993  283
1.6.10  string  ---------------------------------------- conversion to string

CALLING SEQUENCE :

string(x)
[out, in, text] = string(x)

PARAMETERS :

x : real matrix or function

DESCRIPTION :
converts a matrix into a matrix of strings.
If \( x \) is a function \([\text{out, in, text}] = \text{string}(x)\) returns three vectors strings: \text{out} is the vector of output variables, \text{in} is the vector of input variables, and \text{text} is the (column) vector of the source code of the function.
If \( x \) is a \text{lib} variable, \text{text} is a character string column vector. The first element contains the path of library file and the other the name of functions it defines.
Character strings are defined as ’string’ (between quotes) or "string" (between doublequotes); matrices of strings are defined as usual constant matrices.
Concatenation of strings is made by the + operation.

EXAMPLES :

string(rand(2,2))
deff(’y=mymacro(x),’y=x+1’)
[out, in, text] = string(mymacro)
x=123.356; ’Result is ’ + string(x)

See Also: part 282, length 281, quote 67, evstr 35, execstr 37, strsubst 285, strcat 283, strindex 283, sci2exp 307

1.6.11  strings ---------------------------------------- Scilab Object, character strings

DESCRIPTION :

Strings are defined as ’string’ (between quotes) or "string" (between doublequotes); matrices of strings are defined as usual constant matrices.
Concatenation of two strings is made by a + : string1+string2.

EXAMPLE :

[’this’,’is’; ’a 2x2’,’matrix’]
"matrix"=="mat"+"rix"

See Also:  part 282, length 281, strcat 283

1.6.12  stripblanks -------------------------------- strips leading and trailing blanks of strings

CALLING SEQUENCE :

\text{txt}=\text{stripblanks}(\text{txt})

PARAMETERS :

Scilab Group  April 1998  284
**strsubst**

**DESCRIPTION:**
stripblanks strips leading and trailing blanks of strings

**EXAMPLE:**

```matlab
txt = ' 123 '; 
'!' + txt + '!'

a = [' 123 ',' xyz']
strcat(stripblanks(a))
```

**1.6.13 strsubst *** substitute a character string by another in a character string.**

**CALLING SEQUENCE:**

```matlab
str = strsubst(str1, str2, str3)
```

**PARAMETERS:**

- **str1:** a matrix of character string. The strings where to search occurrences of `str2`
- **str2:** a character string. The string to search in `str1`
- **str3:** a character string. The replacement string.
- **str:** a matrix of character string. The result of the substitution on `str2` by `str3` in `str1`

**DESCRIPTION:**
strsubst replaces all occurrences of `str2` in `str1` by `str3`.

**EXAMPLE:**

```matlab
strsubst('SCI/demos/scicos','SCI','.')
strsubst('SCI/demos/scicos','/',' ')
```

**SEE ALSO:** string 284, strings 284
1.7 Dialogs
1.7.1 addmenu .......................... interactive button or menu definition

CALLING SEQUENCE :

addmenu(button [,submenus] [,action])
addmenu(gwin,button [,submenus] [,action])

PARAMETERS :

button : a character string. The button name. On Windows operating systems (not X window), an &
can be placed before the character in the name to be used for keyboard shortcut; this character will
be underlined on the GUI.
submenus : a vector of character string. The sub_menus items names
action : a list with 2 elements action=list(flag,proc_name)
flag : an integer (default value is 0)
flag==0 : the action is defined by a scilab instruction
flag==1 : the action is defined by a C or Fortran procedure
proc_name : a character string which gives the name of scilab variable containing the instruction or the
name of procedure to call.
gwin : integer. The number of graphic window where the button is required to be installed

DESCRIPTION :

The function allows the user to add new buttons or menus in the main window or graphics windows
command panels.

If action is not given the action associated with a button must be defined by a scilab instruction given
by the character string variable which name is

button for a main window command
button_gwin for a graphic window command

If proc_name designes a C or Fortran procedure, this procedure may be interfaced in Fortran subroutine
default/fbutn.f or dynamically linked with scilab using the link function. the calling sequence is in
C: (char* name, int* win, int *entry)
Actions associated with the kth sub_menu must be defined by scilab instructions stored in the kth element
of the character string variable which name is

button for a main window command
button_gwin for a graphic window command

EXAMPLE :

addmenu('foo')
foo='disp(''hello'')'

addmenu('Hello',['Franck';'Peter'])
Hello=['disp(''hello Franck'');'disp(''hello Peter'')']

addmenu(0,'Hello',['Franck';'Peter'])
Hello_0=['disp(''hello Franck'');'disp(''hello Peter'')']

addmenu('Bye',list(0,'French_Bye'))
French_Bye='disp(''Au revoir'')'

//C defined Callback
// creating Callback code
code=[

Scilab Function

### 1.7.2 delmenu

**interactive button or menu deletion**

**CALLING SEQUENCE:**

- `delmenu(button)`
- `delmenu(gwin, button)`

**PARAMETERS:**

- `button`: a character string. The button name. On Windows operating systems (not X window), an @ should be placed before the character in the name used for keyboard shortcut; this character is underlined on the GUI.
- `gwin`: integer. The number of graphic window where the button is required to be installed

**DESCRIPTION:**

The function allows the user to delete buttons or menus created by `addmenu` in the main or graphics windows command panels. Predefined buttons on Scilab graphic windows can also be deleted. If possible, it is better to delete first the latest created button for a given window to avoid gaps in command panels.

**EXAMPLE:**

- `addmenu('foo')`
- `delmenu('foo')`

**SEE ALSO:** `setmenu 290`, `unsetmenu 290`, `addmenu 287`

### 1.7.3 getvalue

**xwindow dialog for data acquisition**

**CALLING SEQUENCE:**

- `[ok,x1,...,x14]=getvalue(desc,labels,typ,ini)`

**SEE ALSO:** `setmenu 290`, `unsetmenu 290`, `addmenu 287`
PARAMETERS :

desc : column vector of strings, dialog general comment
labels : n column vector of strings, labels(i) is the label of the ith required value
typ : list(typ1,dim1,...,typn,dimn)
typi : defines the type of the ith value, may have the following values:
  "mat" : for constant matrix
  "col" : for constant column vector
  "row" : for constant row vector
  "vec" : for constant vector
  "str" : for string
  "lis" : for list
dimi : defines the size of the ith value it must be a integer or a 2-vector of integer, -1 stands for undefined dimension
ini : n column vector of strings, ini(i) gives the suggested response for the ith required value
ok : boolean, %t if ok button pressed, %f if cancel button pressed
xi : contains the ith value if ok=%t. If left hand side as one more xi than required values the last xi contains the vector of answered strings.

DESCRIPTION :
This function encapsulate x_mdialog function with error checking, evaluation of numerical response, ...

REMARKS :
All valid expressions can be used as answers; for matrices and vectors getvalues automatically adds [ ] around the given answer before numeric evaluation.

EXAMPLE :
labels=["magnitude";"frequency";"phase   "];
[ok,mag,freq,ph]=getvalue("define sine signal",labels,...
    list("vec",1,"vec",1,"vec",1),["0.85";"10ˆ2";"%pi/3"])

SEE ALSO:  x_mdialog 293,  x_matrix 292,  x_dialog 292

AUTHOR : S. Steer

1.7.4  halt __________________________________ stop execution

CALLING SEQUENCE :

halt()

DESCRIPTION :
stops execution until something is entered in the keyboard.

SEE ALSO:  pause 64,  return 68,  exec 36

1.7.5  havewindow ______________________________ return scilab window mode

CALLING SEQUENCE :

havewindow()

DESCRIPTION :
returns %t if scilab has it own window and %f if not, i.e. if scilab has been invoked by "scilab -nw". (nw stands for "no-window").
1.7.6  keyboard

DESCRIPTION:
Let C- stands for the control key. The following keyboard commands are available:

C-l clears the Scilab window
C-d deletes the current character
C-p calls back the preceding command
C-n go to next command line
C-a moves the cursor to the beginning of command line.
C-b backspace, moves the cursor one character to the left
C-f forwards, moves the cursor one character to the right
C-k kills command line from cursor to the end.
C-y yank, retrieves killed line.
!beg looks for last command line which begins by beg.
C-c interrupts Scilab and pause after carriage return. (Only functions can be interrupted). Clicking on the stop button enters a C-C.

SEE ALSO:  pause 64, read 254, input 238

1.7.7  setmenu

CALLING SEQUENCE:
setmenu(button [,nsub])
setmenu(gwin,button [,nsub])

PARAMETERS:
button : a character string. The button name
gwin : integer. The number of graphic window where the button is installed
nsub : integer. The number of submenu to de-activate (if any). If button has no sub-menu, nsub is ignored

DESCRIPTION:
The function allows the user to make active buttons or menus created by addmenu in the main or graphics windows command panels.

EXAMPLE:
addmenu('foo') //New button made in main scilab window
unsetmenu('foo') //button foo cannot be activated (grey string)
setmenu('foo') //button foo can be activated (black string)

SEE ALSO: delmenu 288, unsetmenu 290, addmenu 287

1.7.8  unsetmenu

CALLING SEQUENCE:
unsetmenu(button,[nsub])
unsetmenu(gwin,button,[nsub])

Scilab Group  December 1995  290
PARAMETERS:

button: a character string. The button name
button: integer. The number of graphic window where the button is installed
nsub: integer. The number of submenu to de-activate (if any). If button has no sub-menu, nsub is ignored

DESCRIPTION:
The function allows the user to desactivate buttons or menus created by `addmenu` in the main or graphics windows command panels.

EXAMPLE:

```plaintext
//addmenu('foo')
//unsetmenu('foo')
//unsetmenu('File',2)
```

SEE ALSO: `delmenu` 288, `setmenu` 290, `addmenu` 287

1.7.9  x_choices  interactive Xwindow choices through toggle buttons

CALLING SEQUENCE:

```plaintext
rep=x_choices(title,items)
```

PARAMETERS:

- title: vector of strings, title for the popup window.
- items: a list of items `items=list(item1,...,itemn)`, where each `item` is also a list of the following type: `item=list('label',default_choice,choices)`. `default_choice` is an integer which gives the default toggle on entry and `choices` is a row vector of strings which gives the possible choices.
- rep: an integer vector which gives for each item the number of the selected toggle. If user exits dialog with "cancel" button `rep` is set to `[]`.

DESCRIPTION:
Select items through toggle lists and return in `rep` the selected items

Type `x_choices()` to see an example.

EXAMPLE:

```plaintext
l1=list('choice 1',1,['toggle c1','toggle c2','toggle c3']);
l2=list('choice 2',2,['toggle d1','toggle d2','toggle d3']);
l3=list('choice 3',3,['toggle e1','toggle e2']);
rep=x_choices('Toggle Menu',list(l1,l2,l3));
```

1.7.10  x_choose  interactive Xwindow choice

CALLING SEQUENCE:

```plaintext
[num]=x_choose(items,title [,button])
```

PARAMETERS:

- items: column vector of string, items to choose
- title: column vector of string, comment for the dialog
- button: string, text to appear in the button. Default value is 'Cancel'

Scilab Group April 1993 291
**num** : integer, chosen item number or 0 if dialog resumed with "Cancel" button

**DESCRIPTION**:
Returns in **num** the number of the chosen item.

**EXAMPLE**:

```scilab
n=x_choose(['item1';'item2';'item3'],[['that is a comment';'for the dialog']])
n=x_choose(['item1';'item2';'item3'],[['that is a comment'],['Return']])
```

**SEE ALSO**: `x_choices 291`, `x_mdialog 293`, `getvalue 288`, `unix_g 310`

### 1.7.11 x_dialog  Xwindow dialog

**CALLING SEQUENCE**:

```scilab
result=x_dialog(labels,valueini)
```

**PARAMETERS**:

- `labels` : column vector of strings, comment for dialog
- `valueini` : n column vector of strings, initial value suggested

**DESCRIPTION**:
Creates an X-window multi-lines dialog

**EXAMPLE**:

```scilab
//gain=evstr(x_dialog('value of gain ?','0.235'))
//x_dialog(['Method';'enter sampling period'],''1'')
//m=evstr(x_dialog('enter a 3x3 matrix ','[[0 0 0';'0 0 0';'0 0 0]'])))
```

**SEE ALSO**: `x_mdialog 293`, `x_matrix 292`, `evstr 35`, `execstr 37`

### 1.7.12 x_matrix  Xwindow editing of matrix

**CALLING SEQUENCE**:

```scilab
[result]=x_matrix(label,matrix-init)
```

**PARAMETERS**:

- `label` : character string (name of matrix)
- `matrix-init` : real matrix

**DESCRIPTION**:
For reading or editing a matrix.

**EXAMPLE**:

```scilab
//m=evstr(x_matrix('enter a 3x3 matrix ',rand(3,3)))
```

**SEE ALSO**: `x_mdialog 293`, `x_dialog 292`
1.7.13  \texttt{x_mdialog} \hfill \texttt{Xwindow dialog}

\textbf{CALLING SEQUENCE:}

\begin{verbatim}
result=x_mdialog(title,labels,default_inputs_vector)
result=x_mdialog(title,labelsv,labelsh,default_input_matrix)
\end{verbatim}

\textbf{PARAMETERS:}

\begin{description}
\item \texttt{title} : column vector of strings, dialog general comment
\item \texttt{labels} : \textbf{n} column vector of strings, \texttt{labels(i)} is the label of the \textbf{i}th required value
\item \texttt{default_input_vector} : \textbf{n} column vector of strings, \texttt{default_input_vector(i)} is the initial value of the \textbf{i}th required value
\item \texttt{labelsv} : \textbf{n} vector of strings, \texttt{labelsv(i)} is the label of the \textbf{i}th line of the required matrix
\item \texttt{labelsh} : \textbf{m} vector of strings, \texttt{labelsh(j)} is the label of the \textbf{j}th column of the required matrix
\item \texttt{default_input_matrix} : \textbf{n x m} matrix of strings, \texttt{default_input_matrix(i,j)} is the initial value of the \textbf{(i,j)} element of then required matrix
\item \texttt{result} : \textbf{n x m} matrix of string if returned with "Ok" button or [] if returned with "Cancel" button
\end{description}

\textbf{DESCRIPTION:}

\texttt{X-window vector/matrix interactive input function}

\textbf{EXAMPLES:}

\begin{verbatim}
txt=[‘magnitude’;’frequency’;’phase ’];
sig=x_mdialog(‘enter sine signal’,txt,[’1’;’10’;’0’])
mag=evstr(sig(1))
frq=evstr(sig(2))
ph=evstr(sig(3))

rep=x_mdialog([‘System Simulation’;’with PI regulator’],...’P gain’;’I gain ’,[’ ’;’ ’])

n=5;m=4;mat=rand(n,m);
row=’row’;labelv=row(ones(1,n))+string(1:n)
col=’col’;labelh=col(ones(1,m))+string(1:m)
new=evstr(x_mdialog(‘Matrix to edit’,labelv,labelh,string(mat)))
\end{verbatim}

\textbf{SEE ALSO:} \texttt{x_dialog} 292, \texttt{x_choose} 291, \texttt{x_message} 293, \texttt{getvalue} 288, \texttt{evstr} 35, \texttt{execstr} 37

1.7.14  \texttt{x_message} \hfill \texttt{X window message}

\textbf{CALLING SEQUENCE:}

\begin{verbatim}
[num]=x_message(strings [,buttons])
\end{verbatim}

\textbf{PARAMETERS:}

\begin{description}
\item \texttt{strings} : vector of characters strings to be displayed
\item \texttt{buttons} : character string or 2 vector of character strings which specifies button(s) name(s). Default value is "Ok"
\item \texttt{num} : number of button clicked (if 2 buttons are specified)
\end{description}
**DESCRIPTION:**
for displaying a message (diagnostic,...) and waiting for an answer (button click). The function returns only after a click on a button.

**EXAMPLES:**

```scilab
gain=0.235;x_message('value of gain is :'+string(gain))
x_message(['singular matrix';'use least squares'])

r=x_message(['Your problem is ill conditioned';
    'continue ?'],['Yes','No'])
```

**SEE ALSO:**  x_dialog 292, x_mdialog 293, x_message_modeless 294

1.7.15  **x_message_modeless**  ---------------- X window modeless message

**CALLING SEQUENCE:**

```
x_message_modeless(strings)
```

**PARAMETERS:**

`strings`: vector of characters strings to be displayed

**DESCRIPTION:**
for displaying a message (information, user-guide...). The function returns immediately. The message window is killed when "Ok" button is clicked.

**EXAMPLES:**

```scilab
x_message_modeless(['This is a modeless message'
    'Scilab may continue computation'
    'Click on "Ok" to close the message'])
x_message_modeless('Now two message windows are opened')
```

**SEE ALSO:**  x_dialog 292, x_mdialog 293, x_message 293
1.8 Utilities
1.8.1 %helps ___________________ Variable defining the path of help directories

DESCRIPTION:
The variable %helps is an N x 2 matrix of strings. The kth row of %helps, %helps(k,:) represents the kth chapter of the manual and is made of two strings:
%helps(k,1) is a pathname for a help directory.
%helps(k,2) is a title for this help directory. For instance, for k=2, we have the graphics chapter
%helps(2,:).
The variable %helps is defined in the Scilab startup file SCI+"/scilab.star". To add a new help directory, the user should add a row to the variable %helps. (One row for each directory).
For instance, %helps=[%helps; "Path-Of-My-Help-Dir", "My-Title"]; enables the Scilab help browser to look for help manual items in the directory with pathname "Path-Of-My-Help-Dir".
"My-Title" is then the title of a new help chapter which appears in the bottom part of the help window, raised by clicking on the help button.

A valid help directory must contain:
1- A set of .cat files (e.g. item1.cat, item2.cat etc). The .cat files do do require special format. Usually, they are built as Unix man pages.
2- A whatis file, which must have a special format. Each row of the whatis must be as follows:

item - what is item @item

item is the item of the help, i.e. the command help item returns the contents of the file item.cat.
what is item is a brief description of the item.
The whatis file appears in the top window of the help window, once a chapter has been selected in the bottom window.
Clicking on one item of the top window opens the manual page.
The command apropos keyword returns the row(s) of all the whatis file(s) in which the keyword appears.

On Unix-Linux platforms Scilab provides a Makefile for transforming .man pages into .cat pages (see SCIDIR/examples/man-examples).

SEE ALSO: help 298, apropos 26

1.8.2 G_make ___________________________ call make or nmake

CALLING SEQUENCE:
Rfiles=G_make(files,dllname)

PARAMETERS:
files : a character string or a vector of character string.
dllname : a character string.
Rfiles : vector of character string. Rfiles can be used as a first argument when calling addinter function.

DESCRIPTION:
On Unix like systems (i.e. unix or windows/gcwin32) G_make calls the make utility for building target files and returns the value of files in the variable Rfiles. On windows platforms, (i.e when Scilab was compiled with Microsoft VisualC++). G_make calls the nmake utility for building target dllname and it returns the value of dllname in the variable Rfiles. Of course G_make will work if appropriate Makefiles are provided in the current Scilab directory.
G_make can be used to provide OS independant call to addinter. and such examples can be found in the directory SCIDIR/examples/addinter-examples

Scilab Group April 1993 296
files=G_make([TMPDIR+'/ex1cI.o',TMPDIR+'/ex1c.o'],'ex1c.dll');// compilation
addinter(files,'foobar','foubare'); // link

SEE ALSO: addinter 265

1.8.3 c_link ______________________________________ check dynamic link

CALLING SEQUENCE:

c_link(routine-name)
[test,ilib]=c_link(routine-name)
test=c_link(routine-name,num)

PARAMETERS:

routine-name: a character string
num:
test: boolean, indicates if there is a shared library which contains routine-name.
ilib: a scalar, the number of the shared library which contains routine-name

DESCRIPTION:

c_link is a boolean function which checks if the routine routine-name is currently linked. This function returns a boolean value true or false. When used with two return values, the function c_link returns a boolean value in test and the number of the shared library which contains routine-name in ilib (when test is true).

EXAMPLE:

if c_link('foo') then link('foo.o','foo');end
// to unlink all the shared libraries which contain foo
a=%t; while a;[a,b]=c_link('foo'); ulink(b);end

SEE ALSO: link 303, fort 43

1.8.4 chdir ________________________________ changes Scilab current directory

CALLING SEQUENCE:

ierr=chdir('path-name')

PARAMETERS:

ierr: an integer, 1 if chdir failed to change directory and 0 elsewhere.

DESCRIPTION:

Change the current Scilab directory to ‘path-name’

EXAMPLE:

chdir(TMPDIR);
if MSDOS then
    unix_w("dir");
else
    unix_w("ls");
end

SEE ALSO: getcwd 67

Scilab Group April 1993 297
1.8.5 \texttt{dec2hex} \quad \text{hexadecimal representation of integers}

\textbf{CALLING SEQUENCE:}

\begin{verbatim}
h=dec2hex(d)
\end{verbatim}

\textbf{PARAMETERS:}

d : matrix of non negative integers
h : matrix of character strings

\textbf{DESCRIPTION:}
\deichex(x) returns the hexadecimal representation of a matrix of integers

\textbf{EXAMPLE:}

\begin{verbatim}
dec2hex([2748 10; 11 3])
\end{verbatim}

1.8.6 \texttt{demos} \quad \text{guide for scilab demos}

\textbf{CALLING SEQUENCE:}

demos()

\textbf{DESCRIPTION:}
\demos() is an interactive guide to execute various scilab demonstrations The source code of each demo is in the directory SCIDIR/demos/...

1.8.7 \texttt{help} \quad \text{on-line help command}

\textbf{CALLING SEQUENCE:}

\begin{verbatim}
help word
\end{verbatim}

\textbf{DESCRIPTION:}
To each documented word corresponds a \texttt{word.cat} ascii file. these files are organised within directories (chapters). Each chapter must contain *.cat files and a \texttt{whatis} file with one line for each documented word in the chapter. Each line must have the following format:

\begin{verbatim}
word - quick description
\end{verbatim}

List of chapter directories is given in a scilab variable \texttt{%helps}. \texttt{%helps} a two column matrix of strings. Each line is formed as follow:

\begin{verbatim}
chapter_path  chapter_title
\end{verbatim}

\texttt{%helps} default value is set by scilab.star file. If you want to add new help chapters you have to add new lines \texttt{%helps} variables for example in your .scilab startup file.

Note that, once the help has been used, new additions to \texttt{%helps} are not taken into account.

See also Scilab’s manual

\textbf{SEE ALSO:} apropos \texttt{26}, man \texttt{305}, formatman \texttt{660}

Scilab Group \quad April 1993 \quad 298
1.8.8 hex2dec ______ converts hexadecimal representation of integers to numbers

CALLING SEQUENCE :

d=hex2dec(h)

PARAMETERS :

d : matrix of integers
h : matrix of character strings corresponding to hexadecimal representation

DESCRIPTION :

hex2dec(x) returns the matrix of numbers corresponding to the hexadecimal representation.

EXAMPLE :

hex2dec(['ABC','0','A'])

1.8.9 ilib_build ____________________________ utility for shared library management

CALLING SEQUENCE :

ilib_build(lib_name,table,files,libs [,makename])

PARAMETERS :

lib_name : a character string, the generic name of the library without path and extension.
table : 2 column string matrix giving the table of pairs 'scilab-name','interface name'
files : string matrix giving objects files needed for shared library creation
libs : string matrix giving extra libraries needed for shared library creation
makename : character string. The path of the Makefile file without extension.

DESCRIPTION :

This tool is used to create shared libraries and to generate a loader file which can be used to dynamically load the shared library into Scilab with addinter. Many examples are provided in examples/interface-tour-so directory.

EXAMPLE :

//Here with give a complete example on adding new primitive to Scilab
//create the procedure files
f1=['extern double fun2();'
   'void fun1(x,y)'
   'double *x, *y;'
   '{*y=fun2(*x)/(*x);}'];

mputl(f1,'fun1.c')

f2=['#include <math.h>'
   'double fun2(x)'
   'double x;'
   '{ return( sin(x+1.));}'];

mputl(f2,'fun2.c');

//creating the interface file
i=['#include "stack-c.h"
  
  extern int fun1 _PARAMS(( double *x, double *y));
  
  int intfun1(fname)
  
  char * fname;
  
  {
    int m1,n1,l1;
    CheckRhs(1,1);
    CheckLhs(1,1);
    GetRhsVar(1, "d", &m1, &n1, &l1);
    fun1(stk(l1),stk(l1));
    LhsVar(1) = 1;
    return 0;
  }];

mputl(i,'intfun1.c')

// creating the shared library (a gateway, a Makefile and a loader are
// generated.

files=['fun1.o','fun2.o','intfun1.o'];
ilib_build('foo',['scifun1','intfun1'],files,[]);

// load the shared library
exec loader.sce

// using the new primitive
scifun1(33)

SEE ALSO:  addinter 265, link 303, ilib_compile 300, ilib_gen_Make 302, ilib_gen_gateway 302, ?? ilib_gen_loader, ilib_for_link

1.8.10  ilib_compile  ilib_build utility: executes the makefile produced by
         ilib_gen_Make

CALLING SEQUENCE:

libn=ilib_compile(lib_name,makename)

PARAMETERS:

lib_name :: a character string, the generic name of the library without path and extension.
makename : character string. The path of the Makefile file without extension.
libn : character string. The path of the really generated shared library file.

DESCRIPTION:

Utility function used by ilib_build
This executes the makefile produced by ilib_gen_Make, compiles the C and fortran files and generates
the shared library.
Shared libraries can then be used with the link and addinter scilab function for incremental linking.

SEE ALSO:  addinter 265, link 303, ilib_build 299, ilib_gen_Make 302, ilib_gen_gateway 302, ?? ilib_gen_loader, ilib_for_Link

Scilab Group  May 2000
1.8.11 ilib_for_link __________ utility for shared library management with link

CALLING SEQUENCE:

libn=ilib_for_link(names,files,libs,flag,makename,loadername)

PARAMETERS:

names : a string matrix giving the entry names which are to be linked.
files : string matrix giving objects files needed for shared library creation
libs : string matrix giving extra libraries needed for shred library creation
flag : a string flag ("c" or "f") for C or Fortran entry points.
makename : character string. The pathname of the Makefile file without extension (default value Makelib).
loadername : character string. The pathname of the loader file (default value loader.sce).
libn : character string. The path of the really generated shared library file.

DESCRIPTION:

This tool is used to create shared libraries and to generate a loader file which can be used to dynamically load the shared library into Scilab with the link function. New entry points given by names are then accessible through the call function or with non linear tools ode, optim,...

Many examples are provided in examples/link-examples-so directory.

EXAMPLE:

f1=['int extlc(n, a, b, c)'
       'int *n; double *a, *b, *c;'
       '{int k;'
       ' for (k = 0; k < *n; ++k) '
       '   c[k] = a[k] + b[k];'
       ' return(0);}'];

mputl(f1,'fun1.c')

//creating the shared library (a gateway, a Makefile and a loader are //generated.

ilib_for_link('extlc','fun1.o',[],"c")

// load the shared library

exec loader.sce

//using the new primitive
a=[1,2,3];b=[4,5,6];n=3;
c=call('extlc',n,1,'i',a,2,'d',b,3,'d','out',[1,3],4,'d');
if norm(c-(a+b)) > %eps then pause,end

SEE ALSO: addinter 265, link 303, ilib_compile 300, ilib_gen_Make 302, ilib_gen_gateway 302, ?? ilib_gen_loader, ilib_for_link
1.8.12  lib_gen_Make ______ utility for lib_build: produces a makefile for building shared libraries

CALLING SEQUENCE :

\texttt{Makename=lib_gen\_Make(name,files,libs,makename [,with\_gateway])}

PARAMETERS :

lib\_name : a character string, the generic name of the library without path and extension.
files : a vector of character string. The names of the C or Fortran files without the extension and the path part.
libs : a vector of character string. Additional libraries paths or [].
makename : character string. The path of the Makefile file.
with\_gateway : a boolean. If true a file with name <lib\_name>_gateway is added.
Makename : character string. The path of the really generated Makefile file.

DESCRIPTION :
Utility function used by \texttt{lib\_build} This function generates a makefile adapted to the Operating System for building shared libraries to be loaded in Scilab. Proper options and paths are set. Shared libraries can then be used with the \texttt{link} and \texttt{addinter} scilab function for incremental linking. The shared library is built from a set of C or Fortran routines stored in a directory and if required from a set of external libraries. Files are not required to exist, when makefile is generated, but of course are required for executing the makefile.

See Also: \texttt{addinter 265, link 303, lib\_build 299, lib\_compile 300, lib\_gen\_gateway 302, ?? lib\_gen\_loader, lib\_for\_link}

1.8.13  lib_gen_gateway __________ utility for lib_build, generates a gateway file.

CALLING SEQUENCE :

\texttt{lib_gen\_gateway(name,table)}

PARAMETERS :

name : a character string, the generic name of the library without path and extension.
table : 2 column string matrix giving the table of pairs 'scilab-name' 'interface name'

DESCRIPTION :
Utility function used by \texttt{lib\_build} This function generates a gateway file used by \texttt{addinter}.

See Also: \texttt{addinter 265, link 303, lib\_build 299, lib\_compile 300, lib\_gen\_Make 302, ?? lib\_gen\_loader, lib\_for\_link}

1.8.14  lib_gen_loader __________ utility for lib_build: generates a loader file

CALLING SEQUENCE :

\texttt{lib_gen\_loader(name,table)}

PARAMETERS :

Scilab Group  

April 1993
name : a character string, the generic name of the library without path and extension.
table : 2 column string matrix giving the table of pairs 'scilab-name' 'interface name'

DESCRIPTION :
Utility function used by ilib_build This function generates a loader file.

SEE ALSO: addinter 265, link 303, ilib_build 299, ilib_compile 300, ilib_gen_Make 302, ?? ilib_gen_loader, ilib_for_link

1.8.15 intersci scilab tool to interface C of Fortran functions with scilab

DESCRIPTION :
All scilab primitive functions are defined in a set of interface routines. For each function the interfacing code checks first number of rhs and lhs arguments. Then it get pointers on input arguments in the Scilab data base and checks their types. After that it calls procedure associated with Scilab functions, checks returned errors flags and set the results in the data base.
intersci is a program which permits to interface automatically FORTRAN subroutines or C functions to Scilab
With intersci, a user can group all his FORTRAN or C code into a same set, called an interface, and use them in Scilab as Scilab functions. The interfacing is made by creating a FORTRAN subroutine which has to be linked to Scilab together with the user code. This complex FORTRAN subroutine is automatically generated by intersci from a description file of the interface.
Refer to intersci documentation for more details.

SEE ALSO: fort 43, external 38, addinter 265

1.8.16 link dynamic link

CALLING SEQUENCE :

link(files, sub-name)
link(files, sub-name, flag)
lst=link(‘show’)
// Link extensions for machines using ‘dlopen’
// (sun-solaris/linux-elf/alpha/hppa)
x=link(files [, sub-names,flag]);
link(x , sub-names [, flag]);
ulink(x)

PARAMETERS :
files : a character string or a vector of character strings. ld files used to define the new entry point (compiled routines, user libraries, system libraries,..)
sub-name : a character string. Name of the entry point in files to be linked.
sub-names : a character string or a vector of character strings. Name of the entry points in files to be linked.
x : an integer which gives the id of a shared library linked into Scilab with a previous call to link.
flag : character string ‘f’ or ‘c’ for Fortran (default) or C code.
names : a vector of character string. Names of dynamically linked entry points.

DESCRIPTION :
link is a dynamic link facility: this command allows to add new compiled Fortran or C routines to Scilab executable code. Linked routines can be called interactively by the function fort. Linked routines can
also be used as "external" for e.g. non linear problem solvers (ode, optim, intg, dassl...). Here are some examples:

The command link('foo.o','foo','f') links the Fortran object file foo.o with the entry point foo.

The command link('foo.o','foo','c') links the C object file foo.o with the entry point foo.

The command link('SCIDIR/libs/calelm.a','dcopy') links the Fortran routine dcopy in the library calelm.a.

A routine can be linked several times and can be unlinked with ulink. Note that, on some architectures (the ones on which ulink exists) when a routine is linked several times, all the version are kept inside Scilab.

Used with no arguments, link() returns the current linked routines.

If Scilab is compiled with static link (this is the default for SystemV machines) you may have to include the system libraries in the "link" command.

For example, if foo.o defines the object code of a routine named foo, you will use link in one of the following way:

link('foo.o','foo')
link('foo.o -lm -lc','foo','c')
link('foo.o -lfor -lm -lc','foo')
link('foo.o -lftn -lm -lc','foo')
link('foo.o -L/opt/SUNWspro/SC3.0/lib/lib77 -lm -lc','foo')

If Scilab compiled with the "shared" option, the first example can be used even if a warning for unresolved references is issued.

(Experienced) users may also link a new Scilab interface routine to add a set of new functions. See Intersci documentation for interface generation and addinter function.

REMARKS:

IBM: For IBM-RS6000 only one program can be dynamically linked.

Demo: When running a demo, you may have some trouble with the link due to slight differences between systems. In this case, you modify the demo by adding the needed libraries in the link command.

dlopen: For machines using dlopen functionality extended command can be used. a call to link returns an integer which gives the id of the shared library which is loaded into Scilab. This number can then be used as the first argument of the link function in order to link additional function from the linked shared library. The shared library is removed with the ulink command.

For example to link functions f and g form binary file test.o the two following command can be used:

link('test.o',['f','g'])

or

x=link('test.o','f');
link(x,'g');

But

link('test.o','f');
link('test.o','g');

will also work but f and g will be loaded from two different shared libraries and won't be able to share data.

show: The command lst=link('show') will report information about linked shared libraries and linked functions. The return value of the function lst is 1 or 0. If the return value is 1 then the extended calling sequence described as Link extensions for machines using "dlopen" are accepted.
unlink : (dlopen version) If the function \( f \) is changed and one wants to link the new version, it is necessary to use unlink to get rid of previous loaded versions of the function \( f \)

\[
x = \text{link}('test.o','f');
// if I need to reload a new definition of \( f \) a call to unlink
// is necessary.
\text{unlink}(x);
\text{link}('test.o','f');
\]

scilab symbols: In order to load a symbol from the Scilab code on can use

\[
\text{link}("Scilab",['Scilab-entry-point'])
\]

This does not work on all architectures. On some machines, on can link a Scilab internal function after a first call to link (with a default binary file)

\[
\text{link}("test.o",['Scilab-entry-point'])
\]

Note that with \text{dld} (Linux machine aout) you can use an empty string

\[
\text{link}(" ",['Scilab-entry-point'])
\]

\[
\text{SEE ALSO: } \text{fort 43, c_link 297, addinter 265}
\]

1.8.17 man __________________________ on line help source file description format

DESCRIPTION:
The on line help source files are written in a subset of the old text formatting language \text{groff}. The advantage of this language is to allow the most useful formatting, being quite simple. It also allows easy translation in other formatting language like HTML or LaTeX.

Source files (with extension \text{.man}) can be found in the <SCIDIR>/man/* directories. The file name is associated to the keyword (usually a function name) it describes.

Most \text{groff} commands begin by a dot as first character of a line. The first line of each man file must be:

\[
.TH <keyword> 1 "date" "Author" "Category"
\]

The help is then split into a sequence of "sections". Each section begins with the \text{groff} command \text{.SH <section name>}. To give a uniform aspect to the scilab on-line help Scilab uses the following sections names:

NAME : This section gives the keyword(s) documented in this file, on keyword per line followed by a minus sign and a short description of the keyword. Below is the example of the NAME section extracted from the mfscanf.man file.

\[
\begin{align*}
\text{mscanf} & \; - \; \text{interface to the C scanf function} \\
\text{mfscanf} & \; - \; \text{interface to the C fscanf function} \\
\text{msscanf} & \; - \; \text{interface to the C sscanf function}
\end{align*}
\]

CALLING SEQUENCE : Here one gives the Scilab syntaxe(s) which can be used. Below is an example extracted out of the mfscanf.man file:
.nf
[n,v_1,...v_n]=mfscanf(fd,format)
L=mfscanf(fd,format)

[n,v_1,...v_n]=mscanf(format)
L=mscanf(format)

[n,v_1,...v_m]=msscanf(format,str)
L=msscanf(format)
.fi

The .nf, .fi pair of commands is used to specify an unformatted region.
PARAMETERS: Here is given an indented itemized sequence of the parameters description first item is defined using the .TP n command, where n is a number of characters. The following items begin with the command .TP as shown on the following example extracted out of the file.man file:

.TP 10
file-name
: string, file name of the file to be opened
.TP
status
: string, The status of the file to be opened
.RS
.TP 5
"new"
: file must not exist new file (default)
.TP
"old"
: file must already exists.
.RE
.TP
access
: string, The type of access to the file

The .RS .RE pair of commands are used to nest an itemized sequence in an other.
DESCRIPTION: Here one describes the keyword
EXAMPLE: This section gives some examples of the use of the keyword. This must be a sequence of Scilab instruction enclosed in a .nf, .fi pair of commands.
SEE ALSO: Here are given a one line sequence of related keywords separated by a comma followed by a blank as in the example below.

mclose, meof, mfprintf, fprintfMat, mfscanf, fscanfMat, mget, mgetstr

AUTHOR: Here one gives informations on the author and references.

The fonts changes are specified in the formatted text by groff commands like

\fv for verbatim
\fr for Roman
Below is an example extracted out of the mfscanf.man file, the keywords are required to be in verbatim font.

---------------------------------------------------------------------
The \fVmfscanf\fR function reads characters from the stream \fVfd\fR.
The \fVmscanf\fR function reads characters from Scilab window.
The \fVmsscanf\fR function reads characters from the string \fVstr\fR.
---------------------------------------------------------------------

The \fL command is used to define a line break.

Some time it is useful to define tables. An example of such table formatting may be found in the file overloading.man:

\begin{verbatim}
    .TS
    tab(@);
    1  1.
    string@c polynomial@p
    function@m constant@s
    list@l
tlist@<tlist_type>
    boolean@b
    sparse@sp
    boolean sparse@spb
    .TE
\end{verbatim}

These commands define a two column, left aligned, table. The column separator (@) is specified here by the \fVtab(\fD)\fD command.

**FORMATTING MAN FILES:**
These source files may then be used to generate pure ASCII or LaTeX or HTML formatted files using the Scilab function \fVformatman\fR.

**SEE ALSO:** formatman 660, help 298

1.8.18  **sci2exp** converts variable to expression

**CALLING SEQUENCE:**

t=sci2exp(a [,nam] [,lmax])

**PARAMETERS:**

a : a scilab variable, may be
  - constant,
  - polynomial
  - string matrix
  - list
  - boolean matrix

name : character string
t : vector of string, contains the expression or instruction definition
lmax : integer, contains the maximum line length. default value is 90, lmax=0 indicate no line length control a single string is returned

DESCRIPTION :
sci2exp converts variable to an instruction if nam is given or to an expression.

EXAMPLE :

```
 a=[1 2;3 4]
 sci2exp(a,'aa')
 sci2exp(a,'aa',0)
 sci2exp(ssrand(2,2,2))
 sci2exp(poly([1 0 3 4],'s'),'fi')
```

1.8.19 sci2map ______________________ Scilab to Maple variable conversion

CALLING SEQUENCE :

txt=sci2map(a,Map-name)

PARAMETERS :

a : Scilab object (matrix, polynomial, list, string)
Map-name : string (name of the Maple variable)
txt : vector of strings containing the corresponding Maple code

DESCRIPTION :

Makes Maple code necessary to send the Scilab variable a to Maple: the name of the variable in Maple is Map-name. A Maple procedure maple2scilab can be found in SCIDIR/maple directory.

EXAMPLE :

```
 txt=[sci2map([1 2;3 4],'a');
 sci2map(%s^2+3*s+4,'p')]
```

1.8.20 scilab _______ Major unix script to execute Scilab and miscellaneous tools

CALLING SEQUENCE :

```
scilab [-ns -nw -display display -f file -args arguments]
scilab -help [ <key> ]
scilab -k <key>
scilab -xk <key>
scilab -link <objects>
```

DESCRIPTION :

```
scilab [-ns -nw -display display -f path -args arguments] : run scilab.
-ns :if this option is present the startup file SCI/scilab.star is not executed.
-nw :if this option is present then scilab is not run in an X window.
-f file :if this option is present then Scilab script file is executed first into Scilab.
-args arguments :if this option is present arguments are passed to Scilab. They can then be got by sciargs function. For multi arguments passing use a quoted, blank separated sequence of words like:
    scilab -args 'foo1 foo2'
```

Scilab Group

May 1994

308
scilab -help <key> : write on-line documentation about <key> (usually automatically called by
scilab command "help <key>"). Example:

    scilab -help plot3d

scilab -k <key> : gives the list of Scilab commands containing the keyword <key> in their de-
scription (same as UNIX command man -k)
scilab -xk <key> : gives the list of Scilab commands containing the keyword <key> in their de-
scription in a X window.
scilab -link <objects> : Is used to produce a local scilex (executable code of Scilab) linked
with the additional files given by the user in <objects>.
    If, in the list of object files, some names are known from SCIDIR/routines/default, then the
    scilex default files are omitted and replaced with the given ones.
    This command also produces an xscilab script, which when called will ran the new generated
    scilex file.

For example:

    scilab -link C/interf.o C/evol.o C/bib.a

    will create a new scilex file in which the default interf.o file will be replaced by C/interf.o.

1.8.21 scilink .................................. Unix script to relink Scilab

CALLING SEQUENCE :

    scilink <object-files>

DESCRIPTION :
    This script is used to produce a local scilex (executable code of Scilab) linked with the additional files
given by the user in <object-files>.
    If in the list of object files some names are known scilex names (from SCIDIR/routines/default) then the
    scilex default files are omitted and replaced with the given ones.
    This script also produces an xscilab script, which when called will ran the new generated scilex file.
    For example the command

    scilink C/interf.o C/evol.o C/bib.a

    will create a new scilex file in which the default interf.o file will be replaced by C/interf.o.

SEE ALSO: link 303, addinter 265

1.8.22 ulink ................……. unlink a dynamically linked shared object

CALLING SEQUENCE :

    ulink(x)

DESCRIPTION :
    see link

SEE ALSO: link 303
1.8.23  unix  shell (sh) command execution

CALLING SEQUENCE:

stat=unix(command-name)

PARAMETERS:

command-name : A character string containing Unix sh instruction
stat : An integer flag

DESCRIPTION:

Sends a string command-name to Unix for execution by the sh shell. Standard output and standard errors of the shell command are written in the calling shell. stat gives -1 if unix can’t be called (Not enough system memory available) or the sh return code.

EXAMPLE:

unix("ls $SCI/demos");
deff('wd=dir()', 'if MSDOS then unix(''cd>''+TMPDIR+''/path''); else unix(''pwd>''+TMPDIR+''/path''); end.;
    wd=read(TMPDIR+''/path'',''i'' ,1,1,''(a)'')')
wd=dir()

SEE ALSO:  edit 267 , manedit 239 , unix_g 310 , unix_s 311 , unix_w 311 , unix_x 312 , host 48

1.8.24  unix_g  shell (sh) command execution, output redirected to a variable

CALLING SEQUENCE:

rep=unix_g(cmd)

PARAMETERS:

cmd : a character string
rep : a column vector of character strings

DESCRIPTION:

Sends a string cmd to Unix for execution by the sh shell. The standard output is redirected to scilab variable rep. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

EXAMPLE:

if MSDOS then unix_g(''dir '+WSCI+'\demos''); else unix_g('ls $SCI/demos'); end
deff('wd=pwd()', 'if MSDOS then wd=unix_g(''cd''); else wd=unix_g(''pwd''); end')
wd=pwd()

SEE ALSO:  edit 267 , manedit 239 , unix_s 311 , unix_w 311 , unix_x 312 , unix 310
1.8.25  **unix_s**  ----------------- shell (sh) command execution, no output

**CALLING SEQUENCE:**

`unix_s(cmd)`

**PARAMETERS:**

`cmd` : a character string

**DESCRIPTION:**

Sends a string `cmd` to Unix for execution by the sh shell. The standard output is redirected to `/dev/null`. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by `;` is sent: this is not recommended.

**EXAMPLE:**

```plaintext
if MSDOS then
  unix_s("del foo");
else
  unix_s("rm -f foo");
end
```

**SEE ALSO:** edit 267, manedit 239, unix_g 310, unix_w 311, unix_x 312, unix 310

1.8.26  **unix_w**  ----------------- shell (sh) command execution, output redirected to scilab window

**CALLING SEQUENCE:**

`rep=unix_w(cmd)`

**PARAMETERS:**

`cmd` : a character string
`rep` : a column vector of character strings

**DESCRIPTION:**

Sends a string `cmd` to Unix for execution by the sh shell. The standard output is redirected to scilab window. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by `;` is sent: this is not recommended.

**EXAMPLE:**

```plaintext
if MSDOS then unix_w("dir +WSCI+\demos");
else unix_w("ls $SCI/demos"); end
```

**SEE ALSO:** edit 267, manedit 239, unix_g 310, unix_s 311, unix_x 312, unix 310
CALLING SEQUENCE:

unix_x(cmd)

PARAMETERS:

cmd : a character string

DESCRIPTION:
Sends a string cmd to Unix for execution by the sh shell. The standard output is redirected to a xless window. Unix execution errors are trapped; *NOTE* that only the last shell command error is reported when a list of command separated by ";" is sent: this is not recommended.

EXAMPLE:

if MSDOS then unix_x("dir "+WSCI+"\demos");
else unix_x("ls $SCI/demos"); end

SEE ALSO: edit 267, manedit 239, unix_g 310, unix_s 311, unix_w 311, unix 310
1.9 Time and date
1.9.1 date ____________________________ Current date as date string

**CALLING SEQUENCE:**

dt=date()

**PARAMETERS:**

dt : a string

**DESCRIPTION:**
dt=date() returns a string containing the date in dd-mmm-yyyy format.

**EXAMPLES:**
date()

**SEE ALSO:** getdate 314, timer 315

1.9.2 getdate ____________________________ get date and time information

**CALLING SEQUENCE:**

dt=getdate()
x=getdate('s')
dt=getdate(x)

**PARAMETERS:**

dt : an integer vector with 8 entries (see below)
x : an integer containing a date coded in second from 1 Jan 1970

**DESCRIPTION:**
dt =getdate() returns the current date in format given below:

- dt(1) : The year as a number (with the century) between 0000 and 9999.
- dt(2) : The month of the year as a number between 01 and 12.
- dt(3) : The ISO 8601 week number as a number between 01 and 53.
- dt(4) : The Julian day of the year as a number between 001 and 366.
- dt(5) : Specifies the weekday as a decimal number [1,7], with 1 representing Sunday.
- dt(6) : The day of the month as a number between 01 and 31.
- dt(7) : The hour of the day as a number between 00 and 23.
- dt(8) : The minute as a number between 00 and 59.
- dt(9) : The second is output as a number between 00 and 61.

x =getdate('s'), returns a scalar with the number of seconds since Jan 1, 1970, 00:00 UTC (unix time convention)
dt =getdate(x), format the date given by x (number of seconds since Jan 1, 1970, 00:00 UTC) in format given above:

**EXAMPLES:**
w=getdate()
mprintf("Year:%d, Month:%d, Day:%d", w(1), w(2), w(6));

x=getdate('s')
getdate(x)

**SEE ALSO:** date 314, timer 315
### 1.9.3 timer

**CALLING SEQUENCE:**

```plaintext
timer()
```

**DESCRIPTION:**

Returns the CPU time from the preceding call to `timer()`.

**EXAMPLE:**

```plaintext
timer(); A = rand(100, 100); timer()
```

**SEE ALSO:** `unix_g` 310
Chapter 2

Specialized Toolboxes

2.1 General System and Control macros
2.1.1 abcd ------------------------------------- state-space matrices

CALLING SEQUENCE:

[A,B,C,D]=abcd(sl)

PARAMETERS:

sl : linear system (syslin list) in state-space or transfer form
A,B,C,D : real matrices of appropriate dimensions

DESCRIPTION:
returns the A, B, C, D matrices from a linear system Sl.
Utility function. For transfer matrices Sl it is converted into state-space form by tf2ss.
The matrices A, B, C, D are the elements 2 to 5 of the syslin list Sl, i.e. [A,B,C,D] = Sl(2:5).

EXAMPLE:

A=diag([1,2,3]);B=[1;1;1];C=[2,2,2];
sys=syslin('c',A,B,C);
sys("A")
sys("C")
[A1,B1,C1,D1]=abcd(sys);
A1
systf=ss2tf(sys);
[a,b,c,d]=abcd(systf)
spec(a)
c*b-C*B
c*a*b-C*A*B

SEE ALSO: syslin 224, ssrand 220

2.1.2 abinv ------------------------------- AB invariant subspace

CALLING SEQUENCE:

[X,dims,F,U,k,Z]=abinv(Sys,alfa,beta,flag)

PARAMETERS:

Sys : syslin list containing the matrices [A,B,C,D].
alfa : (optional) real number or vector (possibly complex, location of closed loop poles)
beta : (optional) real number or vector (possibly complex, location of closed loop poles)
flag : (optional) character string 'ge' (default) or 'st' or 'pp'
X : orthogonal matrix of size nx (dim of state space).
dims : integer row vector dims=[dimR,dimVg,dimV,noc,nos] with dimR<=dimVg<=dimV<=noc<=nos.
    If flag='st', (resp. 'pp'), dims has 4 (resp. 3) components.
F : real matrix (state feedback)
k : integer (normal rank of Sys)
Z : non-singular linear system (syslin list)
DESCRIPTION:
Output nulling subspace (maximal unobservable subspace) for $Sys = \text{linear system}$ defined by a syslin list containing the matrices $[A,B,C,D]$ of $Sys$. The vector $\text{dims}=[\text{dimR}, \text{dimVg}, \text{dimV}, \text{noc}, \text{nos}]$ gives the dimensions of subspaces defined as columns of $X$ according to partition given below. The $\text{dimV}$ first columns of $X$ i.e $V=X(:,1:1:dimV)$, span the $A\text{B}$-invariant subspace of $Sys$ i.e the unobservable subspace of $(A+B*F,C+D*F)$. $(\text{dimV}=\text{nx}$ iff $C^\dagger(D)=X)$. The $\text{dimR}$ first columns of $X$ i.e $R=X(:,1:1:dimR)$ spans the controllable part of $Sys$ in $V$. $(\text{dimR}=\text{dimV})$. $(\text{dimR}=0$ for a left invertible system). $R$ is the maximal controllability subspace of $Sys$ in kernel($C$).

The $\text{dimVg}$ first columns of $X$ i.e $Vg=X(:,1:1:dimVg)$, span the $A\text{B}$-stabilizable subspace of $Sys$. $(\text{dimVg}=\text{dimV})$. $F$ is a decoupling feedback: for $X=[V,X2]$ $(X2=X(:,dimV+1:nx))$ one has $X2'(A+B*F)*V=0$ and $(C+D*F)*V=0$. The zeros od $Sys$ are given by: $X0=X(:,dimR+1:dimV)$; $\text{spec}(X0'(A+B*F)*X0)$ i.e there are $\text{dimV}-\text{dimR}$ closed-loop fixed modes.

If the optional parameter $\text{alfa}$ is given as input, the $\text{dimR}$ controllable modes of $(A+BF)$ in $V$ are set to $\text{alfa}$ (or to $[\text{alfa}(1), \text{alfa}(2), ...]$). $\text{alfa}$ can be a vector (real or complex pairs) or a (real) number. Default value $\text{alfa}=-1$.

If the optional parameter $\text{beta}$ is given as input, the $\text{noc}$-dim$\text{V}$ controllable modes of $(A+BF)$ "outside" $V$ are set to $\text{beta}$ (or to $[\text{beta}(1), \text{beta}(2), ...]$). Default value $\text{beta}=-1$.

In the $X, U$ bases, the matrices $[X'*(A+B*F)*X,X'*B*U;(C+D*F)*X,D*U]$ are displayed as follows:

\[
\begin{bmatrix}
\text{A11},*,*,*,*,* & \text{B11} & * \\
0,\text{A22},*,*,*,* & 0 & * \\
0,0,\text{A33},*,*,* & 0 & * \\
0,0,0,\text{A44},*,* & 0 & \text{B42} \\
0,0,0,0,\text{A55},* & 0 & 0 \\
0,0,0,0,0,\text{A66} & 0 & 0 \\
0,0,*,*,*,* & 0 & \text{D2}
\end{bmatrix}
\]

where the $X$-partitioning is defined by $\text{dims}$ and the $U$-partitioning is defined by $k$. $A11$ is $(\text{dimR} \times \text{dimR})$ and has its eigenvalues set to $\text{alfa}(i)$'s. The pair $(\text{A11}, \text{B11})$ is controllable and $\text{B11}$ has $\nu-k$ columns. $A22$ is a stable $(\text{dimVg}-\text{dimR} \times \text{dimVg}-\text{dimR})$ matrix. $A33$ is an unstable $(\text{dimVg}-\text{dimR} \times \text{dimVg}-\text{dimR})$ matrix (see $\text{stability}$). $A44$ is $(\text{noc}-\text{dimV} \times \text{noc}-\text{dimV})$ and has its eigenvalues set to $\text{beta}(i)$'s. The pair $(\text{A44}, \text{B42})$ is controllable. $A55$ is a stable $(\text{nos}-\text{noc} \times \text{nos}-\text{noc})$ matrix. $A66$ is an unstable $(\text{nx}-\text{nos} \times \text{nx}-\text{nos})$ matrix (see $\text{stability}$). $Z$ is a column compression of $Sys$ and $k$ is the normal rank of $Sys$ i.e $\text{Sys} * Z$ is a column-compressed linear system. $k$ is the column dimensions of $\text{B42,B52,B62}$ and $D2$. $[\text{B42;B52;B62;D2}]$ is full column rank and has rank $k$.

If $\text{flag}='st'$ is given, a five blocks partition of the matrices is returned and $\text{dims}$ has four components. If $\text{flag}='pp'$ is given a four blocks partition is returned. In case $\text{flag}='ge'$ one has $\text{dims}=[\text{dimR}, \text{dimVg}, \text{dimV}, \text{noc}, \text{nos}]$ where $\text{nc}$ is the dimension of the controllable pair $(\text{A44}, \text{B42})$ (resp. $(\text{A44},*,0,\text{A55})$, $[\text{B42};0]$). In case $\text{flag}='st'$ one has $\text{dims}=[\text{dimR}, \text{dimVg}, \text{dimVg}+\text{nc}, \text{dimVg}+\text{ns}]$ and in case $\text{flag}='pp'$ one has $\text{dims}=[\text{dimR}, \text{dimR}+\text{nc}, \text{dimR}+\text{ns}]$. $\text{nc}$ (resp. $\text{ns}$) is here the dimension of the controllable (resp. stabilizable) subspace of the blocks 3 to 6 (resp. 2 to 6). This function can be used for the (exact) disturbance decoupling problem.

DDPS:

Find $u=Fx+Rd=[F,R]*[x;d]$ which rejects $Q*d$ and stabilizes the plant:

\[
\begin{align*}
xdot &= Ax+Bu+Qd \\
y &= Cx+Du+Td
\end{align*}
\]

DDPS has a solution if $\text{Im}(Q)$ is included in $Vg + \text{Im}(B)$ and stabilizability.
assumption is satisfied.
Let G=(X(:,dimVg+1:$))' = left annihilator of Vg i.e. G*Vg=0;
B2=G*B; Q2=G*Q; DDPS solvable iff [B2;D]*R + [Q2;T] =0 has a solution.
The pair F,R is the solution (with F=output of abinv).
Im(Q2) is in Im(B2) means row-compression of B2=>row-compression of Q2
Then C*[sI-A-B*F]^(1) +D]*(Q+B*R) =0 (<=>G*(Q+B*R)=0)

EXAMPLE :

nu=3;ny=4;nx=7;
nrt=2;ngt=3;ng0=3;nvt=5;rk=2;
flag=list('on',nrt,ngt,ng0,nvt,rk);
Sys=srand(ny,nu,nx,flag);alfa=-1;beta=-2;
[X,dims,F,U,k,Z]=abinv(Sys,alfa,beta);
[A,B,C,D]=abcd(Sys);dimV=dims(3);dimR=dims(1);
V=X(:,1:dimV);X2=X(:,dimV+1:nx);
X2'*(A+B*F)*V
(C+D*F)*V
X0=X(:,dimR+1:dimV); spec(X0'*(A+B*F)*X0)
trzeros(Sys)
spec(A+B*F) //nr=2 evals at -1 and noc-dimV=2 evals at -2.
clean(ss2tf(Sys*Z))

// 2nd Example
nx=6;ny=3;nu=2;
A=diag(1:6);A(2,2)=-7;A(5,5)=-9;B=[1,2;0,3;0,4;0,5;0,0,0];
C=[zeros(ny,ny),eye(ny,ny)];D=[0,1;0,2;0,3];
sl=syslin('c',A,B,C,D);//sl=ss2ss(sl,rand(6,6))*rand(2,2);
[A,B,C,D]=abcd(sl); //The matrices of sl.
alfa=-1;beta=-2;
[X,dims,F,U,k,Z]=abinv(sl,alfa,beta);dimVg=dims(2);
clean(X'*(A+B*F)*X)
clean(X'*B*U)
clean((C+D*F)*X)
clean(D*U)
G=(X(:,dimVg+1:$))';
B2=G*B;nd=3;
R=rand(nu,nd);Q2T=-[B2;D]*R;
p=size(G,1);Q2=Q2T(1:p,:);T=Q2T(p+1:$,:);
Q=G\Q2; //a valid [Q;T] since
[B2;D]*R + [G*Q;T] // is zero
closed=syslin('c',A+B*F,Q+B*R,C+D*F,T+D*R); // closed loop: d-->y
ss2tf(closed) // Closed loop is zero
spec(closed('A')) //The plant is not stabilizable!
[ns,nc,W,sl1]=st_ility(sl);
slnew=syslin('c',A,B,C,D); //New stabilizable
//Fnew=stabil(slnew('A'),slnew('B'),-11);
//slnew('A')=slnew('A')+slnew('B')*Fnew;
//slnew('C')=slnew('C') + slnew('D')*Fnew;
[X,dims,F,U,k,Z]=abinv(slnew,alfa,beta);dimVg=dims(2);
[A,B,C,D]=abcd(slnew);
G=(X(:,dimVg+1:$))';
B2=G*B;nd=3;
R=rand(nu,nd);Q2T=-[B2;D]*R;
p=size(G,1);Q2=Q2T(1:p,:);T=Q2T(p+1:$,:);
Q=G\Q2; //a valid [Q;T] since
ARL2 \textbf{Scilab Function}

\[ [G*B;D] + [G*Q;T] \text{ is zero} \]
\[ \text{closed} = \text{syslin}(\text{c}',A+B*F,Q+B*R,C+D*F,T+D*R); \text{ // closed loop: } d \rightarrow y \]
\[ \text{ss2tf(closed)} \text{ // Closed loop is zero} \]
\[ \text{spec(closed('A'))} \]

\textit{AUTHOR : F.D.}

\textit{SEE ALSO:} \texttt{cainv 323}, \texttt{st_ility 364}, \texttt{sstrand 220}, \texttt{ss2ss 362}, \texttt{ddp 331}

\textbf{2.1.3 \texttt{arhnk}} \hspace{1cm} \textbf{Hankel norm approximant}

\textbf{CALLING SEQUENCE :}

\[ [\text{slm}] = \text{arhnk}(s\text{l},o\text{rd},[\text{tol}]) \]

\textbf{PARAMETERS :}

\texttt{sl} : linear system (\texttt{syslin} list)
\texttt{ord} : integer, order of the approximant
\texttt{tol} : threshold for rank determination in \texttt{equil1}

\textbf{DESCRIPTION :}

computes \texttt{slm}, the optimal Hankel norm approximant of the stable continuous-time linear system \texttt{sl} with matrices \([A,B,C,D]\).

\textbf{EXAMPLE :}

\[ A = \text{diag}([-1,-2,-3,-4,-5]); B = \text{rand}(5,1); C = \text{rand}(1,5); \]
\[ s\text{l} = \text{syslin}(\text{c}',A,B,C); \]
\[ s\text{lapprox} = \text{arhnk}(s\text{l},2); \]
\[ [n\text{k},W] = \text{hankelsv}(s\text{l}); n\text{k} \]
\[ [n\text{kred},W\text{red}] = \text{hankelsv}(s\text{lapprox}); n\text{kred} \]

\textit{SEE ALSO:} \texttt{equil 334}, \texttt{equill 335}, \texttt{hankelsv 383}

\textbf{2.1.4 \texttt{arl2}} \hspace{1cm} \textbf{SISO model realization by L2 transfer approximation}

\textbf{CALLING SEQUENCE :}

\[ h = \text{arl2}(y,\text{den}0,\text{n }[,\text{imp}]) \]
\[ h = \text{arl2}(y,\text{den}0,\text{n }[,\text{imp}],’all’) \]
\[ [\text{den num} , \text{err}] = \text{arl2}(y,\text{den}0,\text{n }[,\text{imp}]) \]
\[ [\text{den num} , \text{err}] = \text{arl2}(y,\text{den}0,\text{n }[,\text{imp}],’all’) \]

\textbf{PARAMETERS :}

\texttt{y} : real vector or polynomial in \( z^{-1} \), it contains the coefficients of the Fourier's series of the rational system to approximate (the impulse response)
\texttt{den0} : a polynomial which gives an initial guess of the solution, it may be \texttt{poly(1,’z’,’c’)}
\texttt{n} : integer, the degree of approximating transfer function (degree of \texttt{den})
\texttt{imp} : integer in \( \{0,1,2\} \) (verbose mode)
\texttt{h} : transfer function \texttt{num/den} or transfer matrix (column vector) when flag ‘all’ is given.
\texttt{den} : polynomial or vector of polynomials, contains the denominator(s) of the solution(s)
\texttt{num} : polynomial or vector of polynomials, contains the numerator(s) of the solution(s)
\texttt{err} : real constant or vector, the L2-error achieved for each solutions
DESCRIPTION:
[den, num, err] = arl2(y, den0, n [, imp]) finds a pair of polynomials num and den such that the transfer function num/den is stable and its impulse response approximates (with a minimal L2 norm) the vector y assumed to be completed by an infinite number of zeros.

If:
\[ y(z) = y(1)\left(\frac{1}{z}\right)^1 + y(2)\left(\frac{1}{z}\right)^2 + \ldots + y(ny)\left(\frac{1}{z}\right)^ny \]
then L2-norm of num/den - y(z) is err.

n is the degree of the polynomial den.

The num/den transfer function is a L2 approximant of the Fourier’s series of the rational system.

Various intermediate results are printed according to imp.

[den, num, err] = arl2(y, den0, n [,imp], 'all') returns in the vectors of polynomials num and den a set of local optimums for the problem. The solutions are sorted with increasing errors err. In this case den0 is already assumed to be poly(1,'z','c').

EXAMPLE:

```scilab
v = ones(1,20);
xbash();
plot2d1('enn',0,[v';zeros(80,1)],2,'051',' ',[1,-0.5,100,1.5])

[d,n,e]=arl2(v,poly(1,'z','c'),1)
plot2d1('enn',0,ldiv(n,d,100),2,'000')
[d,n,e]=arl2(v,d,3)
plot2d1('enn',0,ldiv(n,d,100),3,'000')
[d,n,e]=arl2(v,d,8)
plot2d1('enn',0,ldiv(n,d,100),5,'000')
```

SEE ALSO: ldiv 493, imrep2ss 341, time_id 367, armax 395, frep2tf 338

### 2.1.5 balreal -balanced realization-

CALLING SEQUENCE:

```
[slb [,U] ] = balreal(sl)
```

PARAMETERS:

sl, slb : linear systems (syslin lists)

DESCRIPTION:

Balanced realization of linear system sl=[A, B, C, D]. sl can be a continuous-time or discrete-time state-space system. sl is assumed stable.

slb=[inv(U)*A*U , inv(U)*B , C*U , D]

is the balanced realization.

slb is returned as a syslin list.

EXAMPLE:

```
A=diag([-1,-2,-3,-4,-5]); B=rand(5,2); C=rand(1,5);
sl=syslin('c',A,B,C);
[slb,U]=balreal(sl);
Wc=clean(ctr_gram(slb))
W0=clean(obs_gram(slb))
```

SEE ALSO: ctr_gram 330, obs_gram 349, hankelsv 383, equil 334, equill 335
2.1.6 **bilin**

**general bilinear transform**

**CALLING SEQUENCE:**

\[ [s11]=\text{bilin}(s1,v) \]

**PARAMETERS:**

- \( s1, s11 \): linear systems (syslin lists)
- \( v \): real vector with 4 entries (\( v=[a,b,c,d] \))

**DESCRIPTION:**

Given a linear system in state space form, \( s1=\text{syslin}(\text{dom}, A, B, C, D) \) (syslin list), \( s1=\text{bilin}(s1,v) \) returns in \( s11 \) a linear system with matrices [\( A1, B1, C1, D1 \)] such that the transfer function \( H1(s)=C1*\text{inv}(s*\text{eye}()-A1) \) is obtained from \( H(z)=C*\text{inv}(z*\text{eye}()-A)*B+D \) by replacing \( z=(a*s+b)/(c*s+d) \). One has \( w=\text{bilin}(\text{bilin}(w,[a,b,c,d]),[d,-b,-c,a]) \)

**EXAMPLE:**

\[
\begin{align*}
s &= \text{poly}(0,'s'); z = \text{poly}(0,'z'); \\
w &= \text{ssrand}(1,1,3); \\
wtf &= \text{ss2tf}(w); v = [2,3,-1,4]; a = v(1); b = v(2); c = v(3); d = v(4); \\
&[\text{horner}(wtf,(a*z+b)/(c*z+d)),\text{ss2tf(\text{bilin}(w,[a,b,c,d]))}] \\
&\text{clean}([\text{ss2tf(\text{bilin(\text{bilin}(w,[a,b,c,d]),[d,-b,-c,a]))})-wtf])
\end{align*}
\]

**SEE ALSO:** horner 490, cls2dls 326

2.1.7 **cainv**

**Dual of abinv**

**CALLING SEQUENCE:**

\[ [X, \text{dims}, J, Y, k, Z] = \text{cainv}(S1, alfa, beta, flag) \]

**PARAMETERS:**

- \( S1 \): syslin list containing the matrices [\( A, B, C, D \)].
- \( alfa \): real number or vector (possibly complex, location of closed loop poles)
- \( beta \): real number or vector (possibly complex, location of closed loop poles)
- \( flag \): (optional) character string ‘ge’ (default) or ‘st’ or ‘pp’
- \( X \): orthogonal matrix of size nx (dim of state space).
- \( \text{dims} \): integer row vector \( \text{dims}=[n\text{d1}, n\text{u1}, \text{dimS}, \text{dimSg}, \text{dimN}] \) (5 entries, nondecreasing order). If \( \text{flag}='st' \), (resp. ‘pp’), \( \text{dims} \) has 4 (resp. 3) components.
- \( J \): real matrix (output injection)
- \( Y \): orthogonal matrix of size ny (dim of output space).
- \( k \): integer (normal rank of \( S1 \))
- \( Z \): non-singular linear system (syslin list)

**DESCRIPTION:**

\( \text{cainv} \) finds a bases \( (X, Y) \) (of state space and output space resp.) and output injection matrix \( J \) such that the matrices of \( S1 \) in bases \( (X, Y) \) are displayed as:

\[
\begin{align*}
[A11, *, *, *, *, *] & \quad [*] \\
[0, A22, *, *, *, *] & \quad [*] \\
X'*(A+J*C)*X &= [0, 0, A33, *, *, *] \\
X'*(B+J*D) &= [*] \\
[0, 0, 0, A44, *, *] & \quad [0]
\end{align*}
\]

Scilab Group April 1993 323
The partition of $X$ is defined by the vector $\text{dims}=\text{[nd1, nu1, dimS, dimSg, dimN]}$ and the partition of $Y$ is determined by $k$.

Eigenvalues of $A_{11}$ ($nd1 \times nd1$) are unstable. Eigenvalues of $A_{22}$ ($nu1-nd1 \times nu1-nd1$) are stable.

The pair $(A_{33}, C_{13})$ ($dimS-nu1 \times dimS-nu1$, $k \times dimS-nu1$) is observable, and eigenvalues of $A_{33}$ are set to $\alpha$.

Matrix $A_{44}$ ($dimSg-dimS \times dimSg-dimS$) is unstable. Matrix $A_{55}$ ($dimN-dimSg, dimN-dimSg$) is stable.

The pair $(A_{66}, C_{26})$ ($nx-dimN \times nx-dimN$) is observable, and eigenvalues of $A_{66}$ set to $\beta$.

If $\text{flag}='st'$ is given, a five blocks partition of the matrices is returned and $\text{dims}$ has four components.

If $\text{flag}='pp'$ is given a four blocks partition is returned (see $\text{abinv}$).

This function can be used to calculate an unknown input observer:

```plaintext
// DDEP: dot(x)=A x + Bu + Gd
//       y= Cx     (observation)
//       z= Hx     (z=variable to be estimated, d=disturbance)
// Find: dot(w) = Fw + Ey + Ru such that
//       zhat = Mw + Ny
//       z-Hx goes to zero at infinity
// Solution exists iff Ker H contains Sg(A,C,G) inter KerC (assuming detectability)
// i.e. H is such that:
// For any $W$ which makes a column compression of $[Xp(1:dimSg,:);C]
// with $Xp=X'$ and $[X, dims, J, Y, k, Z]=\text{cainv(syslin('c',A,G,C));}
// $[Xp(1:dimSg,:);C]*W = [0 | \ast]$ one has
// $H*W = [0 \mid \ast]$ (with at least as many aero columns as above).
```

See also: $\text{abinv 318}$, $\text{dt_ility 334}$, $\text{ui_observer 369}$

### 2.1.8 calfrq ................................. frequency response discretization

**CALLING SEQUENCE:**

```
[frq, split]=\text{calfrq(h, [fmin, fmax])}
```

**PARAMETERS:**

- $h$ : SISO linear system (syslin list)
- $fmin, fmax$ : real scalars (min and max frequencies)
- $frq$ : row vector (discretization of interval)
- $\text{split}$ : vector of $frq$ splitting points indexes

Scilab Group  
April 1993  
324
DESCRIPTION:
frequency response discretization; \texttt{frq} is the discretization of \([f_{\text{min}}, f_{\text{max}}]\) such that the peaks in the frequency response are well represented.
Default values for \(f_{\text{min}}\) and \(f_{\text{max}}\) are \(1.d-3, 1.d+3\) if \(h\) is continuous-time or \(1.d-3, 1/(2*h(’dt’))\) if \(h\) is discrete-time.
Singularities are located between \(\texttt{frq(split(k))}\) and \(\texttt{frq(split(k)+1)}\) for \(k>1\).

EXAMPLE:

\[
\begin{align*}
s &= \text{poly}(0,’s’) \\
h &= \text{syslin(’c’, (s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01))] \\
h1 &= h*\text{syslin(’c’, (s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225))} \\
[f1, spl] &= \text{calfrq(h1, 0.01, 1000)}; \\
rf &= \text{repfreq(h1, f1)}; \\
\text{plot2d}\left(\text{real(rf)’, imag(rf)’}\right)
\end{align*}
\]

SEE ALSO: \texttt{bode 86, black 85, nyquist 115, freq 339, repfreq 355, logspace 197}

2.1.9 \texttt{canon} \hspace{1cm} \texttt{canonical controllable form}

CALLING SEQUENCE:

\[
[\text{Ac, Bc, U, ind}] = \text{canon}(A, B)
\]

PARAMETERS:

\(\text{Ac, Bc}\) : canonical form
\(\text{U}\) : current basis (square nonsingular matrix)
\(\text{ind}\) : vector of integers, controllability indices

DESCRIPTION:
gives the canonical controllable form of the pair \((A, B)\).
\(A_c = \text{inv}(U)*A*U, B_c = \text{inv}(U)*B\)
The vector \(\text{ind}\) is made of the \(\epsilon_i\)’s indices of the pencil \([s I - A, B]\) (decreasing order).
For example with \(\text{ind} = [3, 2]\), \(A_c\) and \(B_c\) are as follows:

\[
A_c = \begin{bmatrix} * & * & * & * & * \\ 1,0,0,0,0 \\ 0,1,0,0,0 \\ 6,7,8,9,0 \\ 0,0,0,1,0 \end{bmatrix} \quad B_c = [0] \\
\begin{bmatrix} * & * & * & * \end{bmatrix} \quad \begin{bmatrix} * \\ 0,0,1,0 \end{bmatrix} \quad \begin{bmatrix} 0 \end{bmatrix}
\]

If \((A, B)\) is controllable, by an appropriate choice of \(F\) the \(*\) entries of \(A_c B_c F\) can be arbitrarily set to desired values (pole placement).

EXAMPLE:

\[
\begin{align*}
A &= \begin{bmatrix} 1,2,3,4,5 \\ 1,0,0,0,0 \\ 0,1,0,0,0 \\ 6,7,8,9,0 \\ 0,0,0,1,0 \end{bmatrix}; \\
B &= \begin{bmatrix} 1,2 \\ 0,0 \\ 0,0 \\ 2,1 \end{bmatrix};
\end{align*}
\]

Scilab Group April 1993 325
0,0];
X=rand(5,5);A=X*A*inv(X);B=X*B; //Controllable pair
[Ac,Bc,U,ind]=canon(A,B); //Two indices --> ind=[3.2];
index=1;
for k=1:size(ind,'*')-1,index=[index,1+sum(ind(1:k))];end
Acstar=Ac(index,:);Bcstar=Bc(index,:);
s=poly(0,'s');
p1=s^3+2*s^2-5*s+3;p2=(s-5)*(s-3);
//p1 and p2 are desired closed-loop polynomials with degrees 3,2
c1=coeff(p1);c1=c1(s-1:-1:1);c2=coeff(p2);c2=c2(s-1:-1:1);
Acstardesired=[-c1,0,0,0,0,-c2];
//Acstardesired(index,:) is companion matrix with char. pol=p1*p2
F=Bcstar\(Acstardesired-Acstar); //Feedbak gain
Ac+Bc*F // Companion form
spec(A+B*F/U) // F/U is the gain matrix in original basis.

See Also: obsv_mat 351, cont_mat 327, ctrgram 330, contrss 328, ppol 354, contr 328, stabil 364

Author: F.D.

2.1.10 cls2dls ____________________________ bilinear transform

Calling Sequence:

[sll]=cls2dls(sl,T [,fp])

Parameters:

sl,sll : linear systems (syslin lists)
T : real number, the sampling period
fp : prevarping frequency in hertz

Description:

given sl=[A,B,C,D] (syslin list),a continuous time system cls2dls returns the sampled system obtained by the bilinear transform \( s=(2/T) * (z-1)/(z+1) \).

Example:

s=poly(0,'s');z=poly(0,'z');
sl=syslin('c',c'(s+1)/(s^2-5*s+2)); //Continuous-time system in transfer form
slss=tf2ss(sl); //Now in state-space form
sll=cls2dls(slss,0.2); //sll= output of cls2dls
sllt=ss2tf(sll) // Converts in transfer form
sll2=horner(sl,(2/0.2)*(z-1)/(z+1)) //Compare sll2 and sll

See Also: horner 490

2.1.11 colregul __________________________ removing poles and zeros at infinity

Calling Sequence:

[Stmp,Ws]=colregul(Sl,alfa,beta)

Parameters:

Sl,Stmp : syslin lists

See Also: obsv_mat 351, cont_mat 327, ctrgram 330, contrss 328, ppol 354, contr 328, stabil 364

Author: F.D.

Scilab Group April 1993 326
CONTR Scilab Function

alfa, beta : reals (new pole and zero positions)

DESCRIPTION:
computes a prefilter \( W_s \) such that \( S_{tmp} = S_l * W_s \) is proper and with full rank \( D \) matrix.
Poles at infinity of \( S_l \) are moved to \( \alpha \);
Zeros at infinity of \( S_l \) are moved to \( \beta \);
\( S_l \) is a assumed to be a left invertible linear system (syslin list) in state-space representation with
possibly a polynomial \( D \) matrix.

SEE ALSO: invsyslin 342, inv 516, rowregul 357, rowshuff 532

AUTHOR: F. D., R. N.

2.1.12 cont_frm __________________________ transfer to controllable state-space

CALLING SEQUENCE:

\[ [sl] = \text{cont}_\text{frm}(\text{NUM},\text{den}) \]

PARAMETERS:

NUM : polynomial matrix
den : polynomial
sl : syslin list, \( sl=[A,B,C,D] \).

DESCRIPTION:
controllable state-space form of the transfer \( \text{NUM}/\text{den} \).

EXAMPLE:

\[
\begin{align*}
s &= \text{poly}(0, \text{’s’}); \text{NUM} = [1+s, s]; \text{den} = s^2 - 5s + 1; \\
sl &= \text{cont}_\text{frm}(\text{NUM},\text{den}); \\
slss &= \text{ss2tf}(sl); & \text{//Compare with \( \text{NUM}/\text{den} \)}
\end{align*}
\]

SEE ALSO: tf2ss 367, canon 325, contr 328

2.1.13 cont_mat ______________________________ controllability matrix

CALLING SEQUENCE:

\[ \text{Cc} = \text{cont}_\text{mat}(A,B) \]
\[ \text{Cc} = \text{cont}_\text{mat}(sl) \]

PARAMETERS:

\( a, b \) : two real matrices of appropriate dimensions
\( sl \) : linear system (syslin list)

DESCRIPTION:
\( \text{cont}_\text{mat} \) returns the controllability matrix of the pair \( A, B \) (resp. of the system \( sl=[A,B,C,D] \)).
\[ \text{Cc} = [B, AB, A^2 B, \ldots, A^{(n-1)} B] \]

SEE ALSO: ctr_gram 330, contr 328, canon 325, st_ility 364

Scilab Group April 1993 327
2.1.14 contr  

CALLING SEQUENCE:

\[ [n, [U]] = \text{contr}(A, B [, tol]) \]
\[ [A1, B1, U, ind] = \text{contr}(A, B [, tol]) \]

PARAMETERS:

- **A, B**: real matrices
- **tol**: may be the constant rtol or the 2 vector [rtol atol]
- **rtol**: tolerance used when evaluating ranks (QR factorizations).
- **atol**: absolute tolerance (the **B** matrix is assumed to be 0 if \( \text{norm}(B) < \text{atol} \))
- **n**: dimension of controllable subspace.
- **U**: orthogonal change of basis which puts \((A, B)\) in canonical form.
- **A1**: block Hessenberg matrix
- **B1**: is \( U' \times B \)
- **ind**: vector associated with controllability indices (dimensions of subspaces \( B, B + A \times B, \ldots = \text{ind}(1), \text{ind}(1) + \text{ind}(2), \ldots \))

DESCRIPTION:

\[ [n, [U]] = \text{contr}(A, B, [tol]) \] gives the controllable form of an \((A, B)\) pair (\( dx/dt = A \times x + B \times u \)) or \( x(n+1) = A \times x(n) + b \times u(n) \). The \( n \) first columns of \( U \) make a basis for the controllable subspace. If \( V = U(:,1:n) \), then \( V' \times A \times V \) and \( V' \times B \) give the controllable part of the \((A, B)\) pair. \[ [A1, B1, U, ind] = \text{contr}(A, B) \] returns the Hessenberg controllable form of \((A, B)\).

EXAMPLE:

\( W = \text{ssrand}(2, 3, 5, \text{list}('co', 3)); \) // cont. subspace has dim 3.
\( A = W('A'); B = W('B'); \)
\( [n, U] = \text{contr}(A, B); n \)
\( A1 = U' \times A \times U; \)
\( \text{spec}(A1(n+1:$$, n+1:$$)) \) // uncontrollable modes
\( \text{spec}(A + B \times \text{rand}(3, 5)) \)

SEE ALSO: **canon** 325, **cont_mat** 327, **unobs** 370, **stabil** 364

2.1.15 contrss  

CALLING SEQUENCE:

\[ [slc] = \text{contrss}(sl [, tol]) \]

PARAMETERS:

- **sl**: linear system (syslin list)
- **tol**: is a threshold for controllability (see contr). default value is \( \text{sqrt}(\%\text{eps}) \).

DESCRIPTION:

returns the controllable part of the linear system \( sl = (A, B, C, D) \) in state-space form.

EXAMPLE:

\( A = \begin{bmatrix} 1 & 1; 0 & 2 \end{bmatrix}; B = \begin{bmatrix} 1; 0 \end{bmatrix}; C = \begin{bmatrix} 1, 1 \end{bmatrix}; sl = \text{syslin}('c', A, B, C); \) // Non minimal
\( slc = \text{contrss}(sl); \)
\( sl1 = \text{ss2tf}(sl); sl2 = \text{ss2tf}(slc); \) // Compare sl1 and sl2

SEE ALSO: **cont_mat** 327, **ctr_gram** 330, **cont_frm** 327, **contr** 328
2.1.16  csim ____________________  simulation (time response) of linear system

CALLING SEQUENCE:

[y [,x]]=csim(u,t,sl,[x0 [,tol]])

PARAMETERS:

u  : function, list or string (control)
  t  : real vector specifying times with, t(1) is the initial time (x0=x(t(1))).
  sl : list (syslin)
  y  : a matrix such that y=[y(t(i)], i=1,...,n
  x  : a matrix such that x=[x(t(i)], i=1,...,n
  tol : a 2 vector [atol rtol] defining absolute and relative tolerances for ode solver (see ode)

DESCRIPTION:

simulation of the controlled linear system sl. sl is assumed to be a continuous-time system represented
by a syslin list.

u  is the control and x0 the initial state.

y  is the output and x the state.

The control can be:

1. a function: [inputs]=u(t)
2. a list: list(ut,parameter1,...,parametern) such that: inputs=ut(t,parameter1,...,parametern
(ut is a function)
3. the string "impuls" for impulse response calculation (here sl is assumed SISO without direct feed through and x0=0)
4. the string "step" for step response calculation (here sl is assumed SISO without direct feed-through and x0=0)
5. a vector giving the values of u corresponding to each t value.

EXAMPLE:

s=poly(0,'s');rand('seed',0);w=ssrand(1,1,3);w('A')=w('A')-2*eye();
t=0:0.05:5;
//impulse(w) = step (s * w)
xbasc(0);xset("window",0);xselect();
plot2d([t',t'],[(csim('step',t,tf2ss(s)*w))',0*t'])
xbasc(1);xset("window",1);xselect();
plot2d([t',t'],[(csim('impulse',t,w))',0*t'])
//step(w) = impulse (s^{-1} * w)
xbasc(3);xset("window",3);xselect();
plot2d([t',t'],[(csim('step',t,w))',0*t'])
xbasc(4);xset("window",4);xselect();
plot2d([t',t'],[(csim('impulse',t,tf2ss(1/s)*w))',0*t'])

//input defined by a time function
deff('u=input(t)','u=abs(sin(t))')
xbasc();plot2d([t',t'],[(csim(input,t,w))',0*t'])

SEE ALSO: syslin 224, dsimul 333, flts 336, ltitr 347, rtitr 358, ode 431, impl 421

Scilab Group

Scilab Function

Scilab Group

April 1993
2.1.17 **ctr gram** controllability gramian

**CALLING SEQUENCE:**

\[
\begin{align*}
[Gc] &= \text{ctr\_gram}(A, B [, \text{dom}]) \\
[Gc] &= \text{ctr\_gram}(sl)
\end{align*}
\]

**PARAMETERS:**

- \(A, B\) : two real matrices of appropriate dimensions
- \(\text{dom}\) : character string (‘c’ (default value) or ‘d’)
- \(sl\) : linear system, \text{syslin} list

**DESCRIPTION:**

Controllability gramian of \((A, B)\) or \(sl\) (a \text{syslin} linear system).

\(\text{dom}\) character string giving the time domain: "d" for a discrete time system and "c" for continuous time (default case).

\[
G_c = \int_{0}^{\infty} e^{At} B B' e^{A't} \, dt \\
G_c = \sum_{k=0}^{\infty} A^k B B' A'^k
\]

**EXAMPLE:**

\[
\begin{align*}
A &= \text{diag}([-1, -2, -3]); B = \text{rand}(3, 2); \\
W_c &= \text{ctr\_gram}(A, B) \\
U &= \text{rand}(3, 3); A_1 = U A U; B_1 = U B; \\
W_{c1} &= \text{ctr\_gram}(A_1, B_1) \quad \text{//Not invariant!}
\end{align*}
\]

See Also: \text{equil1 335}, \text{obs\_gram 349}, \text{contr 328}, \text{cont\_mat 327}, \text{cont\_frm 327}, \text{contrss 328}

**AUTHOR:** S. Steer INRIA 1988

---

2.1.18 **dbphi** frequency response to phase and magnitude representation

**CALLING SEQUENCE:**

\[
[\text{db}, \text{phi}] = \text{dbphi}(\text{repf})
\]

**PARAMETERS:**

- \(\text{db}, \text{phi}\) : vector of gains (db) and phases (degrees)
- \(\text{repf}\) : vector of complex frequency response

**DESCRIPTION:**

\(\text{db}(k)\) is the magnitude of \(\text{repf}(k)\) expressed in dB, i.e. \(\text{db}(k) = 20 \times \log(\text{abs(\text{repf}(k)))}/\log(10)\)

and \(\phi(k)\) is the phase of \(\text{repf}(k)\) expressed in degrees.

See Also: \text{repfreq 355}, \text{bode 86}

---

Scilab Group April 1993 330
CALLING SEQUENCE:

\[[\text{Closed}, F, G] = \text{ddp}(\text{Sys}, \text{zeroed}, B1, D1)\]
\[[\text{Closed}, F, G] = \text{ddp}(\text{Sys}, \text{zeroed}, B1, D1, \text{flag}, \alpha, \beta)\]

PARAMETERS:

\text{Sys} : \text{syslin list containing the matrices } (A, B2, C, D2).
\text{zeroed} : \text{integer vector, indices of outputs of } \text{Sys} \text{ which are zeroed.}
B1 : \text{real matrix}
D1 : \text{real matrix. } B1 \text{ and } D1 \text{ have the same number of columns.}
\text{flag} : \text{string 'ge' or 'st' (default) or 'pp'.}
\alpha : \text{real or complex vector (loc. of closed loop poles)}
\beta : \text{real or complex vector (loc. of closed loop poles)}

DESCRIPTION:

Exact disturbance decoupling (output nulling algorithm). Given a linear system, and a subset of outputs, \( z \), which are to be zeroed, characterize the inputs \( w \) of \text{Sys} such that the transfer function from \( w \) to \( z \) is zero. \text{Sys} is a linear system \( \{A, B2, C, D2\} \) with one input and two outputs (i.e. \( \text{Sys: } u \rightarrow (z, y) \)), part the following system defined from \text{Sys} and \( B1, D1 \):

\[
\begin{align*}
\dot{x} &= Ax + B1w + B2u \\
z &= C1x + D11w + D12u \\
y &= C2x + D21w + D22u
\end{align*}
\]

outputs of \text{Sys} are partitioned into \((z, y)\) where \( z \) is to be zeroed, i.e. the matrices \( C \) and \( D2 \) are:

\[
\begin{align*}
C &= [C1; C2] \\
C1 &= C(\text{zeroed}, :) \\
D2 &= [D12; D22] \\
D12 &= D2(\text{zeroed}, :)
\end{align*}
\]

The matrix \( D1 \) is partitioned similarly as \( D1 = [D11; D21] \) with \( D11 = D1(\text{zeroed}, :) \). The control is \( u = Fx + Gw \) and one looks for matriced \( F, G \) such that the closed loop system: \( w \rightarrow z \) given by

\[
\begin{align*}
\dot{x} &= (A + B2*F)x + (B1 + B2*G)w \\
z &= (C1 + D12F)x + (D11 + D12*G)w
\end{align*}
\]

has zero transfer transfer function.

\text{flag='ge' : no stability constraints. flag='st' : look for stable closed loop system \((A+B2*F)\) stable. flag='pp' : eigenvalues of \( A+B2*F \) are assigned to \( \alpha \) and \( \beta \).

Closed is a realization of the \( \rightarrow y \) closed loop system

\[
\begin{align*}
\dot{x} &= (A + B2*F)x + (B1 + B2*G)w \\
y &= (C2 + D22*F)x + (D21 + D22*G)w
\end{align*}
\]

Stability (resp. pole placement) requires stabilizability (resp. controllability) of \((A,B2)\).

EXAMPLE:

\[
\begin{align*}
\text{rand('seed',0); nx=6; nz=3; nu=2; ny=1;}
A=\text{diag(1:6)}; A(2,2)=-7; A(5,5)=-9; B2=[1,2;0,3;0,4;0,5;0,0;0,0];
C1=[\text{zeros(nz,nz);}\text{eye(nz,nz)}]; D12=[0,1;0,2;0,3];
\text{Sys12= syslin('c',A,B2,C1,D12);}
C=[C1; \text{rand(ny,nx)}]; D2=[D12; \text{rand(ny,size(D12,2))}];
\text{Sys=syslin('c',A,B2,C,D2);}
[A,B2,C1,D12]=abcd(Sys12); //The matrices of Sys12.
\alpha=-1; \beta=-2; \text{flag='ge';}
[X,dims,F,U,k,Z]=abinv(Sys12,\alpha,\beta,flag);
\]

Scilab Group April 1993 331
```plaintext
clean(X'*(A+B2*F)*X)
clean(X'*B2*U)
clean((C1+D12*F)*X)
clean(D12*U);
//Calculating an ad-hoc B1,D1
G1=rand(size(B2,2),3);
B1=-B2*G1;
D11=-D12*G1;
D1=[D11;rand(ny,size(B1,2))];

[Closed,F,G]=ddp(Sys,1:nz,B1,D1,'st',alfa,beta);
ss2tf(closed)
```

See Also: abinv 318, ui_observer 369

### 2.1.20 des2tf

**descriptor to transfer function conversion**

**Calling Sequence:**

```
[S]=des2tf(sl)
[Bfs,Bis,chis]=des2tf(sl)
```

**Parameters:**

- `sl`: list (linear system in descriptor form)
- `Bfs`, `Bis`: two polynomial matrices
- `chis`: polynomial
- `S`: rational matrix

**Description:**

Given the linear system in descriptor form i.e. `Sl=list('des',A,B,C,D,E). des2tf` converts `sl` into its transfer function representation:

```
S=C*(s*E-A)^(-1)*B+D
```

called with 3 outputs arguments des2tf returns `Bfs` and `Bis` two polynomial matrices, and `chis` polynomial such that:

```
S=Bfs/chis - Bis
```

`chis` is the determinant of (`s*E-A`) (up to a xactive constant);

**Example:**

```python
s=poly(0,'s');
G=[1/(s+1),s;1+s^2,3*s^3];
Descrip=tf2des(G);Tf1=des2tf(Descrip)
Descrip2=tf2des(G,"withD");Tf2=des2tf(Descrip2)
[A,B,C,D,E]=Descrip2(2:6);Tf3=C*inv(s*E-A)*B+D
```

See Also: glever 512, pol2des 494, tf2des 390, ss2tf 363, des2ss 376, rowshuff 532

Author: F.D.

Scilab Group April 1993 332
2.1.21 **dscr** ____________________________ discretization of linear system

**CALLING SEQUENCE:**

\[
[sld [,r]]=\text{dscr}(sl,dt [,m])
\]

**PARAMETERS:**

- `sl`: `syslin` list containing \([A,B,C,D]\).
- `dt`: real number, sampling period
- `m`: covariance of the input noise (continuous time) (default value = 0)
- `r`: covariance of the output noise (discrete time) given if `m` is given as input
- `sld`: sampled (discrete-time) linear system, `syslin` list

**DESCRIPTION:**

Discretization of linear system. `sl` is a continuous-time system:
\[
\frac{dx}{dt}=A*x+B*u \quad (+\text{ noise})
\]

`sld` is the discrete-time system obtained by sampling `sl` with the sampling period `dt`.

**EXAMPLE:**

```plaintext
s=poly(0,'s');
Sys=syslin('c',[1,1/(s+1);2*s/(s^2+2),1/s])
ss2tf(dscr(tf2ss(Sys),0.1))
```

**SEE ALSO:** `syslin`, `flts`, `dsimul`

2.1.22 **dsimul** __________________________ state space discrete time simulation

**CALLING SEQUENCE:**

\[
y=\text{dsimul}(sl,u)
\]

**PARAMETERS:**

- `sl`: `syslin` list describing a discrete-time linear system
- `u`: real matrix of appropriate dimension
- `y`: output of `sl`

**DESCRIPTION:**

Utility function. If \([A,B,C,D]=abcd(sl)\) and \(x0=sl('X0')\), `dsimul` returns \(y=C*ltitr(A,B,u,x0)+D*u\) i.e. the time response of `sl` to the input `u`. `sl` is assumed to be in state space form (`syslin` list).

**EXAMPLE:**

```plaintext
z=poly(0,'z');
h=(1-2*z)/(z^2-0.2*z+1);
sl=tf2ss(h);
u=zeros(1,20);u(1)=1;
x1=dsimul(sl,u)  //Impulse response
u=ones(1,20);
x2=dsimul(sl,u);  //Step response
```

**SEE ALSO:** `syslin`, `flts`, `ltitr`

Scilab Group April 1993 333
2.1.23       dt_ility  

detectability test

CALLING SEQUENCE :

[k, [n [,U [,Sld ] ] ]]=dt_ility(Sl [,tol])

PARAMETERS :

Sl : linear system (syslin list)
n : dimension of unobservable subspace
k : dimension of unstable, unobservable subspace ( k<=n ).
U : orthogonal matrix
Sld : linear system (syslin list)
tol : threshold for controllability test.

DESCRIPTION :

Detectability test for sl, a linear system in state-space representation. U is a basis whose k first columns span the unstable, unobservable subspace of Sl (intersection of unobservable subspace of (A, C) and unstable subspace of A). Detectability means k=0.

Sld = (U'*A*U, U'*B, C*U, D) displays the "detectable part" of Sl=(A, B, C, D), i.e.

[*,*]                      [*,*]                      [*,*]
U'*A*U = [0,*]            [0,0,*]            [0,0,*]
[0,0,*]                    [0,0,*]                      [0,0,*]

C*U = [0,0,*]                with (A33,C3) observable (dimension nx-n), A22 stable (dimension n-k) and A11 unstable (dimension k).

EXAMPLE :

A=[2,1,1;0,-2,1;0,0,3];
C=[0,0,1];
X=rand(3,3);A=inv(X)*A*X;C=C*X;
W=syslin('c',A,[],C);
[k,n,U,W1]=dt_ility(W);
W1("A")
W1("C")

SEE ALSO: contr 328, st_ility 364, unobs 370, stabil 364

2.1.24       equil  

balancing of pair of symmetric matrices

CALLING SEQUENCE :

T=equil(P,Q)

PARAMETERS :

P, Q : two positive definite symmetric matrices
T : nonsingular matrix

DESCRIPTION :

equil returns t such that:
T*P*T' and inv(T)*Q*inv(T) are both equal to a same diagonal and positive matrix.

EXAMPLE :

Scilab Group  
April 1993
P=rand(4,4);P=P*P';
Q=rand(4,4);Q=Q*Q';
T=equil(P,Q)
clean(T*P*T')
clean(inv(T)'*Q*inv(T))

SEE ALSO: equil 335, balanc 502, ctr_gram 330

2.1.25 equil1 ________________ balancing (nonnegative) pair of matrices

CALLING SEQUENCE:

[T [,siz]]=equil1(P,Q [,tol])

PARAMETERS:

P, Q : two non-negative symmetric matrices
T : nonsingular matrix
siz : vector of three integers
tol : threshold

DESCRIPTION:
equil1 computes t such that:
P1=T*P*T' and Q1=inv(T)'*Q*inv(T) are as follows:
P1 = diag(S1,S2,0,0) and Q1 = diag(S1,0,S3,0) with S1,S2,S3 positive and diagonal matrices with respective dimensions siz=[n1,n2,n3]
tol is a threshold for rank determination in SVD

EXAMPLE:

S1=rand(2,2);S1=S1*S1';
S2=rand(2,2);S2=S2*S2';
S3=rand(2,2);S3=S3*S3';
P=sysdiag(S1,S2,zeros(4,4));
Q=sysdiag(S1,zeros(2,2),S3,zeros(2,2));
X=rand(8,8);
P=X*P*X';Q=inv(X)'*Q*inv(X);
[T,siz]=equil1(P,Q);
P1=clean(T*P*T')
Q1=clean(inv(T)'*Q*inv(T))

SEE ALSO: balreal 322, minreal 348, equil 334, hankelsv 383

AUTHOR: S. Steer 1987

2.1.26 feedback ____________________________ feedback operation

CALLING SEQUENCE:

S1=S11/.S12

PARAMETERS:

S11,S12 : linear systems (syslin list) in state-space or transfer form, or ordinary gain matrices.
S1 : linear system (syslin list) in state-space or transfer form

Scilab Group
April 1993
flts Scilab Function

DESCRIPTION:
The feedback operation is denoted by \(/\) (slashdot). This command returns \(S_1 = S_11 \cdot (I + S_12 \cdot S_11)^{-1}\), i.e. the (negative) feedback of \(S_11\) and \(S_12\). \(S_1\) is the transfer \(v \rightarrow y\) for \(y = S_11 \cdot u\), \(u = v - S_12 \cdot y\).

The result is the same as \(S_1 = \text{LFT}([0; I; I, -S_12], S_11)\).

Caution: do not use with decimal point (e.g. \(1/1.1\) is ambiguous!)

EXAMPLE:

\[
S_1 = \text{ssrand}(2, 2, 3); S_2 = \text{ssrand}(2, 2, 2);
W = S_1 / S_2;
\text{ss2tf}(S_1 / S_2)\]

// Same operation by LFT:
\[
\text{ss2tf}(\text{ltf}([\text{zeros}(2, 2), \text{eye}(2, 2); \text{eye}(2, 2), -S_2], S_1))\]

// Other approach: with constant feedback
\[
\text{BigS} = \text{sysdiag}(S_1, S_2); \quad F = [\text{zeros}(2, 2), \text{eye}(2, 2); -\text{eye}(2, 2), \text{zeros}(2, 2)];
\]
\[
\text{Bigclosed} = \text{BigS} / .F;
W_1 = \text{Bigclosed}(1:2, 1:2); \quad // W_1 = W \text{ (in state-space).}
\text{ss2tf}(W_1)\]

// Inverting
\[
\text{ss2tf}(S_1 \cdot \text{inv}(\text{eye()} + S_2 \cdot S_1))\]

SEE ALSO: \ lft 384, \ sysdiag 224, \ augment 373, \ obscont 349

2.1.27 flts _________________ time response (discrete time, sampled system)

CALLING SEQUENCE:

\[
[y [,x]] = \text{flts}(u, s1 [,x0])
\]
\[
[y] = \text{flts}(u, s1 [,past])
\]

PARAMETERS:

- \(u\): matrix (input vector)
- \(s1\): list (linear system \ text{syslin})
- \(x0\): vector (initial state; default value=0)
- \(past\): matrix (of the past; default value=0)
- \(x, y\): matrices (state and output)

DESCRIPTION:

State-space form:
\[
s1\text{ is a } \text{syslin } \text{list containing the matrices of the following linear system}
\]
\[
s1 = \text{syslin}('d', A, B, C, D) \text{ (see syslin):}
\]
\[
\begin{align*}
\begin{align*}
x[t+1] &= A \cdot x[t] + B \cdot u[t] \\
y[t] &= C \cdot x[t] + D \cdot u[t]
\end{align*}
\end{align*}
\]

or, more generally, if \(D\) is a polynomial matrix (\(p = \text{degree}(D(z))\)):
\[
D(z) = D_0 + z D_1 + z^2 D_2 + \ldots + z^p D_p
\]
\[
y[t] = C x[t] + D_0 u[t] + D_1 u[t+1] + \ldots + D_p u[t+p]
\]
\[
u = [u_0, u_1, \ldots u_n](input)
\]
\[
y = [y_0, y_1, \ldots y_{n-p}](output)
\]
flts Scilab Function

\[ x = x_{n-p+1} \]

(final state, used as x0 at next call to flts)

Transfer form:

\[ y = \text{flts}(u, sl[, \text{past}]) \]

Here sl is a linear system in transfer matrix representation i.e

\[ sl = \text{syslin}('d', \text{transfer matrix}) \quad (\text{see} \ \text{syslin}) \]

\[ past = \begin{bmatrix} u_{-nd} & \cdots & u_{-1} \\ y_{-nd} & \cdots & u_{-1} \end{bmatrix} \]

is the matrix of past values of u and y.

\( nd \) is the maximum of degrees of lcm’s of each row of the denominator matrix of sl.

\[ u = [u_0 \ u_1 \ \ldots \ u_n] \quad (\text{input}) \]
\[ y = [y_0 \ y_1 \ \ldots \ y_n] \quad (\text{output}) \]

\( p \) is the difference between maximum degree of numerator and maximum degree of denominator

**EXAMPLE:**

\[ sl = \text{syslin}('d', 1, 1, 1); u = 1:10; \]
\[ y = \text{flts}(u, sl); \]
\[ \text{plot2d2}("onn", (1:size(u,'c'))', y') \]
\[ [y1, x1] = \text{flts}(u(1:5), sl); \]
\[ y2 = \text{flts}(u(6:10), sl, x1); \]
\[ y - [y1, y2] \]

// With polynomial D:

\[ z = \text{poly}(0, 'z'); \]
\[ D = 1 + z + z^2; \quad p = \text{degree}(D); \]
\[ sl = \text{syslin}('d', 1, 1, 1, D); \]
\[ y = \text{flts}(u, sl); [y1, x1] = \text{flts}(u(1:5), sl); \]
\[ y2 = \text{flts}(u(5-p+1:10), sl, x1); \quad // (update) \]
\[ y - [y1, y2] \]

// Delay (transfer form): flts(u, 1/z)

// Usual responses

\[ z = \text{poly}(0, 'z'); \]
\[ h = (1-2*z)/(z^2+0.3*z+1) \]
\[ u = \text{zeros}(1, 20); u(1) = 1; \]
\[ \text{imprep} = \text{flts}(u, \text{tf2ss}(h)); \quad // \text{Impulse response} \]
\[ \text{plot2d2}("onn", (1:size(u,'c'))', \text{imprep}') \]
\[ u = \text{ones}(1, 20); \]
\[ \text{stprep} = \text{flts}(u, \text{tf2ss}(h)); \quad // \text{Step response} \]
\[ \text{plot2d2}("onn", (1:size(u,'c'))', \text{stprep}') \]

// Other examples

\[ A = [1 \ 2 \ 3; 0 \ 2 \ 4; 0 \ 0 \ 1]; B = [1 \ 0; 0 \ 0; 0 \ 1]; C = \text{eye}(3, 3); Sys = \text{syslin}('d', A, B, C); \]
\[ H = \text{ss2tf}(Sys); u = [1; -1]*(1:10); \]
\[ \text{yh} = \text{flts}(u, H); \]
\[ \text{ys} = \text{flts}(u, Sys); \]
\[ \text{norm}(\text{yh} - \text{ys}, 1) \]

// hot restart
\[ [ys1, x] = \text{flts}(u(:, 1:4), Sys); ys2 = \text{flts}(u(:, 5:10), Sys, x); \]
\[ \text{norm}([ys1, ys2] - \text{ys}, 1) \]

// yh1 = flts(u(:, 1:4), H); yh2 = flts(u(:, 5:10), H, [u(:, 2:4); yh(:, 2:4)]); \]
\[ \text{norm}([yh1, yh2] - yh, 1) \]

// with D<>0

Scilab Group April 1993 337
D=[-3 8 -0.5;4 -0.5;2.2 0.9];
Sys=syslin('d',A,B,C,D);
H=ss2tf(Sys); u=[1;-1]*10;
rh=flts(u,H); rs=flts(u,Sys);
norm(rh-rs,1) //hot restart
[ys1,x]=flts(u(:,1:4),Sys);ys2=flts(u(:,5:10),Sys,x);
norm([ys1,ys2]-rs,1)
//With H:
yh1=flts(u(:,1:4),H);yh2=flts(u(:,5:10),H,[u(:,2:4); yh1(:,2:4)]);
norm([yh1,yh2]-rh)

SEEE AALSO: 1ltitr 347, dsimul 333, rtitr 358

2.1.28 frep2tf transfer function realization from frequency response

CALLING SEQUENCE:

[h [,err]]=frep2tf(frq,repf,dg [,dom,tols,weight])

PARAMETERS:

frq : vector of frequencies in Hz.
repf : vector of frequency response
dg : degree of linear system
dom : time domain (’c’ or ’d’ or dt)
tols : a vector of size 3 giving the relative and absolute tolerance and the maximum number of iterations
(weight vector is not given a default penalization is used (when dom=’c’)).
weight : vector of weights on frequencies
h : SISO transfer function
err : error (for example if dom=’c’ sum(abs(h(2i*pi*frq) - rep)ˆ2)/size(frq,*))

DESCRIPTION:
Frequency response to transfer function conversion. The order of h is a priori given in dg which must be provided. The following linear system is solved in the least square sense.

weight(k) *(n( phi_k) - d(phi_k)*rep_k)=0, k=1,...,n

where phi_k= 2*%i*%pi*frq when dom=’c’ and phi_k=exp(2*%i*%pi*dom*frq) if not.
If the weight vector is not given a default penalization is used (when dom=’c’).
A stable and minimum phase system can be obtained by using function factors.

EXAMPLE:

s=poly(0,’s’);
h=syslin(’c’,(-1)/(sˆ3+5*s+20))
frq=0:0.05:3;repf=repfreq(h,frq);
clean(frep2tf(frq,repf,3))

Sys=ssrand(1,1,10);
frq=logspace(-3,2,200);
[frq,rep]=repfreq(Sys,frq); //Frequency response of Sys
[Sys2,err]=frep2tf(frq,rep,10);Sys2=clean(Sys2)//Sys2 obtained from freq. resp of Sys
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2
xbasc();bode(frq,[rep;rep2]) //Responses of Sys and Sys2
[sort(trzeros(Sys)), sort(roots(Sys2('num')))] //zeros
[sort(spec(Sys('A'))), sort(roots(Sys2('den')))] //poles

Dom=1/1000; // Sampling time
z=poly(0,'z');
h=syslin(dom,(z^2+0.5)/(z^3+0.1*z^2-0.5*z+0.08))
frq=(0:0.01:0.5)/dom; repf=repfreq(h,frq);
[Sys2,err]=frep2tf(frq,repf,3,dom);
[frq,rep2]=repfreq(Sys2,frq); //Frequency response of Sys2
xbasc();plot2d1("onn",frq',abs([repf;rep2]'))

see also: imrep2ss 341, arl2 321, time_id 367, armax 395, frfit 459

2.1.29 freq ____________________________ frequency response

Calling sequence:
[x]=freq(A,B,C [,D],f)
[x]=freq(NUM,DEN,f)

Parameters:
A, B, C, D : real matrices of respective dimensions nxn, nxp, mnx, mxp.
NUM, DEN : polynomial matrices of dimension mxp
x : real or complex matrix

Description:
x=freq(A,B,C [,D],f) returns a real or complex mxp*t matrix such that:
x(:,k*p:(k+1)*p)= C*inv(f(k)*eye()-A)*B + D.
Thus, for f taking values along the imaginary axis or on the unit circle x is the continuous or discrete
time frequency response of (A,B,C,D).
x=freq(NUM,DEN,f) returns a real or complex matrix x such that columns k*(p-1)+1 to k*p of
x contain the matrix NUM(f(k))./DEN(f(k))

Example:
s=poly(0,'s');
sys=(s+1)/(s^3-5*s+4)
rep=freq(sys("num"),sys("den"),[0,0.9,1.1,2,3,10,20])
[horner(sys,0),horner(sys,20)]
//
Sys=tf2ss(sys);
[A,B,C,D]=abcd(Sys);
freq(A,B,C,[0,0.9,1.1,2,3,10,20])

See also: repfreq 355, horner 490

2.1.30 freson __________________________ peak frequencies

Calling sequence:
fr=freson(h)
PARAMETERS:

- h : syslin list
- fr : vector of peak frequencies in Hz

DESCRIPTION:
returns the vector of peak frequencies in Hz for the SISO plant h

EXAMPLE:

```plaintext
h=syslin('c',-1+%s,(3+2*%s+%s^2)*(50+0.1*%s+%s^2))
fr=freson(h)
bode(h)
g=20*log(abs(repfreq(h,fr)))/log(10)
```

SEE ALSO: frep2tf 338, zgrid 160, h_norm 382

### 2.1.31 g_margin

**gain margin**

**CALLING SEQUENCE:**

```plaintext
[gm [,fr]]=g_margin(h)
```

**PARAMETERS:**

- h : syslin list representing a linear system in state-space or transfer form

**DESCRIPTION:**
returns gm, the gain margin in dB of h (SISO plant) and fr, the achieved corresponding frequency in Hz. The gain margin is values of the system gain at points where the nyquist plot crosses the negative real axis.

**EXAMPLE:**

```plaintext
h=syslin('c',-1+%s,3+2*%s+%s^2)
[g,fr]=g_margin(h)
[g,fr]=g_margin(h-10)
nyquist(h-10)
```

SEE ALSO: p_margin 352, black 85, chart 88, nyquist 115

### 2.1.32 gfrancis

**Francis equations for tracking**

**CALLING SEQUENCE:**

```plaintext
[L,M,T]=gfrancis(Plant,Model)
```

**PARAMETERS:**

- Plant : syslin list
- Model : syslin list
- L, M, T : real matrices

**DESCRIPTION:**
Given the the linear plant:

- \[ x' = Fx + Gu \]
- \[ y = Hx + Ju \]
and the linear model
\[
\begin{align*}
\dot{x}_m &= A\cdot x_m + B\cdot u_m \\
y_m &= C\cdot x_m + D\cdot u_m
\end{align*}
\]

the goal is for the plant to track the model i.e. \( e = y - y_m \rightarrow 0 \) while keeping stable the state \( x(t) \) of the plant. \( u \) is given by feedforward and feedback

\[
u = L\cdot x_m + M\cdot u_m + K\cdot (x - T\cdot x_m) = [K, L - K\cdot T]\cdot (x, x_m) + M\cdot u_m
\]

The matrices \( T, L, M \) satisfy generalized Francis equations

\[
\begin{align*}
    F\cdot T + G\cdot L &= T\cdot A \\
    H\cdot T + J\cdot L &= C \\
    G\cdot M &= T\cdot B \\
    J\cdot M &= D
\end{align*}
\]

The matrix \( K \) must be chosen as stabilizing the pair \( (F, G) \). See example of use in directory demos/tracking.

EXAMPLE:

```plaintext
Plant=ssrand(1,3,5);
[F,G,H,J]=abcd(Plant);
nw=4;nuu=2;A=rand(nw,nw);
st=maxi(real(spec(A)));A=A-st*eye(A);
B=rand(nw,nuu);C=2*rand(1,nw);D=0*rand(C*B);
Model=syslin('c',A,B,C,D);
[L,M,T]=gfrancis(Plant,Model);
norm(F*T+G*L-T*A,1)
norm(H*T+J*L-C,1)
norm(G*M-T*B,1)
norm(J*M-D,1)
```

SEE ALSO: lqg 344, ppol 354

2.1.33 **imrep2ss** — state-space realization of an impulse response

**CALLING SEQUENCE:**

```plaintext
[sl]=imrep2ss(v [,deg])
```

**PARAMETERS:**

- \( v \): vector coefficients of impulse response, \( v(:,k) \) is the \( k \)th sample
- \( deg \): integer (order required)
- \( sl \): syslin list

**DESCRIPTION:**

Impulse response to linear system conversion (one input). \( v \) must have an even number of columns.

**EXAMPLE:**

```plaintext
s=poly(0,'s');
H=[1/(s+0.5);2/(s-0.4)] //strictly proper
np=20;w=ldiv(H('num'),H('den'),np);
rep=[w(1:np)';w(np+1:2*np)']; //The impulse response
H1=ss2tf(imrep2ss(rep))
z=poly(0,'z');
```
H=(2*z^2-3.4*z+1.5)/(z^2-1.6*z+0.8)  //Proper transfer function
u=zeros(1,20);u(1)=1;
rep=rtitr(H('num'),H('den'),u);  //Impulse rep.
//  <=> rep=ldiv(H('num'),H('den'),20)
w=z*imrep2ss(rep)  //Realization with shifted impulse response
//  i.e strictly proper to proper
H2=ss2tf(w);

SEE ALSO:  frep2tf 338,  arl2 321,  time_id 367,  armax 395,  markp2ss 347,  ldiv 493

2.1.34  invsyslin _________________________________ system inversion

CALLING SEQUENCE :

[sl2]=invsylin(sl1)

PARAMETERS :

sl1,sl2 : syslin lists (linear systems in state space representation)

DESCRIPTION :
Utility function. Computes the state form of the inverse sl2 of the linear system sl1 (which is also given in state form).
The D-matrix is supposed to be full rank. Old stuff used by inv(S) when S is a syslin list.

SEE ALSO:  rowregul 357,  inv 516

2.1.35  kpure __________________________ continuous SISO system limit feedback gain

CALLING SEQUENCE :

g=kpure(sys)

PARAMETERS :

sys : SISO linear system (syslin)
g : constant

DESCRIPTION :
kpure(sys) computes the gains g such that the system sys feedback by g (sys/.g) has poles on imaginary axis.

EXAMPLE :

s=poly(0,'s');
h=syslin('c',(s-1)/(1+5*s+s^2+s^3))
xbasc();evans(h)
g=kpure(h)
hf=h/.g(1)
roots(denom(hf))

SEE ALSO :  evans 98,  krac2 343
2.1.36 krac2  

**continuous SISO system limit feedback gain**

**CALLING SEQUENCE:**

```plaintext
g=krac2(sys)
```

**PARAMETERS:**

- `sys`: SISO linear system (syslin)
- `g`: constant

**DESCRIPTION:**

`krac2(sys)` computes the gains `g` such that the system `sys` feedback by `g` (`sys/.g`) has 2 real equal poles.

**EXAMPLE:**

```plaintext
h=syslin('c',352*poly(-5,'s')/poly([0,0,2000,200,25,1],'s','c'));
xbasc();evans(h,100)
g=krac2(h)
hf1=h/.g(1);roots(denom(hf1))
hf2=h/.g(2);roots(denom(hf2))
```

**SEE ALSO:** evans 98, kpure 342

2.1.37 lin  

**linearization**

**CALLING SEQUENCE:**

```plaintext
[A,B,C,D]=lin(sim,x0,u0)
[sl]=lin(sim,x0,u0)
```

**PARAMETERS:**

- `sim`: function
- `x0`, `u0`: vectors of compatible dimensions
- `A,B,C,D`: real matrices
- `sl`: syslin list

**DESCRIPTION:**

Linearization of the non-linear system `[y,xdot]=sim(x,u)` around `x0,u0`. `sim` is a function which computes `y` and `xdot`. The output is a linear system (syslin list) `sl` or the four matrices `(A,B,C,D)` For example, if `ftz` is the function passed to `ode` e.g. `[zd]=ftz(t,z,u)` and if we assume that `y=x` `[z]=ode(x0,t0,tf,list(ftz,u)` compute `x(tf)`. If `simula` is the following function:

```plaintext
deff(''[y,xd]=simula(x,u)'',''xd=ftz(tf,x,u); y=x;'');
```

the tangent linear system `sl` can be obtained by:

```plaintext
[A,B,C,D]=lin(simula,z,u)
sl=syslin('c',A,B,C,D,x0)
```

**EXAMPLE:**

```plaintext
deff(''[y,xdot]=sim(x,u)'',''xdot=[u*sin(x);-u*x^2];y=xdot (1)+xdot (2)''
sl=lin(sim,1,2);
```

**SEE ALSO:** external 38, derivat 487

Scilab Group  
April 1993  
343
2.1.38 lqe ............................. linear quadratic estimator (Kalman Filter)

**CALLING SEQUENCE:**

\[ [K, X] = \text{lqe}(P21) \]

**PARAMETERS:**

- \( P21 \): syslin list
- \( K, X \): real matrices

**DESCRIPTION:**

lqe returns the Kalman gain for the filtering problem in continuous or discrete time.

\( P21 \) is a syslin list representing the system \( P21 = [A, B1, C2, D21] \)

The input to \( P21 \) is a white noise with variance:

\[
\begin{bmatrix}
B1 & Q & S \\
B1' & D21' & \\
D21 & S' & R
\end{bmatrix}
\]

\( X \) is the solution of the stabilizing Riccati equation and \( A+K \cdot C2 \) is stable.

In continuous time:

\[
(A-S \cdot \text{inv}(R) \cdot C2) \cdot X + X \cdot (A-S \cdot \text{inv}(R) \cdot C2)' - X \cdot C2' \cdot \text{inv}(R) \cdot C2 \cdot X + Q - S \cdot \text{inv}(R) \cdot S' = 0
\]

\( K = -(X \cdot C2' + S) \cdot \text{inv}(R) \)

In discrete time:

\[
X = A \cdot X \cdot A' - (A \cdot X \cdot C2' + B1 \cdot D21') \cdot \text{pinv}(C2 \cdot X \cdot C2' + D21 \cdot D21') \cdot (C2 \cdot X \cdot A' + D21 \cdot B1') + B1 \cdot B1'
\]

\( K = -(A \cdot X \cdot C2' + B1 \cdot D21') \cdot \text{pinv}(C2 \cdot X \cdot C2' + D21 \cdot D21') \)

\( xhat(t+1) = E(x(t+1)|y(0),...,y(t)) \) (one-step predicted) satisfies the recursion:

\( xhat(t+1) = (A+K \cdot C2) \cdot xhat(t) - K \cdot y(t) \).

**SEE ALSO:** lqr 345

**AUTHOR:** F. D.

2.1.39 lqg ............................. LQG compensator

**CALLING SEQUENCE:**

\[ [K] = \text{lqg}(P, r) \]

**PARAMETERS:**

- \( P \): syslin list (augmented plant) in state-space form
- \( r \): 1x2 row vector = (number of measurements, number of inputs) (dimension of the 2,2 part of \( P \))
- \( K \): syslin list (controller)

**DESCRIPTION:**

lqg computes the linear optimal LQG (H2) controller for the "augmented" plant \( P = \text{syslin}('c', A, B, C, D) \) (continuous time) or \( P = \text{syslin}('d', A, B, C, D) \) (discrete time).

The function lqg2stan returns \( P \) and \( r \) given the nominal plant, weighting terms and variances of noises.

\( K \) is given by the following ABCD matrices: \( [A+B \cdot Kc+Kf \cdot C+Kf \cdot D \cdot Kc, -Kf, Kc, 0] \) where \( Kc = \text{lqr}(P12) \) is the controller gain and \( Kf = \text{lqe}(P21) \) is the filter gain. See example in lqg2stan.

**SEE ALSO:** lqg2stan 345, lqr 345, lqe 344, h_inf 381, obscont 349

**AUTHOR:** F. D.
CALLING SEQUENCE:

\[ [P, r] = \text{lqg2stan}(P22, \text{bigQ}, \text{bigR}) \]

PARAMETERS:

- \( P22 \): syslin list (nominal plant) in state-space form
- \( \text{bigQ} \): \([Q, S; S', N]\) (symmetric) weighting matrix
- \( \text{bigR} \): \([R, T; T', V]\) (symmetric) covariance matrix
- \( r \): 1x2 row vector = (number of measurements, number of inputs) (dimension of the 2,2 part of \( P \))
- \( P \): syslin list (augmented plant)

DESCRIPTION:

\text{lqg2stan} returns the augmented plant for linear LQG (H2) controller design. \( P22 = \text{syslin}(\text{dom}, A, B2, C2) \) is the nominal plant; it can be in continuous time (\( \text{dom} = 'c' \)) or discrete time (\( \text{dom} = 'd' \)).

\[
\begin{align*}
\cdot & \\
x &= Ax + w1 + B2u \\
y &= C2x + w2
\end{align*}
\]

for continuous time plant.

\[
\begin{align*}
\cdot & \\
x[n+1] &= A x[n] + w1 + B2u \\
y &= C2x + w2
\end{align*}
\]

for discrete time plant.

The (instantaneous) cost function is \([x' u'] \text{bigQ} [x;u]\).

The covariance of \([w1; w2] = E[w1;w2] = \text{bigR}\)

If \([B1; D21]\) is a factor of \( \text{bigQ} \), \([C1, D12]\) is a factor of \( \text{bigR} \) and \([A, B2, C2, D22]\) is a realization of \( P22 \), then \( P \) is a realization of \([A, [B1, B2], [C1, -C2], [0, D12; D21, D22]]\). The (negative) feedback computed by \text{lqg} stabilizes \( P22 \), i.e. the poles of \( \text{cl} = P22 / .K \) are stable.

EXAMPLE:

\[
\begin{align*}
\text{ny} &= 2; \text{nu} = 3; \text{nx} = 4; \\
P22 &= \text{ssrand}(\text{ny}, \text{nu}, \text{nx}); \\
\text{bigQ} &= \text{rand}(\text{nx}+\text{nu}, \text{nx}+\text{nu}); \text{bigQ} = \text{bigQ}\' \text{bigQ}'; \\
\text{bigR} &= \text{rand}(\text{nx}+\text{ny}, \text{nx}+\text{ny}); \text{bigR} = \text{bigR}\' \text{bigR}'; \\
[P, r] &= \text{lqg2stan}(P22, \text{bigQ}, \text{bigR}); \text{K} = \text{lqg}(P, r); \\
\text{spec}(\text{h_cl}(P, r, \text{K})) &= \text{spec}(\text{cl}('A')) \\
\text{s} &= \text{poly}(0, 's') \\
\text{lqg2stan}(1/(\text{s}+2), \text{eye}(2,2), \text{eye}(2,2))
\end{align*}
\]

SEE ALSO: \( \text{lqg} 344, \text{lqr} 345, \text{lqe} 344, \text{obscont} 349, \text{h_inf} 381, \text{augment} 373, \text{fstabst} 378, \text{feedback} 335 \)

AUTHOR: F.D.

### 2.1.41 lqr

**LQ compensator (full state)**

CALLING SEQUENCE:

\[ [K, X] = \text{lqr}(P12) \]
PARAMETERS :

\( P_{12} : \) syslin list (state-space linear system)
\( K, X : \) two real matrices

DESCRIPTION :

\( lqr \) computes the linear optimal LQ full-state gain for the plant \( P_{12} = [A, B2, C1, D12] \) in continuous or discrete time.

\( P_{12} \) is a syslin list (e.g. \( P_{12} = \text{syslin('c', A, B2, C1, D12))} \)).

The cost function is l2-norm of \( z' z \) with \( z = C1 x + D12 u \) i.e. \([x, u]' * \text{BigQ} * [x; u]\)
where

\[
\begin{bmatrix}
C1' \\
D12'
\end{bmatrix} \begin{bmatrix}
Q & S \\
S' & R
\end{bmatrix} = \begin{bmatrix}
\end{bmatrix}
\]

The gain \( K \) is such that \( A + B2 K \) is stable.
\( X \) is the stabilizing solution of the Riccati equation.

For a continuous plant:

\[
(A - B2 \times \text{inv}(R) \times S')' \times X + X' (A - B2 \times \text{inv}(R) \times S') - X \times B2 \times \text{inv}(R) \times B2' \times X - Q - S \times \text{inv}(R) \times S' = 0
\]

\( K = -\text{inv}(R) \times (B2' \times X + S) \)

For a discrete plant:

\[
X = A' \times X \times A - (A' \times X \times B2 + C1' \times D12) \times \text{pinv}(B2' \times X \times B2 + D12' \times D12) - (B2' \times X \times A + D12' \times C1) + C1' \times C1;
\]

\( K = -\text{pinv}(B2' \times X \times B2 + D12' \times D12) \times (B2' \times X \times A + D12' \times C1) \)

An equivalent form for \( X \) is

\[
X = Abar' \times \text{inv}(\text{inv}(X)) + B2' \times \text{inv}(R) \times B2' \times Abar + Qbar
\]

with \( Abar = A - B2 \times \text{inv}(R) \times S' \) and \( Qbar = Q - S \times \text{inv}(R) \times S' \)

The 3-blocks matrix pencils associated with these Riccati equations are:

\[
\begin{bmatrix}
| I 0 0 | & | A 0 B2 | \\
| 0 A' 0 | & | -Q I -S | \\
| 0 B2' 0 | & | S' 0 R |
\end{bmatrix}
\begin{bmatrix}
| I 0 0 | & | A 0 B2 |
| 0 I 0 | & | -Q -A' -S |
| 0 0 0 | & | S' -B2' R |
\end{bmatrix}
\]

Caution: It is assumed that matrix \( R \) is non singular. In particular, the plant must be tall (number of outputs >= number of inputs).

EXAMPLE :

\[
A = \text{rand}(2, 2); B = \text{rand}(2, 1); // two states, one input
Q = \text{diag}([2, 5]); R = 2; // Usual notations \( x'Qx + u'Ru \)
\text{Big} = \text{sysdiag}(Q, R); // Now we calculate \( C1 \) and \( D12 \)
[w, wp] = \text{fullrf}({\text{Big}}); C1 = w(:, 1:2); D12 = w(:, 3); // \([C1, D12]' *[C1, D12] = \text{Big} \)
P = \text{syslin}('c', A, B, C1, D12); // The plant (continuous-time)
[K, X] = \text{lqr}(P)
\text{spec}(A + B \times K) // check stability
\text{norm}(A' \times X \times A - X \times B \times \text{inv}(R) \times B' \times X + Q, 1) // Riccati check
P = \text{syslin}('d', A, B, C1, D12); // Discrete time plant
[K, X] = \text{lqr}(P)
\text{spec}(A + B \times K) // check stability
\text{norm}(A' \times X \times A - (A' \times X \times B) \times \text{pinv}(B' \times X \times B + R) \times (B' \times X \times A) + Q - X, 1) // Riccati check
\]

SEE ALSO: \( lqe 344, \text{gcare} 379, \text{leqr} 383 \)

AUTHOR: F.D. Scilab Group April 1993 346
2.1.42 \textbf{ltitr} \hspace{1cm} \textit{discrete time response (state space)}

\begin{verbatim}
CALLING SEQUENCE :

[X]=ltitr(A, B, U, [x0])
[xf, X]=ltitr(A, B, U, [x0])

PARAMETERS :
A, B : real matrices of appropriate dimensions
U, X : real matrices
x0, xf : real vectors (default value=0 for x0)

DESCRIPTION :

calculates the time response of the discrete time system
\[ x[t+1] = Ax[t] + Bu[t]. \]

The inputs ui's are the columns of the U matrix
U=[u0, u1, ..., un];

x0 is the vector of initial state (default value : 0);
X is the matrix of outputs (same number of columns as U).
X=[x0, x1, x2, ..., xn]

xf is the vector of final state xf=X[n+1]

EXAMPLE :

A=eye(2, 2); B=[1; 1];
x0=[-1; -2];
u=[1, 2, 3, 4, 5];
x=ltitr(A, B, u, x0)
x1=A*x0+B*u(1)
x2=A*x1+B*u(2)
x3=A*x2+B*u(3) //....

SEE ALSO: \texttt{rtitr 358, flts 336}
\end{verbatim}

2.1.43 \textbf{markp2ss} \hspace{1cm} \textit{Markov parameters to state-space}

\begin{verbatim}
CALLING SEQUENCE :

[sl]=markp2ss(markpar, n, nout, nin)

PARAMETERS :
markpar : matrix
n, nout, nin : integers
Sl : syslin list

DESCRIPTION :

given a set of n Markov parameters stacked in the (row)-matrix markpar of size noutX(n*nin)
markp2ss returns a state-space linear system sl (syslin list) such that with [A, B, C, D]=abcd(sl):

Scilab Group \hspace{1cm} April 1993
\end{verbatim}
C*B = markpar(1:nout,1:nin),
C*A*B = markpar(1:nout,nin+1:2*nin), ...

EXAMPLE:

W=ssrand(2,3,4);  // random system with 2 outputs and 3 inputs
[a,b,c,d]=abcd(W);
markpar=[c*b,c*a*b,c*a^2*b,c*a^3*b,c*a^4*b];
S=markp2ss(markpar,5,2,3);
[A,B,C,D]=abcd(S);
Markpar=[C*B,C*A*B,C*A^2*B,C*A^3*B,C*A^4*B];
norm(markpar-Markpar,1)
// Caution... c*a^5*b is not C*A^5*B!

SEE ALSO: f rep2tf 338, tf2ss 367, imrep2ss 341

2.1.44 minreal ______________________________ minimal balanced realization

CALLING SEQUENCE:

slb=minreal(sl [,tol])

PARAMETERS:

sl, slb : syslin lists
tol : real (threshold)

DESCRIPTION:

[ae, be, ce] = minreal(a, b, c, domain [, tol]) returns the balanced realization of linear system sl (syslin list).
sl is assumed stable.
tol threshold used in equil1.

EXAMPLE:

A=[-eye(2,2), rand(2,2); zeros(2,2), -2*eye(2,2)];
B=[rand(2,2); zeros(2,2)]; C=rand(2,4);
sl=syslin('c', A, B, C);
slb=minreal(sl);
ss2tf(sl)
ss2tf(slb)
ctr_gram(sl)
clean(ctr_gram(slb))
clean(obs_gram(slb))

SEE ALSO: minss 348, balreal 322, arhnk 321, equil 334, equil1 335

AUTHOR: S. Steer INRIA 1987

2.1.45 minss _________________________________ minimal realization

CALLING SEQUENCE:

[slc]=minss( sl [,tol])

PARAMETERS:

Slcbab Group  April 1993 348
nl, slc: syslin lists (linear systems in state-space form)
tol: real (threshold for rank determination (see contr))

DESCRIPTION:
minss returns in slc a minimal realization of nl.

EXAMPLE:
sl=syslin('c',[1 0;0 2],[1;0],[2 1]);
ssprint(sl);
ssprint(minss(sl))

SEE ALSO: contr 328, minreal 348, arhnk 321, contrss 328, obsvss 352, balreal 322

2.1.46 obsv_gram: Observability gramian

CALLING SEQUENCE:

Go=obsv_gram(A,C [,dom])
Go=obsv_gram(sl)

PARAMETERS:
A, C: real matrices (of appropriate dimensions)
dom: string ("d" or "c" (default value))
sl: syslin list

DESCRIPTION:
Observability gramian of the pair (A, C) or linear system sl (syslin list). dom is the domain which can be
"c": continuous system (default)
"d": discrete system

\[ Go = \int_0^\infty e^{At} C' C e^{At} dt \quad Go = \sum_{k=0}^{\infty} A^k C' C A^k \]

EXAMPLE:
A=-diag(1:3);C=rand(2,3);
Go=obsv_gram(A,C,'c'); // <=> w=syslin('c',A,[],C); Go=obsv_gram(w);
norm(lyap(A,-C'*C,'c')-Go,1)
A=A/4; Go=obsv_gram(A,C,'d'); // discrete time case
norm(lyap(A,-C'*C,'d')-Go,1)

SEE ALSO: ctr_gram 330, obsvss 352, obsv_mat 351, lyap 522

2.1.47 obscont: Observer based controller

CALLING SEQUENCE:

[K]=obscont(P,Kc,Kf)
[J,r]=obscont(P,Kc,Kf)
PARAMETERS:

P : syslin list (nominal plant) in state-space form, continuous or discrete time
Kc : real matrix, (full state) controller gain
Kf : real matrix, filter gain
K : syslin list (controller)
J : syslin list (extended controller)
r : 1x2 row vector

DESCRIPTION:

obscont returns the observer-based controller associated with a nominal plant P with matrices \([A, B, C, D]\) (syslin list).
The full-state control gain is Kc and the filter gain is Kf. These gains can be computed, for example, by pole placement.
A+B*Kc and A+Kf*C are (usually) assumed stable.
K is a state-space representation of the compensator K: \(y\rightarrow u\) in:
\[\begin{align*}
xdot &= A x + B u, \\
y &= C x + D u, \\
zdot &= (A + Kf C) z - Kf y + B u, \\
u &= Kc z
\end{align*}\]
K is a linear system (syslin list) with matrices given by: \(K=[A+B*Kc+Kf*C+Kf*D*Kc,Kf,-Kc]\).
The closed loop feedback system \(Cl: v \rightarrow y\) with (negative) feedback \(K\) (i.e. \(y = Pu, u = v - Ky\), or \(xdot = A x + B u, y = C x + D u, zdot = (A + Kf C) z - Kf y + B u, u = v - Fz\)) is given by \(Cl = P/(.-K)\).
The poles of \(Cl\) (spec(cl('A'))) are located at the eigenvalues of \(A+B*Kc\) and \(A+Kf*C\).
Invoked with two output arguments obscont returns a (square) linear system \(K\) which parametrizes all the stabilizing feedbacks via a LFT.
Let \(Q\) an arbitrary stable linear system of dimension \(r(2)\times r(1)\) i.e. number of inputs x number of outputs in P. Then any stabilizing controller \(K\) for \(P\) can be expressed as \(K=lft(J,r,Q)\). The controller which corresponds to \(Q=0\) is \(K=J(1:nu,1:ny)\) (this \(K\) is returned by \(K=obscont(P,Kc,Kf)\)). \(r\) is size(P) i.e the vector [number of outputs, number of inputs];

EXAMPLE:

\[\begin{align*}
ny=2;nu=3;nx=4;P=ssrand(ny,nu,nx);[A,B,C,D]=abcd(P); \\
Kc=-ppol(A,B,[-1,-1,-1,-1]); //Controller gain \\
Kf=-ppol(A',C',[-2,-2,-2,-2]);Kf=Kf'; //Observer gain \\
cl=P/.(-obscont(P,Kc,Kf)); //closed loop system \\
[J,r]=obscont(P,Kc,Kf); \\
Q=ssrand(nu,ny,3);Q('A')=Q('A')-(maxi(real(spec(Q('A')))))+0.5)*eye(Q('A')) //Q is a stable parameter \\
K=lft(J,r,Q); \\
spec(h_cl(P,K)) // closed-loop A matrix (should be stable);
\end{align*}\]

SEE ALSO: ppol 354, lqg 344, lqr 345, lqe 344, h_inf 381, lft 384, syslin 224, feedback 335, observer 350

AUTHOR: F.D.

2.1.48 observer ______________________________________ observer design

CALLING SEQUENCE:

Obs=observer(Sys,J) 
[Obs,U,m]=observer(Sys [,flag,alfa])

PARAMETERS:

Sys : syslin list (linear system)
J : nx x ny constant matrix (output injection matrix)
flag : character strings ('pp' or 'st' (default))
alfa : location of closed-loop poles (optional parameter, default=-1)
Obs : linear system (syslin list), the observer
U : orthogonal matrix (see dt_ility)
m : integer (dimension of unstable unobservable (st) or unobservable (pp) subspace)

DESCRIPTION:
Obs=observer(Sys,J) returns the observer Obs=syslin(td,A+J*C,[B+J*D,-J],eye(A)) obtained from Sys by a J output injection. (td is the time domain of Sys). More generally, observer returns in Obs an observer for the observable part of linear system Sys: dotx=A x + Bu, y=Cx + Du represented by a syslin list. Sys has nx state variables, nu inputs and ny outputs. Obs is a linear system with matrices [Ao,Bo,Identity], where Ao is no x no, Bo is no x (nu+ny), Co is no x no and no=nx-m.
Input to Obs is [u,y] and output of Obs is:
xhat=estimate of x modulo unobservable subsp. (case flag='pp') or
xhat=estimate of x modulo unstable unobservable subsp. (case flag='st')
case flag='st': z=H*x can be estimated with stable observer iff H*U(:,1:m)=0 and assignable poles of the observer are set to alfa(1),alfa(2),...
case flag='pp': z=H*x can be estimated with given error spectrum iff H*U(:,1:m)=0 all poles of the observer are assigned and set to alfa(1),alfa(2),...
If H satisfies the constraint: H*U(:,1:m)=0 (ker(H) contains unobs-subsp. of Sys) one has H*U=[0,H2] and the observer for z=H*x is H2*Obs with H2=H*U(:,m+1:nx) i.e. Co, the C-matrix of the observer for H*x, is Co=H2.
In the particular case where the pair (A,C) of Sys is observable, one has m=0 and the linear system U*Obs (resp. H*U*Obs) is an observer for x (resp. Hx). The error spectrum is alpha(1), alpha(2),..., alpha(nx).

EXAMPLE:
nc=5;nu=1;ny=1;un=3;us=2;Sys=ssrand(ny,nu,nc,list('dt',us,us,un));
//nx=5 states, nu=1 input, ny=1 output,
//un=3 unobservable states, us=2 of them unstable.
[Obs,U,m]=observer(Sys); //Stable observer (default)
W=U';H=W(m+1:nc,:);[A,B,C,D]=abcd(Sys); //H*U=[0,eye(no,no)];
Sys2=ss2tf(syslin('c',A,B,H)) //Transfer u-->z
Idu=eye(nu,nu);Sys3=ss2tf(H*U(:,m+1:$)*Obs*[Idu;Sys])
//Transfer u-->[u;y=Sys*u]-->Obs-->xhat-->HUxhat=zhat i.e. u-->output of Obs
//this transfer must equal Sys2, the u-->z transfer (H2=eye).

SEE ALSO: dt_ility 334, unobs 370, stabil 364

AUTHOR: F.D.

2.1.49 obsv_mat ___________________________ observability matrix

CALLING SEQUENCE:
[O]=obsv_mat(A,C)
[O]=obsv_mat(sl)

PARAMETERS:
A,C,O : real matrices
sl : syslin list

Scilab Group April 1993
DESCRIPTION:

\[ O = [C; CA; CA^2; \ldots; CA^{(n-1)}] \]

SEE ALSO: contrss 328, obsvss 352, obs_gram 349

\[ 2.1.50 \ \text{obsvss} \quad \text{observable part} \]

CALLING SEQUENCE:

\[
[Ao, Bo, Co] = \text{obsvss}(A, B, C [, tol]) \\
[slo] = \text{obsvss}(sl [, tol])
\]

PARAMETERS:

\( A, B, C, Ao, Bo, Co \): real matrices
\( sl, slo \) : syslin lists
\( tol \): real (threshold) (default value \( 100*\text{eps} \))

DESCRIPTION:

\( slo = (Ao, Bo, Co) \) is the observable part of linear system \( sl = (A, B, C) \) (syslin list)
\( tol \) threshold to test controllability (see \( \text{contr} \)); default value = \( 100*\text{eps} \)

SEE ALSO: contr 328, contrss 328, obsv_mat 351, obs_gram 349

\[ 2.1.51 \ \text{p\_margin} \quad \text{phase margin} \]

CALLING SEQUENCE:

\[
[p\_margin, fr] = \text{p\_margin}(h) \\
p\_margin = \text{p\_margin}(h)
\]

PARAMETERS:

\( h \): SISO linear system (syslin list).
\( p\_margin \): phase margin (in degree)
\( fr \): corresponding frequency (hz)

DESCRIPTION:

The phase margin is the values of the phase at points where the nyquist plot of \( h \) crosses the unit circle.

EXAMPLE:

\[
h = \text{syslin}('c', -1+s, 3+2*s+s^2) \\
[p, fr] = \text{p\_margin}(h) \\
[p, fr] = \text{p\_margin}(h+0.7) \\
\text{nyquist}(h+0.7) \\
t = (0:0.1:2*\pi); \text{plot2d}(
\sin(t), \cos(t), -3, '000')
\]

SEE ALSO: chart 88, black 85, g\_margin 340, nyquist 115

AUTHOR: S. S.
**2.1.52 pfss .partial fraction decomposition**

**CALLING SEQUENCE:**

\[
\text{elts=pfss}(S_l) \\
\text{elts=pfss}(S_l, rmax) \\
\text{elts=pfss}(S_l, 'cord') \\
\text{elts=pfss}(S_l, rmax, 'cord')
\]

**PARAMETERS:**

- \( S_l \): syslin list (state-space or transfer linear system)
- \( rmax \): real number controlling the conditioning of block diagonalization
- \( \text{cord} \): character string ‘c’ or ‘d’.

**DESCRIPTION:**

Partial fraction decomposition of the linear system \( S_l \) (in state-space form, transfer matrices are automatically converted to state-space form by \( \text{tf2ss} \)):

\( \text{elts} \) is the list of linear systems which add up to \( S_l \) i.e. \( \text{elts=}{\text{list}}(S_1, S_2, S_3, \ldots, S_n) \) with:

\[ S_l = S_1 + S_2 + \ldots + S_n. \]

Each \( S_i \) contains some poles of \( S \) according to the block-diagonalization of the \( A \) matrix of \( S \).

For non proper systems the polynomial part of \( S_l \) is put in the last entry of \( \text{elts} \).

If \( S_l \) is given in transfer form, it is first converted into state-space and each subsystem \( S_i \) is then converted in transfer form.

The \( A \) matrix is of the state-space is put into block diagonal form by function \( \text{bdiag} \). The optional parameter \( rmax \) is sent to \( \text{bdiag} \). If \( rmax \) should be set to a large number to enforce block-diagonalization.

If the optional flag \( \text{cord}='c' \) is given the elements in \( \text{elts} \) are sorted according to the real part (resp. magnitude if \( \text{cord}='d' \)) of the eigenvalues of \( A \) matrices.

**EXAMPLE:**

\[
W=\text{ssrand}(1,1,6); \\
\text{elts=pfss}(W); \\
W1=0; \text{for} \ k=1: \text{size} \ (\text{elts}), \ W1=W1+\text{ss2tf} \ (\text{elts}(k)); \text{end} \\
\text{clean} \ (\text{ss2tf} \ (W) - W1)
\]

**SEE ALSO:** pbig 523, bdiag 502, coffg 485, dtsi 377

**AUTHOR:** F.D.

---

**2.1.53 phasemag .phase and magnitude computation**

**CALLING SEQUENCE:**

\[
[\phi, \text{db}]=\text{phasemag}(z [,, \text{mod}])
\]

**PARAMETERS:**

- \( z \): matrix or row vector of complex numbers.
- \( \text{mod} \): character string
  - \( \text{mod}='c' \): “continuous” representation between -infinity and +360 degrees (default)
  - \( \text{mod}='m' \): representation between -360 and 0 degrees
- \( \phi \): phases (in degree) of \( z \).
- \( \text{db} \): magnitude (in Db)
**DESCRIPTION:**

*phasemag* computes the phases and magnitudes of the entries of a complex matrix. For `mod='c'` *phasemag* computes `phi(:,i+1)` to minimize the distance with `phi(:,i)`, i.e. it tries to obtain a "continuous representation" of the phase.

To obtain the phase between `-%pi` and `%pi` use `phi=atan(imag(z),real(z))`

**EXAMPLE:**

```plaintext
s=poly(0,'s');
h=syslin('c',1/((s+5)*(s+10)*(100+6*s+s*s)*(s+.3)));
[frq,rf] = repfreq(h,0.1,20,0.005);
xbasc(0);
plot2d(frq',phasemag(rf,'c')');
xbasc(1);
plot2d(frq',phasemag(rf,'m')');
```

**SEE ALSO:**  
repmat 355, gainplot 104, atan 167, bode 86

### 2.1.54  ppol ----------------------------------------------- pole placement

**CALLING SEQUENCE:**

```plaintext
K=ppol(A,B,poles)
```

**PARAMETERS:**

- `A, B`: real matrices of dimensions `nxn` and `nxm`.
- `poles`: real or complex vector of dimension `n`.
- `K`: real matrix (negative feedback gain)

**DESCRIPTION:**

*K=ppol(A,B,poles)* returns a `mxn` gain matrix `K` such that the eigenvalues of `A-B*K` are `poles`. The pair `(A,B)` must be controllable. Complex numbers in `poles` must appear in conjugate pairs.

An output-injection gain `F` for `(A,C)` is obtained as follows:

*Ft=ppol(A',C',poles); F=Ft'**

The algorithm is by P.H. Petkov.

**EXAMPLE:**

```plaintext
A=rand(3,3);B=rand(3,2);
F=ppol(A,B,[-1,-2,-3]);
spec(A-B*F)
```

**SEE ALSO:**  
canon 325, stabil 364

### 2.1.55  projsl --------------------------------------------- linear system projection

**CALLING SEQUENCE:**

```plaintext
[slp]=projsl(sl,Q,M)
```

**PARAMETERS:**

- `sl, slp`: syslin lists
- `Q, M`: matrices (projection factorization)
DESCRIPTION:

slp = projected model of sl where Q*M is the full rank factorization of the projection. If (A, B, C, D) is the representation of sl, the projected model is given by (M*A*Q, M*B, C*Q, D).

Usually, the projection Q*M is obtained as the spectral projection of an appropriate auxiliary matrix W e.g. W = product of (weighted) gramians or product of Riccati equations.

EXAMPLE:

```scilab
rand('seed',0);sl=ssrand(2,2,5);[A,B,C,D]=abcd(sl);poles=spec(A)
[Q,M]=pbig(A,0,'c'); //keeping unstable poles
slred=projsl(sl,Q,M);spec(slred('A'))
sl('D')=rand(2,2); //making proper system
trzeros(sl) //zeros of sl
wi=inv(sl); //wi=inverse in state-space
[q,m]=psmall(wi('A'),2,'d'); //keeping small zeros (poles of wi) i.e. abs(z)<2
slred2=projsl(sl,q,m);
trzeros(slred2) //zeros of slred2 = small zeros of sl
// Example keeping second order modes
A=diag([-1,-2,-3]);
sl=syslin('c',A,rand(3,2),rand(2,3));[nk2,W]=hankelsv(sl)
[Q,M]=pbig(W,nk2(2)-%eps,'c'); //keeping 2 eigenvalues of W
slr=projsl(sl,Q,M); //reduced model
hankelsv(slr)
```

SEE ALSO: pbig 523

AUTHOR: F. D.

2.1.56 repfreq .................................................... frequency response

CALLING SEQUENCE:

```scilab
[ [frq,] repf]=repfreq(sys,fmin,fmax [,step])
[ [frq,] repf]=repfreq(sys [,frq])
[ frq,repf,splitf]=repfreq(sys,fmin,fmax [,step])
[ frq,repf,splitf]=repfreq(sys [,frq])
```

PARAMETERS:

sys : syslin list : SIMO linear system
fmin, fmax : two real numbers (lower and upper frequency bounds)
frq : real vector of frequencies (Hz)
step : logarithmic discretization step
splitf : vector of indexes of critical frequencies.
repf : vector of the complex frequency response

DESCRIPTION:

repfreq returns the frequency response calculation of a linear system. If sys(s) is the transfer function of Sys, repf(k) equals sys(s) evaluated at \( s = i \cdot \text{frq}(k) \cdot 2 \cdot \pi \) for continuous time systems and at \( \exp(2 \cdot i \cdot \pi \cdot dt \cdot \text{frq}(k)) \) for discrete time systems (dt is the sampling period).

db(k) is the magnitude of repf(k) expressed in dB i.e. \( \text{db}(k) = 20 \cdot \log10(\text{abs}(\text{repf}(k))) \) and phi(k) is the phase of repf(k) expressed in degrees.

If fmin, fmax, step are input parameters, the response is calculated for the vector of frequencies frq given by: frq=[10.^((log10(fmin)):step:(log10(fmax))) fmax];

If step is not given, the output parameter frq is calculated by frq=calfrq(sys,fmin,fmax).
Vector `frq` is split into regular parts with the `split` vector. `frq(splitf(k):splitf(k+1)-1)` has no critical frequency. `sys` has a pole in the range `[frq(splitf(k)),frq(splitf(k)+1)]` and no poles outside.

**EXAMPLE:**

```plaintext
A=diag([-1,-2]); B=[1;1]; C=[1,1];
Sys=syslin('c',A,B,C);
frq=0:0.02:5;w=frq*2*%pi; //frq=frequencies in Hz ;w=frequencies in rad/sec;
[frq1,rep]=repfreq(Sys,frq);
[db,phi]=dbphi(rep);
Systf=ss2tf(Sys) //Transfer function of Sys
x=horner(Systf,w(2)*sqrt(-1)) // x is Systf(s) evaluated at s = i w(2)
rep=20*log(abs(x))/log(10) //magnitude of x in dB
db(2) // same as rep
ang=atan(imag(x),real(x)); //in rad.
ang=ang*180/%pi //in degrees
phi(2)
repf=repfreq(Sys,frq);
repf(2)-x
```

**SEE ALSO:** bode 86, freq 339, calfrq 324, horner 490, nyquist 115, dbphi 330

**AUTHOR:** S. S.

### 2.1.57 `ricc`  Scilab Function

**Calling Sequence:**

- `[X]=ricc(A,B,C,"cont")`
- `[X]=ricc(F,G,H,"disc")`

**Parameters:**

- `A,B,C`: real matrices of appropriate dimensions
- `F,G,H`: real matrices of appropriate dimensions
- `X`: real matrix
- "cont", "disc": imposed string (flag for continuous or discrete)

**Description:**

Riccati solver.

Continuous time:

- `X=ricc(A,B,C,'cont')`

  gives a solution to the continuous time ARE

- `A'*X+X*A-X*B*X+C=0`

  `B` and `C` are assumed to be nonnegative definite. `(A,G)` is assumed to be stabilizable with `G*G'` a full rank factorization of `B`.

  `(A,H)` is assumed to be detectable with `H*H'` a full rank factorization of `C`.

Discrete time:

- `X=ricc(F,G,H,'disc')`

  gives a solution to the discrete time ARE

```plaintext
Scilab Group April 1993 356
```
\[ X = F'XF - F'XG1((G2 + G1'XG1)^{-1})G1'XF + H \]

\( F \) is assumed invertible and \( G = G1*\text{inv}(G2)*G1' \).

One assumes \((F,G1)\) stabilizable and \((C,F)\) detectable with \( C'Ca \) full rank factorization of \( H \). Use preferably ric_desc.

**EXAMPLE:**

// Standard formulas to compute Riccati solutions
A=rand(3,3);B=rand(3,2);C=rand(3,3);C=C*C';R=rand(2,2);R=R*R'+eye();
B=B*inv(R)*B';
X=ricc(A,B,C,'cont');

\[
\begin{align*}
\text{H} &= [A -B; -C -A'] ; \\
[T,d] &= \text{gschur} (\text{eye} (H), H, 'cont'); T=T(:,:,1:d);
\end{align*}
\]

\[
X1=T(4:6,:) / T(1:3,:);
\]

\[
\text{norm}(X1-X,1)
\]

\[
[T,d] = \text{schur} (H, 'cont'); T=T(:,:,1:d);
\]

\[
X2=T(4:6,:) / T(1:3,:);
\]

\[
\text{norm}(X2-X,1)
\]

// Discrete time case
F=A;B=rand(3,2);G1=B;G2=R;G=G1/G2*G1';H=C;
X=ricc(F,G,H,'disc');

\[
\begin{align*}
\text{H1} &= [\text{eye}(3,3) G; \text{zeros}(3,3) F'] ; \\
\text{H2} &= [F \text{ zeros}(3,3); -H \text{ eye}(3,3)];
\end{align*}
\]

\[
[T,d] = \text{gschur} (H2, H1, 'disc'); T=T(:,:,1:d); X1=T(4:6,:) / T(1:3,:);
\]

\[
\text{norm}(X1-X,1)
\]

Fi=inv(F);
Hami=[Fi Fi*G; H*Fi F'+H*Fi*G];

\[
[T,d] = \text{schur} (\text{Hami}, 'd'); T=T(:,:,1:d);
\]

Fit=inv(F');
Ham=[F*G*Fit+H -G*Fit; -Fit*H Fit];

\[
[T,d] = \text{schur} (\text{Ham}, 'd'); T=T(:,:,1:d); X2=T(4:6,:) / T(1:3,:);
\]

\[
\text{norm}(X2-X,1)
\]

**SEE ALSO:** riccati 389, ric_desc 388, schur 533, gschur 513

### 2.1.58 rowregul ______________ removing poles and zeros at infinity

**CALLING SEQUENCE:**

\[
[\text{Stmp}, Ws] = \text{rowregul} (S1, alfa, beta)
\]

**PARAMETERS:**

- \( S1, \text{Stmp} \) : syslin lists
- \( alfa, beta \) : real numbers (new pole and zero positions)

**DESCRIPTION:**

computes a postfilter \( Ws \) such that \( \text{Stmp} = Ws * S1 \) is proper and with full rank \( D \) matrix.

Poles at infinity of \( S1 \) are moved to \( alfa \);

Zeros at infinity of \( S1 \) are moved to \( beta \);

\( S1 \) is assumed to be a right invertible linear system (syslin list) in state-space representation with possibly a polynomial \( D \) matrix.

This function is the dual of colregul (see function code).

**EXAMPLE:**

Scilab Group April 1993
s=%s;
w=[1/s,0;s/(s^3+2),2/s];
Sl=tf2ss(w);
[Stmp,Ws]=rowregul(Sl,-1,-2);
Stmp('D') // D matrix of Stmp
(clean(ss2tf(Stmp))

SEE ALSO: invsyslin 342, colregul 326

AUTHOR: F. D., R. N.

2.1.59 rtitr ______________________ discrete time response (transfer matrix)

CALLING SEQUENCE:

[y]=rtitr(Num,Den,u [,up,yp])

PARAMETERS:

Num,Den : polynomial matrices (resp. dimensions : nxm and nxn)

u : real matrix (dimension mx(t+1)

up,yp : real matrices (up dimension mx (maxi(degree(Den))) (default values=0), yp dimension nx (maxi(degree(Den))))

y : real matrix

DESCRIPTION:

y=rtitr(Num,Den,u [,up,yp]) returns the time response of the discrete time linear system with transfer matrix Den^(-1) Num for the input u, i.e y and u are such that Den y = Num u at t=0,1,...
If d1=maxi(degree(Den)), and d2=maxi(degree(Num)) the polynomial matrices Den(z) and Num(z) may be written respectively as:

D(z)= D_0 + D_1 z + ... + D_d1 z^d1

N(z)= N_0 + N_1 z + ... + N_d2 z^d2

and Den y = Num u is interpreted as the recursion:

D(0)y(t)+D(1)y(t+1)+...+ D(d1)y(t+d1)= N(0) u(t) +....+ N(d2) u(t+d2)

It is assumed that D (d1) is non singular.
The columns of u are the inputs of the system at t=0,1,...,T:

u=[u(0) , u(1),...,u(T)]

The outputs at t=0,1,...,T+d1-d2 are the columns of the matrix y:

y=[y(0), y(1), .... y(T+d1-d2)]

up and yp define the initial conditions for t<0 i.e

up=[u(-d1), ..., u(-1) ]

yp=[y(-d1), ... y(-1) ]

Depending on the relative values of d1 and d2, some of the leftmost components of up, yp are ignored. The default values of up and yp are zero: up = 0*ones (m,d1), yp=0*ones (n,d1)

EXAMPLE:
z=poly(0,'z');
Num=1+z;Den=1+z;u=[1,2,3,4,5];
rtitr(Num,Den,u)-u

// Other examples
// siso
// causal
n1=1;d1=poly([1 1],'z','coeff'); // y(j)=-y(j-1)+u(j-1)
r1=[0 1 0 1 0 1 0 1 0 1 0];
r=rtitr(n1,d1,ones(1,10));norm(r1-r,1)
// hot restart
r=rtitr(n1,d1,ones(1,9),1,0);norm(r1(2:11)-r
// non causal
n2=poly([1 1 1],'z','coeff');d2=d1; // y(j)=-y(j-1)+u(j-1)+u(j)+u(j+1)
r2=[2 1 2 1 2 1 2];
r=rtitr(n2,d2,ones(1,10));norm(r2-r,1)
// hot restart
r=rtitr(n2,d2,ones(1,9),1,2);norm(r2(2:9)-r,1)

// MIMO example
// causal
d1=d1*diag([1 0.5]);n1=[1 3 1;2 4 1];r1=[5;14]*r1;
r=rtitr(n1,d1,ones(3,10));norm(r1-r,1)
//
r=rtitr(n1,d1,ones(3,9),[1;1;1],[0;0]);
norm(r1(:,2:11)-r,1)
// polynomial n1 (same ex.)
n1(1,1)=poly(1,'z','c');r=rtitr(n1,d1,ones(3,10));norm(r1-r,1)
//
r=rtitr(n1,d1,ones(3,9),[1;1;1],[0;0]);
norm(r1(:,2:11)-r,1)
// non causal
d2=d1;n2=n2*n1;r2=[5;14]*r2;
r=rtitr(n2,d2,ones(3,10));norm(r2-r)
//
r=rtitr(n2,d2,ones(3,9),[1;1;1],[10;28]);
norm(r2(:,2:9)-r,1)
//
// State-space or transfer
a = [0.21 , 0.63 , 0.56 , 0.23 , 0.31
 0.76 , 0.85 , 0.66 , 0.23 , 0.93
 0 , 0.69 , 0.73 , 0.22 , 0.21
 0.33 , 0.88 , 0.2 , 0.88 , 0.31
 0.67 , 0.07 , 0.54 , 0.65 , 0.36];
b = [0.29 , 0.5 , 0.92
 0.57 , 0.44 , 0.04
 0.48 , 0.27 , 0.48
 0.33 , 0.63 , 0.26
 0.59 , 0.41 , 0.41];
c = [0.28 , 0.78 , 0.11 , 0.15 , 0.84
 0.13 , 0.21 , 0.69 , 0.7 , 0.41];
d = [0.41 , 0.11 , 0.56
 0.88 , 0.2 , 0.59];
s=syslin('d',a,b,c,d);
h=ss2tf(s);num=h('num');den=h('den');den=den(1,1)*eye(2,2);
u=1;u(3,10)=0;r3=flts(u,s);
r=rtitr(num,den,u);norm(r3-r,1)

SEE ALSO: ltitr 347, exp 508, flts 336

### 2.1.60 sm2des  
**system matrix to descriptor**

**CALLING SEQUENCE:**

\[ [\text{Des}] = \text{sm2des}(\text{Sm}); \]

**PARAMETERS:**

- \( \text{Sm} \): polynomial matrix (pencil system matrix)
- \( \text{Des} \): descriptor system \((\text{list(‘des’,A,B,C,D,E)})\)

**DESCRIPTION:**

Utility function: converts the system matrix:

\[
\text{Sm} = [-sE + A \quad \text{B}; \\
\quad [ \quad C \quad D ]
\]

to descriptor system \(\text{Des}=\text{list(‘des’,A,B,C,D,E)}\).

SEE ALSO: ss2des 361, sm2ss 360

### 2.1.61 sm2ss  
**system matrix to state-space**

**CALLING SEQUENCE:**

\[ [\text{Sl}] = \text{sm2ss}(\text{Sm}); \]

**PARAMETERS:**

- \( \text{Sm} \): polynomial matrix (pencil system matrix)
- \( \text{Sl} \): linear system \((\text{syslin list})\)

**DESCRIPTION:**

Utility function: converts the system matrix:

\[
\text{Sm} = [-sI + A \quad \text{B}; \\
\quad [ \quad C \quad D ]
\]

to linear system in state-space representation \((\text{syslin list})\).

SEE ALSO: ss2des 361

### 2.1.62 specfact  
**spectral factor**

**CALLING SEQUENCE:**

\[ [\text{W0},L] = \text{specfact}(A,B,C,D) \]

**DESCRIPTION:**

Given a spectral density matrix \(\phi(s)\):

\[
\text{specfact} \quad \frac{\text{num}}{\text{den}} \quad \text{u} \\
\text{r=rtitr(num,den,u);norm(r3-r,1)}
\]

Scilab Group  
April 1993  
360
\[
-R + C*(s*I-A)^{-1}B + B'^*(-s*I-A')^{-1}C' \quad \text{with } R=D+D' > 0
\]

specfact computes \(W_0\) and \(L\) such that \(W(s)=W_0+L*(s*I-A)^{-1}*B\) is a spectral factor of \(\Phi(s)\), i.e.
\[\phi(s)=W'(-s)*W(s)\]

**EXAMPLE:**

\[
A=\begin{bmatrix} -1 & -2 \end{bmatrix}; B=[1;1]; C=[1,1]; D=1; s=poly(0,'s');
W1=syslin('c',A,B,C,D);
\]
\[
\phi=\text{gtild}(W1,'c')+W1;
\]
\[
\phi_0=\text{clean}(\text{ss2tf}(\phi));
\]
\[
\phi_0-\text{horner}(\phi_0,-s)' \quad // \text{check this is 0...}
\]
\[
[A,B,C,D]=\text{abcd}(W1);
\]
\[
[W0,L]=\text{specfact}(A,B,C,D);
\]
\[
W=syslin('c',A,B,L,W0);
\]
\[
Ws=\text{ss2tf}(W);
\]
\[
\text{horner}(Ws,-s)*Ws
\]

**SEE ALSO:**  
- \texttt{gtild} 380, \texttt{sfact} 497, \texttt{fspecg} 377
- \texttt{pol2des} 494, \texttt{tf2des} 390, \texttt{des2ss} 376

**AUTHOR:** F. D.

### 2.1.63 \texttt{ss2des} \hspace{1cm} (polynomial) state-space to descriptor form

**CALLING SEQUENCE:**

\[
S=\text{ss2des}(S1)
S=\text{ss2des}(S1,\text{flag})
\]

**PARAMETERS:**

\[
S1: \text{syslin list: proper or improper linear system.}
\]
\[
\text{flag : character string "withD"}
\]
\[
S: \text{list}
\]

**DESCRIPTION:**

Given the linear system in state-space representation \(S1\) (\text{syslin} list), with a \(D\) matrix which is either polynomial or constant, but not zero \texttt{ss2des} returns a descriptor system as \(('des',A,B,C,0,E)\) such that:
\[
S1=C*(s*E-A)^{-1}*B
\]

If the flag "withD" is given, \(S=('des',A,B,C,D,E)\) with a \(D\) matrix of maximal rank.

**EXAMPLE:**

\[
s=poly(0,'s');
\]
\[
G=[1/(s+1),s;1+s^2,3*s^3]; S1=\text{tf2ss}(G);
\]
\[
S=\text{ss2des}(S1);
\]
\[
S1=\text{ss2des}(S1,"withD")
\]
\[
\text{Des=des2ss}(S); \text{Des}(5)=\text{clean}(\text{Des}(5))
\]
\[
\text{Des1=des2ss}(S1)
\]

**SEE ALSO:**  
- \texttt{pol2des} 494, \texttt{tf2des} 390, \texttt{des2ss} 376

**AUTHOR:** F. D.
2.1.64  \texttt{ss2ss} \hfill \textit{state-space to state-space conversion, feedback, injection}

CALLING SEQUENCE:

\[ [\text{Sl1}, \text{right}, \text{left}] = \text{ss2ss}(\text{Sl}, T, [F, [G, [\text{flag}]]]) \]

PARAMETERS:

- \text{Sl} : linear system (syslin list) in state-space form
- \text{T} : square (non-singular) matrix
- \text{Sl1}, \text{right}, \text{left} : linear systems (syslin lists) in state-space form
- \text{F} : real matrix (state feedback gain)
- \text{G} : real matrix (output injection gain)

DESCRIPTION:

Returns the linear system \( \text{Sl1} = [A1, B1, C1, D1] \) where \( A1 = \text{inv}(T)A T, \ B1 = \text{inv}(T)B, \ C1 = C T, \ D1 = D \).

Optional parameters \text{F} \ and \text{G} \ are state feedback and output injection respectively. For example, \text{Sl1}=\text{ss2ss}(\text{Sl}, T, \text{F}) \) returns \text{Sl1} \ with:

\[ \text{Sl1} = \begin{pmatrix} T^{-1}(A + BF)T & T^{-1}(B) \\ (C + DF)T & D \end{pmatrix} \]

and \text{right} \ is a non singular linear system such that \text{Sl1}=\text{Sl} \times \text{right}.

\text{Sl1} \times \text{inv} \times \text{right} \ is a factorization of \text{Sl}.

\text{Sl1}=\text{ss2ss}(\text{Sl}, T, 0*\text{F}, \text{G}) \) returns \text{Sl1} \ with:

\[ \text{Sl1} = \begin{pmatrix} T^{-1}(A + GC)T & T^{-1}(B + GD, -G) \\ CT & D \end{pmatrix} \]

and \text{left} \ is a non singular linear system such that \text{Sl1}=\text{left} \times \text{Sl} \ (\text{right}=\text{Id} \ if \text{F}=0).

When both \text{F} \ and \text{G} \ are given, \text{Sl1}=\text{left} \times \text{Sl} \times \text{right}.

- When \text{flag} \ is used and \text{flag}=1 \ an output injection as follows is used

\[ \text{Sl1} = \begin{pmatrix} T^{-1}(A + GC)T & T^{-1}(B + GD, -G) \\ CT & D \end{pmatrix} \]

and then a feedback is performed, \text{F} \ must be of size \( (m+p, n) \) \((x \text{ in } \mathbb{R}^n, y \text{ in } \mathbb{R}^p, u \text{ in } \mathbb{R}^m)\).

\text{right} \ and \text{left} \ have the following property:

\[ \text{Sl1} = \text{left} \times \text{sysdiag} \times (\text{sys}, \text{eye}(p,p)) \times \text{right} \]

- When \text{flag} \ is used and \text{flag}=2 \ a feedback (\text{F} \ must be of size \( (m, n) \)) is performed and then the above output injection is applied. \text{right} \ and \text{left} \ have the following property:

\[ \text{Sl1} = \text{left} \times \text{sysdiag} \times (\text{sys} \times \text{right}, \text{eye}(p,p)) \]

EXAMPLE:

\text{Sl}=\text{ssrand}(2,2,5); \text{trzeros}(\text{Sl}) \quad \text{// zeros are invariant:}

\text{Sl1}=\text{ss2ss}(\text{Sl}, \text{rand}(5,5), \text{rand}(2,5), \text{rand}(5,2));

\text{trzeros} \times \text{(Sl1), \trzeros} \times (\text{rand}(2,2) \times \text{Sl1} \times \text{rand}(2,2))

\text{// output injection \ [ A + GC, (B + GD, -G)]}

\text{// \[ \begin{bmatrix} C & (D & 0) \end{bmatrix} \]

\text{p}=1, m=2, n=2; \text{sys} = \text{ssrand} \times (p, m, n);

\text{// feedback (m, n) first and then output injection.}

Scilab Group  \hfill April 1993 362
F1=rand(m,n);
G=rand(n,p);
[sys1,right,left]=ss2ss(sys,rand(n,n),F1,G,2);

// Sl1 equiv left*sysdiag(sys*right,eye(p,p))
res=clean(ss2tf(sys1) - ss2tf(left*sysdiag(sys*right,eye(p,p))))

// output injection then feedback (m+p,n)
F2=rand(p,n); F=[F1;F2];
[sys2,right,left]=ss2ss(sys,rand(n,n),F,G,1);

// Sl1 equiv left*sysdiag(sys,eye(p,p))*right
res=clean(ss2tf(sys2)-ss2tf(left*sysdiag(sys,eye(p,p))*right))

// when F2= 0; sys1 and sys2 are the same
F2=0*rand(p,n); F=[F1;F2];
[sys2,right,left]=ss2ss(sys,rand(n,n),F,G,1);
res=clean(ss2tf(sys2)-ss2tf(sys1))

SEE ALSO: projsl 354, feedback 335

2.1.65 ss2tf ______________ conversion from state-space to transfer function

CALLING SEQUENCE:

[h]=ss2tf(sl)
[Ds,NUM,chi]=ss2tf(sl)

PARAMETERS:

sl : linear system (syslin list)
h : transfer matrix

DESCRIPTION:

Called with three outputs [Ds,NUM,chi]=ss2tf(sl) returns the numerator polynomial matrix NUM, the characteristic polynomial chi and the polynomial part Ds separately i.e:

h=NUM/chi + Ds

Method:

One uses the characteristic polynomial and det(A+Eij)=det(A)+C(i,j) where C is the adjugate matrix of A.

EXAMPLE:

s=poly(0,'s');
h=[1,1/s;1/(s^2+1),s/(s^2-2)]
sl=tf2ss(h);
h=clean(ss2tf(sl))
[Ds,NUM,chi]=ss2tf(sl)

SEE ALSO: tf2ss 367, syslin 224, nlev 522, glever 512
2.1.66  st_ility

CALLING SEQUENCE:

\[ [\text{ns}, [\text{nc}, [,U [,Slo] ]]] = \text{st_ility}(Sl [,\text{tol}]) \]

PARAMETERS:

- \( Sl \) : syslin list (linear system)
- \( \text{ns} \) : integer (dimension of stabilizable subspace)
- \( \text{nc} \) : integer (dimension of controllable subspace \( \text{nc} \leq \text{ns} \))
- \( U \) : basis such that its \( \text{ns} \) (resp. \( \text{nc} \)) first components span the stabilizable (resp. controllable) subspace
- \( Slo \) : a linear system (syslin list)
- \( \text{tol} \) : threshold for controllability detection (see contr)

DESCRIPTION:

\( Slo = (U'^*A*U, U'^*B, C*U, D, U'^*x0) \) (syslin list) displays the stabilizable form of \( Sl \). Stabilizability means \( \text{ns} = \text{nx} \) (dim. of \( A \) matrix).

\[
\begin{bmatrix}
\ast, & \ast, & \ast \\
0, & \ast, & 0 \\
0, & 0, & 0
\end{bmatrix}
\]

where \((A11,B1)\) (dim(A11) = \( \text{nc} \)) is controllable and \( A22 \) (dim(A22) = \( \text{ns} - \text{nc} \)) is stable. "Stable" means real part of eigenvalues negative for a continuous linear system, and magnitude of eigenvalues lower than one for a discrete-time system (as defined by syslin).

EXAMPLE:

\[
A = \text{diag}([0.9, -2, 3]); B = [0; 0; 1]; Sl = \text{syslin}('c', A, B, []);
\]

\[
[\text{ns}, \text{nc}, U] = \text{st_ility}(Sl);
\]

\[
U'^*A*U
\]

\[
U'^*B
\]

SEE ALSO:  dt_ility 334,  contr 328,  stabil 364,  ssrand 220

AUTHOR: S. Steer INRIA 1988

2.1.67  stabil

CALLING SEQUENCE:

\[ F = \text{stabil}(A, B, \text{alfa}) \]
\[ K = \text{stabil}(\text{Sys}, \text{alfa}, \text{beta}) \]

PARAMETERS:

- \( A \) : square real matrix (\( nx \times nx \))
- \( B \) : real matrix (\( nx \times nu \))
- \( \text{alfa}, \text{beta} \) : real or complex vector (in conjugate pairs) or real number.
- \( F \) : real matrix (\( nx \times nu \))
- \( \text{Sys} \) : linear system (syslin list) (\( m \) inputs, \( p \) outputs).
- \( K \) : linear system (\( p \) inputs, \( m \) outputs)

Scilab Group  April 1993 364
svplot Scilab Function

DESCRIPTION:
F = stabil(A, B, alfa) returns a gain matrix F such that A+B*F is stable if pair (A, B) is stabilizable. Assignable poles are set to alfa(1), alfa(2), .... If (A,B) is not stabilizable a warning is given and assignable poles are set to alfa(1), alfa(2), .... If alfa is a number all eigenvalues are set to this alfa (default value is alfa=-1).
K = stabil(Sys, alfa, beta) returns K, a compensator for Sys such that (A,B) -controllable eigenvalues are set to alfa and (C,A) -observable eigenvalues are set to beta. All assignable closed loop poles (which are given by the eigenvalues of Aclosed=h_cl(Sys,K) are set to alfa(i)'s and beta(j)'s.

EXAMPLE:
// Gain:
Sys=srand(0,2,5,list('st',2,3,3));
A=sys('A');B=sys('B');F=stabil(A,B);
spec(A) //2 controllable modes 2 unstable uncontrollable modes
//and one stable uncontrollable mode
spec(A+B*F) //the two controllable modes are set to -1.
// Compensator:
Sys=srand(3,2,5,list('st',2,3,3)); //3 outputs, 2 inputs, 5 states
//2 controllables modes, 3 controllable or stabilizable modes.
K=stabil(Sys,-2,-3); //Compensator for Sys.
spec(Sys('A'))
spec(h_cl(Sys,K)) //K Stabilizes what can be stabilized.

SEE ALSO: st_ility 364, contr 328, ppol 354

2.1.68 svplot ________________________________________ singular-value sigma-plot

CALLING SEQUENCE:
[SVM]=svplot(sl,[w])

PARAMETERS:
sl : syslin list (continuous, discrete or sampled system)
w : real vector (optional parameter)

DESCRIPTION:
computes for the system sl=([A,B,C,D]) the singular values of its transfer function matrix:
\[ G(jw) = C(jw*I-A)B^{-1}+D \]
or
\[ G(exp(jw)) = C(exp(jw)*I-A)B^{-1}+D \]
or
\[ G(exp(jwT)) = C(exp(jw*T)*I-A)B^{-1}+D \]
evaluated over the frequency range specified by w. (T is the sampling period, T=sl('dt') for sampled systems).
sl is a syslin list representing the system [A,B,C,D] in state-space form. sl can be continuous or discrete time or sampled system.
The i-th column of the output matrix SVM contains the singular values of G for the i-th frequency value w(i).

SVM = svplot(sl)
is equivalent to

Scilab Group  April 1993 365
SVM = svplot(sl, logspace(-3, 3)) (continuous)

SVM = svplot(sl, logspace(-3, %pi)) (discrete)

**EXAMPLE:**

```plaintext
x = logspace(-3, 3);
y = svplot(ssrand(2, 2, 4));
xgrid(12)
xtitle("Singular values plot", "(Rd/sec)", "Db");
```

AUTHOR: F.D

### 2.1.69 sysfact- factorization

**CALLING SEQUENCE:**

```plaintext
[S, Series] = sysfact(Sys, Gain, flag)
```

**PARAMETERS:**

- `Sys`: syslin list containing the matrices [A, B, C, D].
- `Gain`: real matrix
- `flag`: string ‘post’ or ‘pre’
- `S`: syslin list
- `Series`: syslin list

**DESCRIPTION:**

If `flag` equals ‘post’, `sysfact` returns in `S` the linear system with ABCD matrices (A+B*Gain, B, Gain, I), and `Series`, a minimal realization of the series system Sys*S. If `flag` equals ‘pre’, `sysfact` returns the linear system (A+Gain*C, Gain, C, I) and `Series`, a minimal realization of the series system S*Sys.

AUTHOR: F.D

**EXAMPLE:**

```plaintext
//Kalman filter
Sys = ssrand(3, 2, 4); Sys('D') = rand(3, 2);
S = sysfact(Sys, lqr(Sys), 'post');
ww = minss(Sys*S);
ss2tf(gtild(ww)*ww), Sys('D')' *Sys('D')
//Kernel
Sys = ssrand(2, 3, 4);
[X, d, F, U, k, Z] = abinv(Sys);
ss2tf(Sys*Z)
ss2tf(Sys*sysfact(Sys, F, 'post')*U)
```

**SEE ALSO:** lqr 345, lqe 344
2.1.70 syssize ___________________________ size of state-space system

CALLING SEQUENCE:

[r,nx]=syssize(Sl)

PARAMETERS:

Sl: linear system (syslin list) in state-space
r: 1 x 2 real vector
nx: integer

DESCRIPTION:
returns in r the vector [number of outputs, number of inputs] of the linear system Sl. nx is the number of states of Sl.

SEE ALSO: size 211

2.1.71 tf2ss ____________________________ transfer to state-space

CALLING SEQUENCE:

sl=tf2ss(h [,tol])

PARAMETERS:

h: rational matrix
tol: may be the constant rtol or the 2 vector [rtol atol]
rtol: tolerance used when evaluating observability.
atol: absolute tolerance used when evaluating observability.
sl: linear system (syslin list sl=[A,B,C,D(s)])

DESCRIPTION:
transfer to state-space conversion:
h=C*(s*eye()-A)^-1*B+D(s)

EXAMPLE:

s=poly(0,'s');
H=[2/s,(s+1)/(s^2-5)];
Sys=tf2ss(H);
clean(ss2tf(Sys))

SEE ALSO: ss2tf 363, tf2des 390, des2tf 332

2.1.72 time_id ___________________________ SISO least square identification

CALLING SEQUENCE:

[H [,err]]=time_id(n,u,y)

PARAMETERS:

n: order of transfer
u: one of the following
trzeros Scilab Function

ul : a vector of inputs to the system
"impuls" : if y is an impulse response
"step" : if y is a step response.
y : vector of response.
H : rational function with degree n denominator and degree n-1 numerator if y(1)==0 or rational function
with degree n denominator and numerator if y(1)<0.
err : \| y - impuls(H,npt) \| ^2, where impuls(H,npt) are the npt first coefficients of im-
pulse response of H

DESCRIPTION :
Identification of discrete time response. If y is strictly proper (y(1)=0) then time_id computes the
least square solution of the linear equation: Den*y-Num*u=0 with the constraint coeff(Den,n):=1.
if y(1)^=0 then the algorithm first computes the proper part solution and then add y(1) to the solution

EXAMPLE :

z=poly(0,’z’);
h=(1-2*z)/(z^2-0.5*z+5)
rep=[0;ldiv(h(’num’),h(’den’),20)]; //impulse response
H=time_id(2,’impuls’,rep)
// Same example with flts and u
u=zeros(1,20);u(1)=1;
rep=flts(u,tf2ss(h)); //impulse response
H=time_id(2,u,rep)
// step response
u=ones(1,20);
rep=flts(u,tf2ss(h)); //step response.
H=time_id(2,’step’,rep)
H=time_id(3,u,rep) //with u as input and too high order required

AUTHOR : Serge Steer INRIA

SEE ALSO : imrep2ss 341, arl2 321, armax 395, frep2tf 338

2.1.73 trzeros __________________________ transmission zeros and normal rank

CALLING SEQUENCE :

[tr]=trzeros(Sl)
[nt,dt,rk]=trzeros(Sl)

PARAMETERS :
Sl : linear system (syslin list)
nt : complex vectors
dt : real vector
rk : integer (normal rank of Sl)

DESCRIPTION :
Called with one output argument, trzeros(Sl) returns the transmission zeros of the linear system Sl.
Sl may have a polynomial (but square) D matrix.
Called with 2 output arguments, trzeros returns the transmission zeros of the linear system Sl as
tr=nt./dt;
(Note that some components of dt may be zeros)
Called with 3 output arguments, rk is the normal rank of Sl
Transfer matrices are converted to state-space.
If $S_l$ is a (square) polynomial matrix $\text{trzeros}$ returns the roots of its determinant. For usual state-space system $\text{trzeros}$ uses the state-space algorithm of Emami-Naeni & Van Dooren. If $D$ is invertible the transmission zeros are the eigenvalues of the "$A$ matrix" of the inverse system: $A - B \times \text{inv}(D) \times C$; If $C \times B$ is invertible the transmission zeros are the eigenvalues of $N \times A \times M$ where $M \times N$ is a full rank factorization of $\text{eye}(A) - B \times \text{inv}(C \times B) \times C$; For systems with a polynomial $D$ matrix zeros are calculated as the roots of the determinant of the system matrix. Caution: the computed zeros are not always reliable, in particular in case of repeated zeros.

**EXAMPLE:**

```plaintext
W1 = ssrand(2, 2, 5); trzeros(W1)  // call trzeros
roots(det(systmat(W1)))  // roots of det(system matrix)
```

**SEE ALSO:** $\text{gspec}$ 514, $\text{kroneck}$ 517

### 2.1.74 $\text{ui\_observer}$ unknown input observer

**CALLING SEQUENCE:**

```
[UIobs, J, N] = ui_observer(Sys, reject, C1, D1)
[UIobs, J, N] = ui_observer(Sys, reject, C1, D1, flag, alfa, beta)
```

**PARAMETERS:**

- **Sys:** syslin list containing the matrices $(A,B,C2,D2)$.
- **reject:** integer vector, indices of inputs of $\text{Sys}$ which are unknown.
- **C1:** real matrix
- **D1:** real matrix. $C1$ and $D1$ have the same number of rows.
- **flag:** string ‘ge’ or ‘st’ (default) or ‘pp’.
- **alfa:** real or complex vector (loc. of closed loop poles)
- **beta:** real or complex vector (loc. of closed loop poles)

**DESCRIPTION:**

Unknown input observer.

$\text{Sys:} (w, u) \rightarrow y$ is a $(A,B,C2,D2)$ syslin linear system with two inputs $w$ and $u$, $w$ being the unknown input. The matrices $B$ and $D2$ of $\text{Sys}$ are (implicitly) partitioned as: $B=[B1,B2]$ and $D2=[D21,D22]$ with $B1=B(:,\text{reject})$ and $D21=D2(:,\text{reject})$ where reject=indices of unknown inputs. The matrices $C1$ and $D1$ define $z = C1 \times x + D1 \times (w, u)$, the to-be-estimated output. The matrix $D1$ is (implicitly) partitioned as $D1=[D11,D12]$ with $D11=D1(:,\text{reject})$

The data $(\text{Sys, reject, C1, D1})$ define a 2-input 2-output system:

- $\text{xdot} = A \times x + B1 \times w + B2 \times u$
- $z = C1 \times x + D11 \times w + D12 \times u$
- $y = C2 \times x + D21 \times w + D22 \times u$

An observer $(u, y) \rightarrow \text{zhat}$ is looked for the output $z$. $\text{flag}='ge'$ no stability constraints $\text{flag}='st'$ stable observer (default) $\text{flag}='pp'$ observer with pole placement alfa,beta = desired location of closed loop poles (default -1,-2) $J=y$-output to x-state injection. $N=y$-output to z-estimated output injection.
\textbf{zeropen Scilab Function}

UIobs = linear system \((u,y) \rightarrow z\) that such that: The transfer function: \((w,u) \rightarrow z\) equals the composed transfer function: \([0,1; UIobs Sys] (w,u) \rightarrow (u,y) \rightarrow z\) i.e. transfer function of system \([A,B,C1,D1]\) equals transfer function \(UIobs^* [0,1; Sys]\)

Stability (resp. pole placement) requires detectability (resp. observability) of \((A,C2)\).

\textbf{EXAMPLE :}

\begin{verbatim}
A=diag([3,-3,7,4,-4,8]);
B=[eye(3,3);zeros(3,3)];
C=[0,0,1,2,3;0,0,0,0,0,1];
D=[1,2,3;0,0,0,0,0,0];
rand('seed',0);w=ss2ss(syslin('c',A,B,C,D),rand(6,6));
[A,B,C,D]=abcd(w);
B=[B,matrix(1:18,6,3)];D=[D,matrix(-(1:6),2,3)];
reject=1:3;
Sys=syslin('c',A,B,C,D);
N1=[-2,-3];C1=-N1*C;D1=-N1*D;
nuw=length(reject);nu=size(Sys('B'),2)-nuw;
nuy=size(Sys('C'),1);nz=size(C1,1);
[UIobs,J,N]=ui_observer(Sys,reject,C1,D1);

W=[zeros(nu,nuw),eye(nu,nu);Sys];UIobsW=UIobs*W;
//(w,u) \rightarrow z=UIobs^* [0,1; Sys](w,u)
clean(ss2tf(UIobsW));
//wu_to_z=syslin('c',A,B,C1,D1);
clean(ss2tf(wu_to_z));
clean(ss2tf(wu_to_z)-ss2tf(UIobsW),1.d-7)

///2nd example////
(nx=2;ny=3;nwu=2;Sys=ssrand(ny,nwu,nx);
C1=rand(1,nx);D1=[0,1];
UIobs=ui_observer(Sys,1,C1,D1);
\end{verbatim}

\textbf{AUTHOR : F.D.}

\textbf{SEE ALSO :} cainv 323, ddp 331, abinv 318

\begin{verbatim}
2.1.75 unobs unobservable subspace
\end{verbatim}

\textbf{CALLING SEQUENCE :}

\[ [n, U] = \text{unobs}(A, C, [tol]) \]

\textbf{PARAMETERS :}

- \(A, C\) : real matrices
- \(tol\) : tolerance used when evaluating ranks (QR factorizations).
- \(n\) : dimension of unobservable subspace.
- \(U\) : orthogonal change of basis which puts \((A,B)\) in canonical form.

\textbf{DESCRIPTION :}

\[ [n, U] = \text{unobs}(A, C, [tol]) \] gives the unobservable form of a \((A,C)\) pair. The \(n\) first columns of \(U\) make a basis for the unobservable subspace. The \((2,1)\) block (made of last \(nx-n\) rows and \(n\) first columns) of \(U^*A^*U\) is zero and the \(n\) first columns of \(C^*U\) are zero.

\textbf{EXAMPLE :}

\begin{verbatim}
A=diag([1,2,3]);C=[1,0,0];
unobs(A,C)
\end{verbatim}

\textbf{SEE ALSO :} contr 328, contrss 328, canon 325, cont_mat 327, spantwo 535, dt_ility 334
CALLING SEQUENCE:

\[ [Z, U] = \text{zeropen}(\text{Sl}) \]

PARAMETERS:

- \text{Sl} : a linear system (syslin list in state-space form \([A, B, C, D]\))
- \text{Z} : matrix pencil \(Z = sE - A\)
- \text{U} : square orthogonal matrix

DESCRIPTION:

\( Z = sE - F \) is the zero pencil of the linear system \(S1\) with matrices \([A, B, C, D]\). Utility function.

With \(U\) row compression of \([B; D]\), i.e., \(U* [B; D] = [0; *] \); one has:

\[
U*[-sI+A |B; [ Z |0;
   C |D] = [ * |*]
\]

The zeros of \(Z\) are the zeros of \(S1\).

SEE ALSO: systmat 499, kroneck 517
2.2 Robust control toolbox
2.2.1 augment .................................................................................................................. augmented plant

CALLING SEQUENCE:

\[ [P, r] = \text{augment}(G) \]
\[ [P, r] = \text{augment}(G, \text{flag1}) \]
\[ [P, r] = \text{augment}(G, \text{flag1,flag2}) \]

PARAMETERS:

\( G \): linear system (syslin list), the nominal plant
\( \text{flag1} \): one of the following (upper case) character string: 'S', 'R', 'T', 'SR', 'ST', 'RT', 'SRT'
\( \text{flag2} \) : one of the following character string: 'o' (stands for 'output', this is the default value) or 'i' (stands for 'input').

\( P \): linear system (syslin list), the "augmented" plant
\( r \) : 1x2 row vector, dimension of \( P_{22} = G \)

DESCRIPTION:

If \( \text{flag1}='SRT' \) (default value), returns the "full" augmented plant

\[
\begin{bmatrix}
I & -G \\
0 & I
\end{bmatrix} \rightarrow 'S'
\]
\[
\begin{bmatrix}
0 & G \\
-------
I & -G
\end{bmatrix} \rightarrow 'R'
\]
\[
\begin{bmatrix}
I & -G \\
0 & I
\end{bmatrix} \rightarrow 'T'
\]

'S', 'R', 'T' refer to the first three (block) rows of \( P \) respectively.

If one of these letters is absent in \( \text{flag1} \), the corresponding row in \( P \) is missing.

If \( G \) is given in state-space form, the returned \( P \) is minimal. \( P \) is calculated by:

\[
[I,0,0;0,I,0;-I,0,I;I,0,0]*[I,-G;0,G]*[I,-I;0,0;0,I;0,0]
\]

The augmented plant associated with input sensitivity functions, namely

\[
\begin{bmatrix}
I & -I \\
G & -G
\end{bmatrix} \rightarrow 'R' \quad (G*input sensitivity)
\]
\[
\begin{bmatrix}
0 & I
\end{bmatrix} \rightarrow 'T' \quad (K*G*input sensitivity)
\]

is obtained by the command \([P, r] = \text{augment}(G, \text{flag,}'i')\). For state-space \( G \), this \( P \) is calculated by:

\[
[I,-I;0,0;0,I;0,0]+[0;I;0;I]*G*[I,-I]
\]

and is thus generically minimal.

Note that weighting functions can be introduced by left-multiplying \( P \) by a diagonal system of appropriate dimension, e.g., \( P = \text{sysdiag}(W1,W2,W3,\text{eye}(G))*P \).

Sensitivity functions can be calculated by \text{lft}. One has:

For output sensitivity functions \([P,r]=\text{augment}(P,'SRT');\) \( \text{lft}(P,r,K)=[\text{inv}(\text{eye}()+G*K);K*\text{inv}(\text{eye}()+G*K);G*K*\text{inv}(\text{eye}()+G*K)] \);

For input sensitivity functions \([P,r]=\text{augment}(P,'SRT','i');\) \( \text{lft}(P,r,K)=[\text{inv}(\text{eye}()+K*G);G*\text{inv}(\text{eye}()+K*G);K*G*\text{inv}(\text{eye}()+G*K)] \);

EXAMPLE:

\text{G=ssrand}(2,3,2); //Plant}
\text{K=ssrand}(3,2,2); //Compensator}
\text{[P,r]=augment(G,'T');}
\text{T=lft(P,r,K); //Complementary sensitivity function}
\text{Ktf=ss2tf(K);Gtf=ss2tf(G);}
\text{Ttf=ss2tf(T);T11=Ttf(1,1);}
\text{Oloop=Gtf*Ktf;}
\text{Tn=Oloop*inv(eye(Oloop)+Oloop);}
clean(T1l-Tn(1,1));
//
[Pl,r]=augment(G,’T’,’i’);
T1=lft(Pi,r,K);T1tf=ss2tf(T1); //Input Complementary sensitivity function
Oloop=Ktf*Gtf;
T1n=Oloop*inv(eye(Oloop)+Oloop);
clean(T1tf(1,1)-T1n(1,1))

SEE ALSO:  lft 384,  sensi 390

2.2.2  bstap ___________________________ hankel approximant

CALLING SEQUENCE :

[Q]=bstap(Sl)

PARAMETERS :

sl : linear system (syslin list) assumed continuous-time and anti-stable.
Q : best stable approximation of Sl (syslin list).

DESCRIPTION :
Computes the best approximant Q of the linear system Sl

\[ ||S_I - Q||_\infty = ||T|| \]

where is the H-infinity norm of the Hankel operator associated with Sl.

SEE ALSO:  syslin 224

2.2.3  ccontrg ___________________________ central H-infinity controller

CALLING SEQUENCE :

[K]=ccontrg(P,r,gamma);

PARAMETERS :

P : syslin list (linear system in state-space representation)
r : 1x2 row vector, dimension of the 2,2 part of P
gamma : real number

DESCRIPTION :
returns a realization K of the central controller for the general standard problem in state-space form.
Note that gamma must be > gopt (output of gamitg)
P contains the parameters of plant realization (A,B,C,D) (syslin list) with

\[ B = ( B1 , B2 ) , \quad C = ( C1 ) , \quad D = ( D11 \ D12) \]
\[ ( C2 ) \quad ( D21 \ D22) \]

r(1) and r(2) are the dimensions of D22 (rows x columns)

SEE ALSO:  gamitg 378,  h_inf 381

AUTHOR:  P. Gahinet (INRIA)
### 2.2.4 colinout

**inner-outer factorization**

**CALLING SEQUENCE:**

\[ \text{[Inn,X,Gbar]} = \text{colinout}(G) \]

**PARAMETERS:**

- \( G \): linear system \((\text{syslin list}) [A,B,C,D] \)
- \( \text{Inn} \): inner factor \((\text{syslin list}) \)
- \( \text{Gbar} \): outer factor \((\text{syslin list}) \)
- \( X \): row-compressor of \( G \) \((\text{syslin list}) \)

**DESCRIPTION:**

Inner-outer factorization (and column compression) of \((lxp) G = [A,B,C,D] \) with \( l \leq p \).

\( G \) is assumed to be fat \((l \leq p)\) without zero on the imaginary axis and with a \( D \) matrix which is full row rank.

\( G \) must also be stable for having \( \text{Gbar} \) stable.

Dual of \text{rowinout}.

**SEE ALSO:** \text{syslin 224}, \text{rowinout 389}

### 2.2.5 copfac

**right coprime factorization**

**CALLING SEQUENCE:**

\[ \text{[N,M,XT,YT]} = \text{copfac}(G[,\text{polf, polc, tol}]) \]

**PARAMETERS:**

- \( G \): \text{syslin} list (continuous-time linear system)
- \( \text{polf}, \text{polc} \): respectively the poles of \( \text{XT} \) and \( \text{YT} \) and the poles of \( N \) and \( M \) (default values = -1).
- \( \text{tol} \): real threshold for detecting stable poles (default value \( 100*\%\text{eps} \))

\( N, M, \text{XT}, \text{YT} \): linear systems represented by \text{syslin} lists

**DESCRIPTION:**

\( \text{[N,M,XT,YT]} = \text{copfac}(G[,\text{polf, polc, [tol]}]) \) returns a right coprime factorization of \( G \).

\( G = N*M^\perp \) where \( N \) and \( M \) are stable, proper and right coprime. (i.e. \([N M]\) left-invertible with stability)

\( \text{XT} \) and \( \text{YT} \) satisfy:

\[ [\text{XT } -\text{YT}], [\text{M N}]' = \text{eye} \] (Bezout identity)

\( G \) is assumed stabilizable and detectable.

**SEE ALSO:** \text{syslin 224}, \text{lcf 383}

### 2.2.6 dcf

**double coprime factorization**

**CALLING SEQUENCE:**

\[ \text{[N,M,X,Y,NT,MT,XT,YT]} = \text{dcf}(G[,\text{polf, polc, [tol]}]) \]

**PARAMETERS:**

- \( G \): \text{syslin} list (continuous-time linear system)
- \( \text{polf}, \text{polc} \): respectively the poles of \( \text{XT} \) and \( \text{YT} \) and the poles of \( N \) and \( M \) (default values = -1).
tol : real threshold for detecting stable poles (default value $100 \times \text{eps}$).

DESCRIPTION:
returns eight stable systems $(N, M, X, Y, NT, MT, XT, YT)$ for the doubly coprime factorization

$$
egin{pmatrix}
    XT & -YT \\
    -NT & MT
\end{pmatrix}
\begin{pmatrix}
    M & Y \\
    M & X
\end{pmatrix}
= \text{eye}
$$

$G$ must be stabilizable and detectable.

SEE ALSO: copfac 375

2.2.7 des2ss ------------------------- descriptor to state-space

CALLING SEQUENCE:

\[
\text{[Sl]} = \text{des2ss}(A, B, C, D, E, [,\text{tol}])
\]

\[
\text{[Sl]} = \text{des2ss}(\text{Des})
\]

PARAMETERS:

$A, B, C, D, E$ : real matrices of appropriate dimensions

Des : list

Sl : syslin list

tol : real parameter (threshold) (default value $100 \times \text{eps}$).

DESCRIPTION:

Descriptor to state-space transform.

$\text{Sl} = \text{des2ss}(A, B, C, D, E)$ returns a linear system $\text{Sl}$ equivalent to the descriptor system $(E, A, B, C, D)$.

For index one $(E, A)$ pencil, explicit formula is used and for higher index pencils rowshuff is used.

$\text{Sl} = \text{des2ss}(\text{Des})$ with Des=list(’des’, $A$, $B$, $C$, $D$, $E$) returns a linear system $\text{Sl}$ in state-space form with possibly a polynomial $D$ matrix.

A generalized Leverrier algorithm is used.

EXAMPLE:

\[
s = \text{poly}(0, ’s’); G = [1 / (s - 1), s; 1, 2 / s^3];
\]

\[
\text{S1} = \text{tf2des}(G); \text{S2} = \text{tf2des}(G, ’withD’);
\]

\[
\text{W1} = \text{des2ss}(\text{S1}); \text{W2} = \text{des2ss}(\text{S2});
\]

\[
\text{clean}(\text{ss2tf}(\text{W1}))
\]

\[
\text{clean}(\text{ss2tf}(\text{W2}))
\]

SEE ALSO: des2tf 332, glever 512, rowshuff 532

2.2.8 dhnorm -------------------------- discrete H-infinity norm

CALLING SEQUENCE:

\[
\text{hinfnorm} = \text{dhnorm}(\text{sl}, [,\text{tol}], [,\text{normax}])
\]

PARAMETERS:

$\text{sl}$ : the state space system (syslin list) (discrete-time)

tol : tolerance in bisection step, default value 0.01

normax : upper bound for the norm, default value is 1000

hinfnorm : the discrete infinity norm of $\text{sl}$

DESCRIPTION:

produces the discrete-time infinity norm of a state-space system (the maximum over all frequencies on the unit circle of the maximum singular value).

SEE ALSO: h_norm 382, linfn 385

Scilab Group

April 1993

376
2.2.9 \texttt{dtsi} \hspace{1cm} \text{stable anti-stable decomposition}

\textbf{CALLING SEQUENCE:}

\[ [Ga, Gs, Gi] = \text{dtsi}(G, \text{tol}) \]

\textbf{PARAMETERS:}

- $G$: linear system \((\text{syslin list})\)
- $Ga$: linear system \((\text{syslin list})\) antistable and strictly proper
- $Gs$: linear system \((\text{syslin list})\) stable and strictly proper
- $Gi$: real matrix (or polynomial matrix for improper systems)
- $\text{tol}$: optional parameter for detecting stable poles. Default value: \(100 \times \text{eps}\)

\textbf{DESCRIPTION:}

returns the stable-antistable decomposition of $G$:

\[ G = Ga + Gs + Gi, \quad (Gi = G(\infty)) \]

$G$ can be given in state-space form or in transfer form.

\textbf{SEE ALSO:} \texttt{syslin 224}, \texttt{pbig 523}, \texttt{psmall 527}, \texttt{pfss 353}

2.2.10 \texttt{fourplan} \hspace{1cm} \text{augmented plant to four plants}

\textbf{CALLING SEQUENCE:}

\[ [P11, P12, P21, P22] = \text{fourplan}(P, r) \]

\textbf{PARAMETERS:}

- $P$: \text{syslin list} (linear system)
- $r$: \(1x2\) row vector, dimension of $P22$
- $P11, P12, P21, P22$: \text{syslin lists.}

\textbf{DESCRIPTION:}

Utility function.

$P$ being partitioned as follows:

\[
P = \begin{bmatrix}
P11 & P12 \\
P21 & P22
\end{bmatrix}
\]

with \(\text{size}(P22) = r\) this function returns the four linear systems $P11, P12, P21, P22$.

\textbf{SEE ALSO:} \texttt{lqg 344}, \texttt{lqg2stan 345}, \texttt{lqr 345}, \texttt{lqe 344}, \texttt{lft 384}

2.2.11 \texttt{fspecg} \hspace{1cm} \text{stable factorization}

\textbf{CALLING SEQUENCE:}

\[ [gm] = \text{fspecg}(g) \]

\textbf{PARAMETERS:}

- $g, gm$: \text{syslin lists} (linear systems in state-space representation)

\textbf{DESCRIPTION:}

returns $gm$ with $gm$ and $gm^-1$ stable such that:

\[ \text{gtild}(g) \ast g = \text{gtild}(gm) \ast gm \]

$g$ and $gm$ are continuous-time linear systems in state-space form. Imaginary-axis poles are forbidden.
### 2.2.12 fstabst

**Youla’s parametrization**

**Calling Sequence:**

\[
[J] = \text{fstabst}(P, r)
\]

**Parameters:**

- \( P \): `syslin` list (linear system)
- \( r \): 1x2 row vector, dimension of \( P_{22} \)
- \( J \): `syslin` list (linear system in state-space representation)

**Description:**

Parameterization of all stabilizing feedbacks.

\( P \) is partitioned as follows:

\[
P = \begin{bmatrix}
P_{11} & P_{12} \\
P_{21} & P_{22}
\end{bmatrix}
\]

(in state-space or transfer form: automatic conversion in state-space is done for the computations)

\( r \) = size of \( P_{22} \) subsystem, (2,2) block of \( P \)

\[
J = \begin{bmatrix}
J_{11} & J_{12} \\
J_{21} & J_{22}
\end{bmatrix}
\]

\( K \) is a stabilizing controller for \( P \) (i.e. \( P_{22} \)) if \( K = \text{lft}(J, r, Q) \) with \( Q \) stable.

The central part of \( J \), \( J_{11} \) is the lqg regulator for \( P \).

This \( J \) is such that defining \( T \) as the 2-port \( \text{lft} \) of \( P \) and \( J : [T, rT] = \text{lft}(P, r, J, r) \) one has that \( T_{12} \) is inner and \( T_{21} \) is co-inner.

**Example:**

\[
y = 2; nu = 3; nx = 4;
P_{22} = \text{ssrand}(ny, nu, nx);
bigQ = \text{rand}(nx + nu, nx + nu) \cdot \text{rand}(nx + nu, nx + nu);
bigR = \text{rand}(nx + ny, nx + ny) \cdot \text{rand}(nx + ny, nx + ny);
[P, r] = \text{lqg2stan}(P_{22}, bigQ, bigR);
s = \text{ssrand}(nu, ny, 1);
s('A') = -1; // Stable \ Q
K = \text{lft}(J, r, Q);
A = \text{h_cl}(P, r, K); \ spec(A)
\]

**See Also:** `obscont 349`, `lft 384`, `lqg 344`, `lqg2stan 345`

### 2.2.13 gamitg

**H-infinity gamma iterations**

**Calling Sequence:**

\[
[\text{gopt}] = \text{gamitg}(G, r, \text{prec} [, \text{options}])
\]

**Parameters:**

- \( G \): `syslin` list (plant realization)
- \( r \): 1x2 row vector, dimension of \( G_{22} \)
- \( \text{prec} \): desired relative accuracy on the norm
- \( \text{options} \): string ‘t’
- \( \text{gopt} \): real scalar, optimal H-infinity gain

**See Also:** `gamitg 346`, `h_cl 311`, `h_norm 347`, `lft 384`, `lqg 344`, `lqg2stan 345`, `gamitg 346`
**DESCRIPTION:**

\[ \text{gopt}=\text{gamitg}(G,r,\text{prec} [,\text{options}]) \] returns the H-infinity optimal gain \( \text{gopt} \).

\( G \) contains the state-space matrices \([A,B,C,D]\) of the plant with the usual partitions:

\[
B = \begin{pmatrix} B_1 & B_2 \end{pmatrix}, \quad C = \begin{pmatrix} C_1 \end{pmatrix}, \quad D = \begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix}
\]

These partitions are implicitly given in \( r \): \( r(1) \) and \( r(2) \) are the dimensions of \( D_{22} \) (rows x columns).

With \text{option}='t', \text{gamitg} traces each bisection step, i.e., displays the lower and upper bounds and the current test point.

**SEE ALSO:** ccontrg 374, h_inf 381

**AUTHOR:** P. Gahinet

**2.2.14**  
**gcare**  
control Riccati equation

**CALLING SEQUENCE:**

\[ [X,F]=\text{gcare}(Sl) \]

**PARAMETERS:**

- \( Sl \): linear system (syslin list)
- \( X \): symmetric matrix
- \( F \): real matrix

**DESCRIPTION:**

Generalized Control Algebraic Riccati Equation (GCARE). \( X \) = solution, \( F \) = gain.

The GCARE for \( Sl=[A,B,C,D] \) is:

\[
(A-B*Si*D'*C)'*X+X*(A-B*Si*D'*C)-X*B*Si*B'*X+C'*Ri*C=0
\]

where \( S=(\text{eye()})+D'*D), Si=\text{inv}(S), Ri=\text{inv}(R) \) and \( F=-Si*(D'*C+B'*X) \)

is such that \( A+B*F \) is stable.

**SEE ALSO:** gfare 379

**2.2.15**  
**gfare**  
filter Riccati equation

**CALLING SEQUENCE:**

\[ [Z,H]=\text{gfare}(Sl) \]

**PARAMETERS:**

- \( Sl \): linear system (syslin list)
- \( Z \): symmetric matrix
- \( H \): real matrix

**DESCRIPTION:**

Generalized Filter Algebraic Riccati Equation (GFARE). \( Z \) = solution, \( H \) = gain.

The GFARE for \( Sl=[A,B,C,D] \) is:

\[
(A-B*D'*Ri*C)*Z+Z*(A-B*D'*Ri*C)'-Z*C'*Ri*C*Z+B*Si*B'=0
\]

where \( S=(\text{eye()})+D'*D), Si=\text{inv}(S), Ri=\text{inv}(R) \) and \( H=-(B*D'+Z*C')*Ri \)

is such that \( A+H*C \) is stable.

**SEE ALSO:** gcare 379
CALLING SEQUENCE:

Gt=gtild(G)
Gt=gtild(G,flag)

PARAMETERS:

G : either a polynomial or a linear system (syslin list) or a rational matrix
Gt : same as G
flag : character string: either 'c' or 'd' (optional parameter).

DESCRIPTION:

If G is a polynomial matrix (or a polynomial), Gt=gtild(G,'c') returns the polynomial matrix
Gt(s)=G(-s)'

If G is a polynomial matrix (or a polynomial), Gt=gtild(G,'d') returns the polynomial matrix
Gt=G(1/z)*z^n where n is the maximum degree of G.

For continuous-time systems represented in state-space by a syslin list, Gt = gtild(G,'c') returns a
state-space representation of G(-s) i.e the ABCD matrices of Gt are A', -C', B', D'. If G is
improper (D= D(s)) the D matrix of Gt is D(-s)'.

For discrete-time systems represented in state-space by a syslin list, Gt = gtild(G,'d') returns a
state-space representation of G(-1/z)' i.e the (possibly improper) state-space representation of -z*C*inv(z*A-B)*C + D(1/z).

For rational matrices, Gt = gtild(G,'c') returns the rational matrix Gt(s)=G(-s) and Gt =

gtild(G,'d') returns the rational matrix Gt(z)= G(1/z)'.

The parameter flag is necessary when gtild is called with a polynomial argument.

EXAMPLE:

//Continuous time
s=poly(0,'s');G=[s,s^3;2+s^3,s^2-5]
Gt=gtild(G,'c')
Gt=horner(G,-s)' //continuous-time interpretation
Gt=gtild(G,'d');
Gt=horner(G,1/s)'*s^3 //discrete-time interpretation

G=ssrand(2,2,3);Gt=gtild(G); //State-space (G is cont. time by default)
clean((horner(ss2tf(G),-s))'-ss2tf(Gt)) //Check
// Discrete-time
z=poly(0,'z');
Gss=ssrand(2,2,3);Gss('dt')='d'; //discrete-time
Gss(5)=[1,2;0,1]; //With a constant D matrix
G=ss2tf(Gss);Gt1=horner(G,1/z)';
Gt=gtild(Gss);
Gt2=clean(ss2tf(Gt)); clean(Gt1-Gt2) //Check
//Improper systems
z=poly(0,'z');
Gss=ssrand(2,2,3);Gss(7)='d'; //discrete-time
Gss(5)=[z,z^2;1+z,3]; //D(z) is polynomial
G=ss2tf(Gss);Gt1=horner(G,1/z)'; //Calculation in transfer form
Gt=gtild(Gss); //in state-space
Gt2=clean(ss2tf(Gt));clean(Gt1-Gt2) //Check

SEE ALSO: syslin 224, horner 490, factors 488
2.2.17  **h2norm**  

**CALLING SEQUENCE:**

\[ [n]=h2norm(Sl [,tol]) \]

**PARAMETERS:**

- \( Sl \): linear system (syslin list)
- \( n \): real scalar

**DESCRIPTION:**

produces the H2 norm of a linear continuous time system \( Sl \).
(For \( Sl \) in state-space form \( h2norm \) uses the observability gramian and for \( Sl \) in transfer form \( h2norm \) uses a residue method)

2.2.18  **h_cl**  

**CALLING SEQUENCE:**

\[ [Acl]=h_cl(P,r,K) \]
\[ [Acl]=h_cl(P22,K) \]

**PARAMETERS:**

- \( P, P22 \): linear system (syslin list), augmented plant or nominal plant respectively
- \( r \): 1x2 row vector, dimensions of 2,2 part of \( P \) \((r=[\text{rows},\text{cols}]=\text{size}(P22))\)
- \( K \): linear system (syslin list), controller
- \( Acl \): real square matrix

**DESCRIPTION:**

Given the standard plant \( P \) (with \( r=\text{size}(P22) \)) and the controller \( K \), this function returns the closed loop matrix \( Acl \).
The poles of \( Acl \) must be stable for the internal stability of the closed loop system.
\( Acl \) is the \( A \)-matrix of the linear system \([I -P22;-K I]^{-1} \) i.e. the \( A \)-matrix of \( \text{lft}(P,r,K) \)

**SEE ALSO:**  \text{lft} 384

**AUTHOR:** F. D.

2.2.19  **h_inf**  

**CALLING SEQUENCE:**

\[ [Sk,ro]=h_inf(P,r,romin,romax,nmax) \]
\[ [Sk,rk,ro]=h_inf(P,r,romin,romax,nmax) \]

**PARAMETERS:**

- \( P \): syslin list: continuous-time linear system (“augmented” plant given in state-space form or in transfer form)
- \( r \): size of the \( P22 \) plant i.e. 2-vector \([\text{#outputs},\text{#inputs}]\)
- \( \text{romin,romax} \): a priori bounds on \( ro \) with \( ro=1/gama^2; (\text{romin}=0 \text{ usually}) \)
- \( \text{nmax} \): integer, maximum number of iterations in the gama-iteration.
**DESCRIPTION:**

`h_inf` computes H-infinity optimal controller for the continuous-time plant \( P \).
The partition of \( P \) into four sub-plants is given through the 2-vector \( r \) which is the size of the 22 part of \( P \).
\( P \) is given in state-space e.g. \( P = \text{syslin}(\text{'c'},A,B,C,D) \) with \( A,B,C,D \) = constant matrices or \( P = \text{syslin}(\text{'c'},H) \) with \( H \) a transfer matrix.

\[
[Sk,ro]=\text{H}_\inf(P,r,\text{romin},\text{romax},\text{nmax})
\]
returns \( ro \) in \([\text{romin},\text{romax}]\) and the central controller \( Sk \) in the same representation as \( P \).

(All calculations are made in state-space, i.e. conversion to state-space is done by the function, if necessary).

Invoked with three LHS parameters, \( [Sk,rk,ro]=\text{H}_\inf(P,r,\text{romin},\text{romax},\text{nmax}) \) returns \( ro \) and the Parameterization of all stabilizing controllers:
- a stabilizing controller \( K \) is obtained by \( K=\text{lft}(Sk,r:\text{PHI}) \) where \( \text{PHI} \) is a linear system with dimensions \( r' \) and satisfy:
  \[ H_{\text{norm}}(\text{PHI}) < \gamma \]
  \( rk (=r) \) is the size of the \( Sk_{22} \) block and \( ro = 1/\gamma^2 \) after \( nmax \) iterations.
Algorithm is adapted from Safonov-Limebeer. Note that \( P \) is assumed to be a continuous-time plant.

**SEE ALSO:** gamitg 378, ccontrg 374, leqr 383

**AUTHOR:** F.D. (1990)

---

### 2.2.20 h_inf_st ............................. static H_infinity problem

**CALLING SEQUENCE:**

\[
[Kopt,gamaopt]=\text{h_inf_stat}(D,r)
\]

**PARAMETERS:**

- \( D \) : real matrix
- \( r \) : 1x2 vector
- \( Kopt \) : matrix

**DESCRIPTION:**

computes a matrix \( Kopt \) such that largest singular value of:
\[ \text{lft}(D,r,K)=D_{11}+D_{12} \cdot K \cdot \text{inv}(I-D_{22} \cdot K) \cdot D_{21} \]
is minimal (Static H-infinity four blocks problem).
\( D \) is partitioned as \( D=[D_{11} D_{12}; D_{21} D_{22}] \) where \( \text{size}(D_{22})=r=[r1 \ r2] \)

**AUTHOR:** F.D.

---

### 2.2.21 h_norm ........................................... H-infinity norm

**CALLING SEQUENCE:**

\[
[hinfnorm [,frequency]]=\text{h_norm}(sl [,rerr])
\]

**PARAMETERS:**

- \( sl \) : the state space system (syslin list)
- \( rerr \) : max. relative error, default value 1e-8
- \( hinfnorm \) : the infinity norm of \( Sl \)
- \( frequency \) : frequency at which maximum is achieved

**DESCRIPTION:**

produces the infinity norm of a state-space system (the maximum over all frequencies of the maximum singular value).

**SEE ALSO:** linfn 385, linf 385, svplot 365

Scilab Group

April 1993
2.2.22 hankelsv

**Hankel singular values**

**CALLING SEQUENCE:**

\[
[nk2,W]=\text{hankelsv}(sl [,tol])
\]

**PARAMETERS:**

- \(sl\) : syslin list representing the linear system (state-space).
- \(tol\) : tolerance parameter for detecting imaginary axis modes (default value is \(1000*\text{eps}\)).

**DESCRIPTION:**

Returns \(nk2\), the squared Hankel singular values of \(sl\) and \(W = P*Q = \text{controllability gramian times observability gramian}\). \(nk2\) is the vector of eigenvalues of \(W\).

**EXAMPLE:**

\[
A=\text{diag}([-1,-2,-3]);
sl=\text{syslin}('c',A,\text{rand}(3,2),\text{rand}(2,3));[nk2,W]=\text{hankelsv}(sl)
\]

\[
[Q,M]=\text{pbig}(W,nk2(2)-\text{eps},'c');
slr=\text{projsl}(sl,Q,M);\text{hankelsv}(slr)
\]

**SEE ALSO:** balreal 322, equil 334, equil 335

2.2.23 lcf

**normalized coprime factorization**

**CALLING SEQUENCE:**

\[
[N,M]=\text{lcf}(sl)
\]

**PARAMETERS:**

- \(sl\) : linear system given in state space or transfer function (syslin list)
- \(N, M\) : two linear systems (syslin list)

**DESCRIPTION:**

Computes normalized coprime factorization of the linear dynamic system \(sl\).

\(sl = M^{-1} N\)

**AUTHOR:** F. D.

2.2.24 leqr

**H-infinity LQ gain (full state)**

**CALLING SEQUENCE:**

\[
[K,X,err]=\text{leqr}(P12,Vx)
\]

**PARAMETERS:**

- \(P12\) : syslin list
- \(Vx\) : symmetric nonnegative matrix (should be small enough)
- \(K, X\) : two real matrices
- \(err\) : a real number (l1 norm of LHS of Riccati equation)
DESCRIPTION:
leqr computes the linear suboptimal H-infinity LQ full-state gain for the plant \( P_{12} = [A, B_2, C_1, D_{12}] \) in continuous or discrete time.

\( P_{12} \) is a \( \text{syslin} \) list (e.g. \( P_{12} = \text{syslin('c',A,B_2,C_1,D_{12})} \)).

\[
\begin{bmatrix}
C_1' \\
[ ] \\
D_{12}'
\end{bmatrix}
\begin{bmatrix}
Q & S \\
[ ] & [ ]
\end{bmatrix}
\begin{bmatrix}
C_1 \\
D_{12}
\end{bmatrix}
= \begin{bmatrix}
[ ] & [ ]
\end{bmatrix}
\begin{bmatrix}
S' & R
\end{bmatrix}
\]

\( V_x \) is related to the variance matrix of the noise \( w \) perturbing \( x \); (usually \( V_x = \text{gama}^{-2}B_1B_1' \)).
The gain \( K \) is such that \( A + B_2K \) is stable.
\( X \) is the stabilizing solution of the Riccati equation.
For a continuous plant:
\[
(A-B_2\text{inv}(R)*S')'X+X*(A-B_2\text{inv}(R)*S')-X*(B_2\text{inv}(R)*B_2'-V_x)*X+Q-S\text{inv}(R)*S'=0
\]
\( K = -\text{inv}(R)*(B_2'*X+S) \)

For a discrete time plant:
\[
X-(Abar'\text{inv}((\text{inv}(X)+B_2\text{inv}(R)*B_2'-V_x))*Abar+Qbar=0
\]
\( K = -\text{inv}(R)*(B_2'\text{inv}(\text{inv}(X)+B_2\text{inv}(R)*B_2'-V_x)*Abar+S') \)

with \( Abar = A-B_2'\text{inv}(R)*S' \) and \( Qbar = Q-S\text{inv}(R)*S' \)
The 3-blocks matrix pencils associated with these Riccati equations are:

\[
\begin{array}{ccc}
\text{discrete} & \text{continuous} \\
| \text{I} & \text{-}V_x & 0 | & \text{I} & 0 & 0 | & \text{I} & 0 & 0 | & \text{A} & \text{V}_x & \text{B}_2 |\\
| 0 & \text{A'} & 0 | & \text{-}Q & \text{I} & -S | & 0 & \text{I} & 0 | & \text{-}Q & \text{-}A' & \text{-}S |\\
| 0 & \text{B}_2' & 0 | & \text{S'} & 0 & \text{R} | & 0 & 0 & 0 | & \text{S'} & \text{-}B_2' & \text{R} |
\end{array}
\]

SEE ALSO: leqr

AUTHOR: F.D.

2.2.25 lft linear fractional transformation

CALLING SEQUENCE:

\[
[P_1]=\text{LFT}(P,K) \\
[P_1]=\text{LFT}(P,r,K) \\
[P_1,r_1]=\text{LFT}(P,r,P#,r#)
\]

PARAMETERS:

\( P \): linear system (\text{syslin} list), the "augmented" plant, implicitly partitioned into four blocks (two input ports and two output ports).
\( K \): linear system (\text{syslin} list), the controller (possibly an ordinary gain).
\( r \): 1x2 row vector, dimension of \( P_{22} \)
\( P# \): linear system (\text{syslin} list), implicitly partitioned into four blocks (two input ports and two output ports).
\( r# \): 1x2 row vector, dimension of \( P_{#22} \)
DESCRIPTION:
Linear fractional transform between two standard plants \( P \) and \( P^\# \) in state space form or in transfer form (syslin lists).

\[ r = \text{size}(P22) \quad r^\# = \text{size}(P22^\#) \]

\( \text{LFT}(P, r, K) \) is the linear fractional transform between \( P \) and a controller \( K \) (\( K \) may be a gain or a controller in state space form or in transfer form);

\( \text{LFT}(P, K) \) is \( \text{LFT}(P, r, K) \) with \( r = \text{size of } K \) transpose;

\[ P_1 = P_{11} + P_{12} \cdot K \cdot (I - P_{22} \cdot K)^{-1} \cdot P_{21} \]

\( [P_1, r_1] = \text{LFT}(P, r, P^\#, r^\#) \) returns the generalized (2 ports) lft of \( P \) and \( P^\# \).

\( P_1 \) is the pair two-port interconnected plant and the partition of \( P_1 \) into 4 blocks in given by \( r_1 \) which is the dimension of the 22 block of \( P_1 \).

\( P \) and \( R \) can be PSSDs i.e. may admit a polynomial \( D \) matrix.

EXAMPLE:

\[
\begin{align*}
s &= \text{poly}(0, 's'); \\
P &= \begin{bmatrix}
1/s, & 1/(s+1); 1/(s+2), & 2/s
\end{bmatrix}; \\
K &= 1/(s-1); \\
\text{lft}(P, K) \\
\text{lft}(P, [1, 1], K) \\
\text{lft}(P(1, 1) + P(1, 2) \cdot K \cdot \text{inv}(1 - P(2, 2) \cdot K) \cdot P(2, 1)) \quad \text{//Numerically dangerous!}
\end{align*}
\]

\[
\text{ss2tf}(\text{lft}(\text{tf2ss}(P), \text{tf2ss}(K)))
\]

\[
\text{lft}(P, -1)
\]

\[
f = [0, 0; 0, 1]; w = P ./ f; w(1,1) \\
\text{//Improper plant (PID control)}
\]

\[
W = [1, 1; 1, 1/(s^2 + 0.1 \cdot s)]; K = 1 + 1/s + s
\]

\[
\text{lft}(W, [1, 1], K); \quad \text{ss2tf}(\text{lft}(\text{tf2ss}(W), [1, 1], \text{tf2ss}(K)))
\]

SEE ALSO: sensi 390, augment 373, feedback 335, sysdiag 224

\[ 2.2.26 \quad \text{linf \hspace{1cm} \text{infinity norm}} \]

CALLING SEQUENCE:

\[
\text{linf}(g [, \text{eps}], [\text{tol}])
\]

PARAMETERS:

\( g \) : is a syslin linear system.
\( \text{eps} \) : is error tolerance on \( n \).
\( \text{tol} \) : threshold for imaginary axis poles.

DESCRIPTION:

returns the L\(_\infty\) norm of \( g \).

\[
\begin{align*}
n &= \text{sup} \left[ \text{sigmax}(g(jw)) \right] \\
\end{align*}
\]

\( \text{w} \)

(sigmax largest singular value).

SEE ALSO: h\_norm 382, linfn 385

\[ 2.2.27 \quad \text{linf \hspace{1cm} \text{infinity norm}} \]

CALLING SEQUENCE:

\[
[x, \text{freq}] = \text{linfn}(G, \text{PREC}, \text{RELTOL}, \text{options});
\]

Scilab Group April 1993 385
PARAMETERS:
G : is a syslin list
PREC : desired relative accuracy on the norm
RELTOL : relative threshold to decide when an eigenvalue can be considered on the imaginary axis.
options : available options are 'trace' or 'cond'
x is the computed norm.
freq : vector

DESCRIPTION:
Computes the Linf (or Hinf) norm of G. This norm is well-defined as soon as the realization G= (A, B, C, D) has no imaginary eigenvalue which is both controllable and observable.
freq is a list of the frequencies for which ||G|| is attained, i.e., such that \|G (j \omega)\| = ||G||.
If -1 is in the list, the norm is attained at infinity.
If -2 is in the list, G is all-pass in some direction so that \|G (j \omega)\| = ||G|| for all frequencies omega.
The algorithm follows the paper by G. Robel (AC-34 pp. 882-884, 1989). The case D=0 is not treated separately due to superior accuracy of the general method when (A, B, C) is nearly non minimal.
The 'trace' option traces each bisection step, i.e., displays the lower and upper bounds and the current test point.
The 'cond' option estimates a confidence index on the computed value and issues a warning if computations are ill-conditioned.
In the general case (A neither stable nor anti-stable), no upper bound is prespecified.
If by contrast A is stable or anti stable, lower and upper bounds are computed using the associated Lyapunov solutions.
SEE ALSO: h_norm 382

AUTHOR: P. Gahinet

2.2.28 lqg_ltr .......................... LQG with loop transform recovery

CALLING SEQUENCE:
[kf, kc]=lqg_ltr(sl,mu,ro)

PARAMETERS:
sl : linear system in state-space form (syslin list)
mu, ro : real positive numbers chosen "small enough"

DESCRIPTION:
returns the Kalman gains for:
\[ x = A x + B u + L w_1 \]  
\[ y = C x + \mu I w_2 \]  
\[ z = H x \]
Cost function:
\[ J_{lqg} = E\left( \int_0^\infty [z(t)^T z(t) + \rho^2 u(t)^T u(t)] dt \right) \]
The lqg_ltr approach looks for L, \mu, H, ro such that: J(lqg) = J(freq) where
\[ J_{freq} = \int_0^\infty tr[SWW^*S^*] + tr[TT^*] dw \]
and

Scilab Group April 1993 386
S = (I + G*K)^(-1)
T = G*K*(I+G*K)^(-1)

SEE ALSO: syslin 224

2.2.29 macglov ____________________________ Mac Farlane Glover problem

CALLING SEQUENCE :

[P,r]=macglov(Sl)

PARAMETERS :

Sl : linear system (syslin list)
P : linear system (syslin list), "augmented" plant
r : 1x2 vector, dimension of P22

DESCRIPTION :

[P,r]=macglov(Sl) returns the standard plant P for the Glover-McFarlane problem.
For this problem ro_optimal = 1-\text{hankel norm}( [N,M] ) with [N,M]=\text{lcf}(sl) (Normalized coprime factorization) i.e.
gama_optimal = 1/sqrt(ro_optimal)

AUTHOR : F. D.

2.2.30 nehari ________________________________ Nehari approximant

CALLING SEQUENCE :

[x]=nehari(R [,tol])

PARAMETERS :

R : linear system (syslin list)
x : linear system (syslin list)
tol : optional threshold

DESCRIPTION :

[x]=nehari(R [,tol]) returns the Nehari approximant of R.
R = linear system in state-space representation (syslin list).
R is strictly proper and - R^\sim is stable (i.e. R is anti stable).

\| R - X \|oo = min \| R - Y \|oo 
Y in Hoo

2.2.31 parrot ________________________________ Parrot’s problem

CALLING SEQUENCE :

K=parrot(D,r)

PARAMETERS :

D,K : matrices

Scilab Group        April 1993
 DESCRIPTION:
Given a matrix D partitioned as \([D_{11} D_{12}; D_{21} D_{22}]\) where size(D22)=r=[r1,r2] compute a matrix K such that largest singular value of \([D_{11} D_{12}; D_{21} D_{22}+K]\) is minimal (Parrot’s problem)

SEE ALSO: h_inf_st 382

2.2.32  ric_desc  --------------------------------- Riccati equation

CALLING SEQUENCE:

\[
X=ric\_desc(H [,E]) \\
[X1,X2,zero]=ric\_desc(H [,E])
\]

PARAMETERS:

H,E : real square matrices
X1,X2 : real square matrices
zero : real number

DESCRIPTION:
Riccati solver with hamiltonian matrices as inputs.
In the continuous time case calling sequence is (one input):
ric_descr(H)
Riccati equation is:

\[
(Ec)\quad A'*X + X*A + X*R*X -Q = 0.
\]

Defining the hamiltonian matrix \(H\) by:

\[
H = \begin{bmatrix}
A & R \\
Q & -A'
\end{bmatrix}
\]

with the calling sequence \([X1,X2,zero]=ric\_desc(H)\), the solution \(X\) is given by \(X=X1/X2\).
zero = \(L1\) norm of rhs of (Ec)
The solution \(X\) is also given by \(X=riccati(A,Q,R,’c’))\)
In the discrete-time case calling sequence is (two inputs):
ric_descr(H,E)
The Riccati equation is:

\[
(Ed)\quad A'*X*A-(A'*X*B*(R+B'*X*B)^-1)*(B'*X*A)+C-X = 0.
\]

Defining \(G=B/R*B'\) and the hamiltonian pencil \((E,H)\) by:

\[
E=[eye(n,n),G; 0*ones(n,n),A'] \quad H=[A, 0*ones(n,n); -C, eye(n,n)]
\]

with the calling sequence \([X1,X2,err]=ric\_desc(H,E)\), the solution \(X\) is given by \(X=X1/X2\).
zero= \(L1\) norm of rhs of (Ed)
The solution \(X\) is also given by \(X=riccati(A,G,C,’d’))\) with \(G=B/R*B'\)

SEE ALSO:  riccati 389

Scilab Group  
April 1993 388
### 2.2.33 riccati

**Riccati equation**

**CALLING SEQUENCE:**

\[
X = \text{riccati}(A, B, C, \text{dom}, [\text{typ}])
\]

\[
[X1, X2] = \text{riccati}(A, B, C, \text{dom}, [\text{typ}])
\]

**PARAMETERS:**

- \(A, B, C\): real matrices \(nxn\), \(B\) and \(C\) symmetric.
- \(\text{dom}\): ‘c’ or ‘d’ for the time domain (continuous or discrete).
- \(\text{typ}\): string: ‘eigen’ for block diagonalization or ‘schur’ for Schur method.
- \(X1, X2, X\): square real matrices (\(X2\) invertible), \(X\) symmetric.

**DESCRIPTION:**

\(X = \text{riccati}(A, B, C, \text{dom}, [\text{typ}])\) solves the Riccati equation:

\[
A'X + XA - XBX + C = 0
\]

in continuous time case, or:

\[
A'X + XA - \left(\frac{A' \times X \times B1}{B2 + B1' \times X \times B1}\right) \times (B1' \times X \times A) + C - X
\]

with \(B = B1 / B2 * B1'\) in the discrete time case. If called with two output arguments, \text{riccati} returns \(X1, X2\) such that \(X = X1 / X2\).

**SEE ALSO:** \text{ric_desc} 388

### 2.2.34 rowinout

**inner-outer factorization**

**CALLING SEQUENCE:**

\[
[\text{Inn}, X, \text{Gbar}] = \text{rowinout}(G)
\]

**PARAMETERS:**

- \(G\): linear system (\text{syslin} list) \([A, B, C, D]\)
- \(\text{Inn}\): inner factor (\text{syslin} list)
- \(\text{Gbar}\): outer factor (\text{syslin} list)
- \(X\): row-compressor of \(G\) (\text{syslin} list)

**DESCRIPTION:**

Inner-outer factorization (and row compression) of \((l \times p)\) \(G = [A, B, C, D]\) with \(l \geq p\).

\(G\) is assumed to be tall \((l \geq p)\) without zero on the imaginary axis and with a \(D\) matrix which is full column rank.

\(G\) must also be stable for having \(\text{Gbar}\) stable.

\(G\) admits the following inner-outer factorization:

\[
G = [\text{Inn}] \begin{bmatrix} Gbar \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix}
\]

where \(\text{Inn}\) is square and inner (all pass and stable) and \(\text{Gbar}\) square and outer i.e: \(G\bar{\text{bar}}\) is square bi-proper and bi-stable (\(G\text{bar inverse is also proper and stable}\)).

Note that:

\[
\begin{bmatrix} \text{Gbar} \end{bmatrix} XG = \begin{bmatrix} - \\ 0 \end{bmatrix}
\]

Scilab Group April 1993 389
is a row compression of $G$ where $X = \text{Inn}$ inverse is all-pass i.e:

$$
T
$$

$X (-s) X(s) = \text{Identity}$

(for the continuous time case).

SEE ALSO: syslin 224, colinout 375

### 2.2.35 sensi

**sensitivity functions**

**CALLING SEQUENCE:**

\[
[\text{Se}, \text{Re}, \text{Te}] = \text{sensi}(G, K) \\
[\text{Si}, \text{Ri}, \text{Ti}] = \text{sensi}(G, K, \text{flag})
\]

**PARAMETERS:**

- $G$: standard plant (syslin list)
- $K$: compensator (syslin list)
- flag: character string ‘o’ (default value) or ‘i’
- $\text{Se}$: output sensitivity function $(I+G*K)^{-1}$
- $\text{Re}$: $K*\text{Se}$
- $\text{Te}$: $G*K*\text{Se}$ (output complementary sensitivity function)

**DESCRIPTION:**

sensi computes sensitivity functions. If $G$ and $K$ are given in state-space form, the systems returned are generically minimal. Calculation is made by lft, e.g., $\text{Se}$ can be given by the commands $P = \text{augment}(G, 'S'), \text{Se} = \text{lft}(P, K)$. If flag = ‘i’, $[\text{Si}, \text{Ri}, \text{Ti}] = \text{sensi}(G, K, 'i')$ returns the input sensitivity functions.

\[
[\text{Se}; \text{Re}; \text{Te}] = [\text{inv}(\text{eye()}+G*K); K*\text{inv}(\text{eye()}+G*K); G*K*\text{inv}(\text{eye()}+G*K)]; \\
[\text{Si}; \text{Ri}; \text{Ti}] = [\text{inv}(\text{eye()}+K*G); G*\text{inv}(\text{eye()}+K*G); K*G*\text{inv}(\text{eye()}+K*G)];
\]

**EXAMPLE:**

\[
G = \text{ssrand}(1,1,3); K = \text{ssrand}(1,1,3); \\
[\text{Se}, \text{Re}, \text{Te}] = \text{sensi}(G, K); \\
\text{Sel} = \text{inv}(\text{eye()}+G*K); \quad //\text{Other way to compute} \\
\text{ss2tf}(\text{Se}) \quad //\text{Se seen in transfer form} \\
\text{ss2tf}(\text{Sel}) \\
\text{ss2tf}(\text{Te}) \\
\text{ss2tf}((G*K*\text{Sel}) \\
[\text{Si}, \text{Ri}, \text{Ti}] = \text{sensi}(G, K, 'i'); \\
w1 = [\text{ss2tf}(\text{Si}); \text{ss2tf}(\text{Ri}); \text{ss2tf}(\text{Ti})] \\
w2 = [\text{ss2tf}(\text{inv}(\text{eye()}+K*G)); \text{ss2tf}(G*\text{inv}(\text{eye()}+K*G)); \text{ss2tf}(K*G*\text{inv}(\text{eye()}+K*G))]; \\
clean(w1-w2)
\]

SEE ALSO: augment 373, lft 384, h_cl 381

### 2.2.36 tf2des

**transfer function to descriptor**

**CALLING SEQUENCE:**

\[
S = \text{tf2des}(G) \\
S = \text{tf2des}(G, \text{flag})
\]

Scilab Group April 1993
PARAMETERS:

G : linear system (syslin list) with possibly polynomial D matrix
flag : character string "withD"
S : list

DESCRIPTION:
Transfer function to descriptor form: S=list (’d’, A, B, C, D, E)

\[
E \cdot \dot{x} = A \cdot x + B \cdot u
\]
\[
y = C \cdot x + D \cdot u
\]

Note that \( D=0 \) if the optional parameter flag="withD" is not given. Otherwise a maximal rank D matrix is returned in the fifth entry of the list S

EXAMPLE:

\[
s = \text{poly}(0,’s’);
G = [1/(s-1), s; 1, 2/s^3];
S1 = \text{tf2des}(G); \text{des2tf}(S1)
S2 = \text{tf2des}(G, "withD"); \text{des2tf}(S2)
\]

SEE ALSO: pol2des 494, tf2ss 367, ss2des 361, des2tf 332
2.3 Tools for dynamical systems
2.3.1 arma

DESCRIPTION:
Armax processes can be coded with Scilab tlist of type ‘ar’. armac is used to build Armax scilab object. An ‘ar’ tlist contains the fields [‘a’, ‘b’, ‘d’, ‘ny’, ‘nu’, ‘sig’].

armac: this function creates a Scilab tlist which code an Armax process $A(z^{-1})y = B(z^{-1})u + D(z^{-1})\text{sig}\cdot e(t)$

```matlab
-->ar=armac([1,2],[3,4],1,1,1,sig);
```

```matlab
-->ar('a')
ans =
   !1 . 2 . !

-->ar('sig')
ans =
   1.
```

armap (ar [, out]): Display the armax equation associated with ar

armapd (ar [, out]): Display the armax equation associated with ar using polynomial matrix display.

$[A,B,D]=arma2p(ar)$: extract polynomial matrices from ar representation

 arma: is used to identify the coefficients of a n-dimensional ARX process $A(z^{-1})y = B(z^{-1})u + \text{sig}\cdot e(t)$

 armax1: armax1 is used to identify the coefficients of a 1-dimensional ARX process $A(z^{-1})y = B(z^{-1})u + D(z^{-1})\text{sig}\cdot e(t)$

 arsimul: armax trajectory simulation.

 arspec: Spectral power estimation of armax processes. Test of mese and arsimul

 exar1: An Example of ARMAX identification (K.J. Astrom) The armax process is described by: $a=[1,-2.851,2.717,-0.865]$ $b=[0,1,1,1]$ $d=[1,0.7,0.2]$ $\text{exar2: ARMAX example (K.J. Astrom). A simulation of a bi dimensional version of the example of exar1.}$ $\text{exar3: Spectral power estimation of arma processes from Sawaragi et all where a value of m=18 is used. Test of mese and arsimul}$

 gbruit: noise generation

 narsimul: armax simulation (using rtitr)

 odedi: Simple tests of ode and arsimul. Tests the option 'discret' of ode

 prbs_a: pseudo random binary sequences generation

 reglin: Linear regression

AUTHOR: J.P.C

2.3.2 arma2p

CALLING SEQUENCE:

$[A,B,D]=arma2p(ar)$

PARAMETERS:

A, B, D: three polynomial matrices
armax Scilab Function

DESCRIPTION:
this function extract polynomial matrices \((A, B, D)\) from an armax description.

\[
A(z^{-1})y = B(z^{-1})u + D(z^{-1})\sigma e(t)
\]

EXAMPLE:
\[
\begin{align*}
a &= [1, -2.851, 2.717, -0.865] \cdot \text{eye}(2, 2) \\
b &= [0, 1, 1, 1] \cdot [1; 1] \\
d &= [1, 0.7, 0.2] \cdot \text{eye}(2, 2) \\
sig &= \text{eye}(2, 2)
\end{align*}
\]
\[
\text{ar} = \text{armac}(a, b, d, 2, 1, \text{sig})
\]
\[
// extract polynomial matrices from ar representation
[A, B, D] = \text{arma2p}(\text{ar});
\]

SEE ALSO: arma 393, armax 395, armax1 396, arsimul 396, armac 394

2.3.3 armac ______________________ Scilab description of an armax process

CALLING SEQUENCE:

\[
[\text{ar}] = \text{armac}(a, b, d, n_y, n_u, \sigma)
\]

PARAMETERS:
\[
\begin{align*}
a &= [\text{Id}, a_1, \ldots, a_r] & \text{is a matrix of size } (n_y, r \times n_y) \\
b &= [b_0, \ldots, b_s] & \text{is a matrix of size } (n_y, (s+1) \times n_u) \\
d &= [\text{Id}, d_1, \ldots, d_p] & \text{is a matrix of size } (n_y, p \times n_y) \\
n_y & \text{ : dimension of the output } y \\
n_u & \text{ : dimension of the output } u \\
\sigma & \text{ : a matrix of size } (n_y, n_y)
\end{align*}
\]

DESCRIPTION:
This function creates a description as a tlist of an ARMAX process

\[
A(z^{-1})y = B(z^{-1})u + D(z^{-1})\sigma e(t)
\]

\[
\text{ar} \text{ is defined by}
\]

\[
\text{ar} = \text{tlist}([''a'', 'a', 'b', 'd', 'n_u', 'n_y', 'sig'], a, b, d, n_u, n_y, \sigma);
\]

and thus the coefficients of \text{ar} can be retrieved by e.g. ar ("a") .

EXAMPLE:
\[
\begin{align*}
a &= [1, -2.851, 2.717, -0.865] \cdot \text{eye}(2, 2) \\
b &= [0, 1, 1, 1] \cdot [1; 1] \\
d &= [1, 0.7, 0.2] \cdot \text{eye}(2, 2) \\
sig &= \text{eye}(2, 2)
\end{align*}
\]
\[
\text{ar} = \text{armac}(a, b, d, 2, 1, \text{sig})
\]
\[
// extract polynomial matrices from ar representation
[A, B, D] = \text{arma2p} (\text{ar});
\]

SEE ALSO: arma 393, armax 395, armax1 396, arsimul 396, armac 394, tlist.
2.3.4 \texttt{armax} \quad \texttt{Scilab Function}

\textbf{CALLING SEQUENCE :}

\[
[\text{arc,la,lb,sig,resid}]=\text{armax}(r,s,y,u,[\text{bof,prf}])
\]

\textbf{PARAMETERS :}

\begin{itemize}
  \item \textit{y} : output process \textit{y}(ny,n); ( \textit{ny} : dimension of \textit{y}, \textit{n} : sample size)
  \item \textit{u} : input process \textit{u}(nu,n); ( \textit{nu} : dimension of \textit{u}, \textit{n} : sample size)
  \item \textit{r} and \textit{s} : auto-regression orders \textit{r} \geq 0 et \textit{s} \geq -1
  \item \textit{bof} : optional parameter. Its default value is 0 and it means that the coefficient \textit{b0} must be identified. if \textit{bof}=1 the \textit{b0} is supposed to be zero and is not identified
  \item \textit{prf} : optional parameter for display control. If \textit{prf}=1, the default value, a display of the identified Arma is given.
  \item \textit{arc} : a Scilab arma object (see \texttt{armac})
  \item \textit{la} : is the list(\textit{a,a+eta,a-eta}) ( \textit{la} = \textit{a} in dimension 1); where \textit{eta} is the estimated standard deviation. \textit{a} = \text{Id,a1,a2,...,ar} where each \textit{ai} is a matrix of size (\textit{ny},\textit{ny})
  \item \textit{lb} : is the list(b,b+etb,b-etb) ( \textit{lb} = \textit{b} in dimension 1); where \textit{etb} is the estimated standard deviation. \textit{b} = \text{b0,.....,bs} where each \textit{bi} is a matrix of size (\textit{nu},\textit{nu})
  \item \textit{sig} : is the estimated standard deviation of the noise and \textit{resid} = \text{[ sig*\textit{e}(t0),....]} (\textit{CU})
\end{itemize}

\textbf{DESCRIPTION :}

\texttt{armax} is used to identify the coefficients of a n-dimensional ARX process

\[
A(z^-1)y = B(z^-1)u + \text{sig*}e(t)
\]

where \textit{e}(\textit{t}) is a \textit{n}-dimensional white noise with variance 1. \textit{sig} an \textit{nxn} matrix and \textit{A}(\textit{z}) and \textit{B}(\textit{z}):

\[
A(z) = 1+a1*z+...+a_r*z^-r; \quad ( r=0 \Rightarrow A(z)=1)
\]

\[
B(z) = b0+b1*z+...+b_s z^-s \quad ( s=-1 \Rightarrow B(z)=0)
\]

for the method see Eykhoff in trends and progress in system identification, page 96. with \textit{z}(\textit{t})=[\textit{y}(\textit{t}-1),...\textit{y}(\textit{t}-\textit{r}),\textit{u}(\textit{t}),...,\textit{u}(\textit{t}-\textit{s})] and \textit{coef}=[-\textit{a1},...,\textit{ar},\textit{bo},...,\textit{bs}] we can write \textit{y}(\textit{t})=\textit{coef}*z(\textit{t}) + \text{sig*}e(\textit{t}) and the algorithm minimises sum_{\textit{t}=1}^{\text{CU}}(\textit{y}(\textit{t})-\textit{coef}'z(\textit{t}))^2) where \textit{t0}=\text{maxi}(\text{maxi}(\textit{r},\textit{s})+1,1)).

\textbf{EXAMPLE :}

\begin{verbatim}
//Ex1- Arma model : y(t) = 0.2*u(t-1)+0.01*e(t-1)
ny=1,nu=1,sig=0.01;
Arma=armac(1,[0,0.2],[0,1],ny,nu,sig) //defining the above arma model
u=rand(1,10000,'normal'); //a random input sequence u
y=arsimul(Arma,u); //simulation of a y output sequence associated with u.
Armaest=armax(0,1,y,u); //Identified model given u and y.
Acoeff=Armaest('a'); //Coefficients of the polynomial A(x)
Bcoeff=Armaest('b') //Coefficients of the polynomial B(x)
Dcoeff=Armaest('d'); //Coefficients of the polynomial D(x)
[Ar,Bx,Dx]=arma2p(Armaest) //Results in polynomial form.

//Ex2- Arma1: y_t -0.8*y_{t-1} + 0.2*y_{t-2} = sig*e(t)
ny=1,nu=1,sig=0.001;
// First step: simulation the Arma1 model, for that we define
// Arma2: y_t -0.8*y_{t-1} + 0.2*y_{t-2} = sig*u(t)
// with normal deviates for u(t).
Arma2=armac([1,-0.8,0.2],sig,0,ny,nu,0);
//Definition of the Arma2 arma model (a model with B=sig and without noise!)
u=rand(1,10000,'normal'); // An input sequence for Arma2
\end{verbatim}
arsimul Scilab Function

y = arsimul(Arma2,u); // y = output of Arma2 with input u
// can be seen as output of Arma1.
// Second step: Identification. We look for an Arma model
// y(t) + a1*y(t-1) + a2 *y(t-2) = sig*e(t)
Armallest = armax(2,-1,y,[]);
[A,B,D]=arma2p(Armallest)

AUTHOR: J-Ph. Chancelier.

SEE ALSO: imrep2ss 341, time_id 367, arl2 321, armax 395, frep2tf 338

2.3.5 armax1 ___________________________ armax identification

CALLING SEQUENCE:

[a,b,d,sig,resid]=armax1(r,s,q,y,u,[b0f])

PARAMETERS:

y : output signal
u : input signal
r,s,q : auto regression orders with r >=0, s >=-1.
b0f : optional parameter. Its default value is 0 and it means that the coefficient b0 must be identified. if
b0f=1 the b0 is supposed to be zero and is not identified
a : is the vector [1,a1,....,ar]
b : is the vector [b0,......,bs]
d : is the vector [1,d1,....,dq]
sig : resid=[ sig*echap(1),....,] ;

DESCRIPTION:

armax1 is used to identify the coefficients of a 1-dimensional ARX process:

\[ A(z^{-1})y = B(z^{-1})u + D(z^{-1})\text{sig}\times e(t) \]
\[ e(t) \text{ is a 1-dimensional white noise with variance 1.} \]
\[ A(z) = 1+a1z+...+a_rz^r; \quad ( r=0 \Rightarrow A(z)=1) \]
\[ B(z) = b0+b1z+...+b_s z^s \quad ( s=-1 \Rightarrow B(z)=0) \]
\[ D(z) = 1+d1z+...+d_qz^q \quad ( q=0 \Rightarrow D(z)=1) \]

for the method, see Eykhoff in trends and progress in system identification) page 96. with \( z(t)=[y(t-1),...,y(t-r),u(t),...,u(t-s),e(t-1),...,e(t-q)] \) and \( \text{coef}=[-a1,...,ar,b0,...,b_s,d1,...,dq] \)
\( \text{y(t)}=\text{coef}\times z(t) + \text{sig}\times e(t). \)

A sequential version of the AR estimation where e(t-i) is replaced by an estimated value is used (RLLS).
With q=0 this method is exactly a sequential version of armax

AUTHOR: J.-Ph.C

2.3.6 arsimul ___________________________ armax simulation

CALLING SEQUENCE:

[z]=arsimul(a,b,d,sig,u,[up,yp,ep])
[z]=arsimul(ar,u,[up,yp,ep])

PARAMETERS:

ar : an armax process. See armac.
a : is the matrix[Id,a1,...,ar] of dimension (n,(r+1)*n)

Scilab Group April 1993 396
b : is the matrix [b0, ..., b_s] of dimension (n, (s+1)*m)
d : is the matrix [Id, d_1, ..., d_t] of dimension (n, (t+1)*n)
u : is a matrix (m,N), which gives the entry u(:,j)=u

**sig** : is a (n,n) matrix e_{-}[k] is an n-dimensional Gaussian process with variance I

up, yp : optional parameter which describe the past. up=[u_{-}[0], u_{-}[1], ..., u_{-}[s-1]]; yp=[y_{-}[0], y_{-}[1], ..., y_{-}[t-1]]; ep=[e_{-}[0], e_{-}[1], ..., e_{-}[t-1]]; if they are omitted, the past value are supposed to be zero

z : z=[y(1), ..., y(N)]

**DESCRIPTION**:
simulation of an n-dimensional armax process A(z^{-1}) z(k)= B(z^{-1})u(k) + D(z^{-1})*sig*e(k)

A(z)= Id+a_1*z+...+a_r*z^r; ( r=0 = Id) B(z)= b_0+b_1*z+...+b_s z^s; ( s=-1 = Id) D(z)= Id+d_1*z+...+d_t z^t; ( t=0 = Id) D(z)= Id

z et e are in R^n et u in R^m

**METHOD**:
a state-space representation is constructed and ode with the option "discr" is used to compute z

**AUTHOR** : J-Ph.C.

### 2.3.7 narsimul ................................. armax simulation (using rtitr)

**CALLING SEQUENCE**:

[z]=narsimul(a,b,d,sig,u,[up,yp,ep])

[z]=narsimul(ar,u,[up,yp,ep])

**DESCRIPTION**:
ARMAX simulation. Same as arsimul but the method is different the simulation is made with rtitr

**AUTHOR** : J-Ph. Chancelier ENPC Cergrene

### 2.3.8 noisegen ................................. noise generation

**CALLING SEQUENCE**:

[]=noisegen(pas,Tmax,sig)

**DESCRIPTION**:
generates a Scilab function [b]=Noise(t) where Noise(t) is a piecewise constant function (constant on \{k*pas, (k+1)*pas\}). The value on each constant interval are random values from i.i.d Gaussian variables of standard deviation sig. The function is constant for t<=0 and t>=Tmax.

**EXAMPLE**:

noisegen(0.5,30,1.0);
x=-5:0.01:35;
y=feval(x,Noise);
plot(x,y);

### 2.3.9 odedi .................................................. test of ode

**CALLING SEQUENCE**:

[]=odedi()

**DESCRIPTION**:
Simple tests of ode and arsimul. Tests the option 'discr' of ode

Scilab Group April 1993
### 2.3.10 prbs_a  
**pseudo random binary sequences generation**

**CALLING SEQUENCE:**

\[ \text{[u]} = \text{prbs_a} (n, nc, \text{[ids]}) \]

**DESCRIPTION:**

generation of pseudo random binary sequences \( u = [u_0, u_1, \ldots, u_{(n-1)}] \); \( u \) takes values in \([-1,1]\) and changes at most \( nc \) times its sign. \( \text{ids} \) can be used to fix the date at which \( u \) must change its sign; \( \text{ids} \) is then an integer vector with values in \([1:n]\).

**EXAMPLE:**

\[ u = \text{prbs_a}(50, 10); \]
\[ \text{plot2d2} ("onn", (1:50)', u', 1, "151", ', ', [0, -1.5, 50, 1.5]); \]

### 2.3.11 reglin  
**Linear regression**

**CALLING SEQUENCE:**

\[ \text{[a, b, sig]} = \text{reglin} (x, y) \]

**DESCRIPTION:**

solve the regression problem \( y = a \cdot x + b \) in the least square sense. \( \text{sig} \) is the standard deviation of the residual. \( x \) and \( y \) are two matrices of size \( x(p,n) \) and \( y(q,n) \), where \( n \) is the number of samples.

The estimator \( a \) is a matrix of size \( (q,p) \) and \( b \) is a vector of size \( (q,1) \).

// simulation of data for \( a(3,5) \) and \( b(3,1) \)
\[ x = \text{rand}(5, 100); \]
\[ a = \text{testmatrix} ("magi", 5); a = a(1:3,:); \]
\[ b = [9; 10; 11] \]
\[ y = a \cdot x + b \cdot \text{ones}(1, 100) + 0.1 \cdot \text{rand}(3, 100); \]

// identification
\[ [a, b, sig] = \text{reglin} (x, y); \]
\[ \text{maxi(abs(aa - a))} \]
\[ \text{maxi(abs(bb - b))} \]

// an other example : fitting a polynom
\[ f = 1:100; x = [f .* f; f]; \]
\[ y = [2, 3] \cdot x + 10 \cdot \text{ones}(f) + 0.1 \cdot \text{rand}(f); \]
\[ [a, b] = \text{reglin} (x, y); \]

**SEE ALSO:**  
pinv 525, leastsq 427, qr 528
2.4 Examples
2.4.1 artest ___________________________ arnold dynamical system

CALLING SEQUENCE :

artest(f_l,[odem,xdim,npts])
arnold(t,x)
 iarf([a])

PARAMETERS :

f_l : can be "arnold" or arnold. It is the name of the external for the arnold system. If f_l is set to "arnold" a Fortran coded version of the arnold system where a(1:6)=1 will be used and if f_l is set to arnold a Scilab coded version will be used. arnold is a Scilab macro coding the Arnold system. This macro is loaded when calling artest.
 iarf : is used to fix the values of a in the Scilab coded case. a is a vector of size 6.
 odom, xdim, npts : are optional arguments. Their meaning and syntax can be found in the portr3d help

DESCRIPTION :

A call to the function artest() will interactively display a phase portrait of a the following dynamical system :

\[
\begin{align*}
\dot{y}(1) &= a(1)\cos(y(2)) + a(2)\sin(y(3)) \\
\dot{y}(2) &= a(3)\cos(y(3)) + a(4)\sin(y(1)) \\
\dot{y}(3) &= a(5)\cos(y(1)) + a(6)\sin(y(2))
\end{align*}
\]

SEE ALSO: portr3d 405, ode 431, chaintest 401, lotest 403

2.4.2 bifish . shows a bifurcation diagram in a fish population discrete time model

CALLING SEQUENCE :

bifish([f_ch])

PARAMETERS :

f_ch : can be one of fish, fishr and fishr2. This option selects the population model.

DESCRIPTION :

The dynamical system fish is the following :

\[
\begin{align*}
y &= b \exp(-0.1*(x(k)_1 + x(k)_2)) \\
x(k+1) &= [y \ 2*y \ s \ 0.0] * x(k)
\end{align*}
\]

and the parameters s evolves to show the bifurcation diagram. fishr and fishr2 are constructed as above but with added white noises.

fishr
\[
y = b \exp(-0.1*(xk(1) + xk(2))) \\
xkp1 = [y \ 2*y \ s*(1+0.1*(rand()-0.5)) \ 0.0] * xk
\]

fishr2
\[
z = \exp(-0.1*(xk(1) + xk(2))) \\
xkp1 = [b*z**2*(1+0.1*(rand()-0.5)) \ s*(1+0.1*(rand()-0.5)) \ 0.0] * xk
\]

The three macros fish, fishr, fishr2 are loaded in Scilab when calling bifish.

SEE ALSO: ode 431
2.4.3 **boucle** phase portrait of a dynamical system with observer

CALLING SEQUENCE:

\[
[] = \text{boucle}(fch, [abruit, xdim, npts, farrow])
\]

PARAMETERS:

- \( fch \): Scilab macro. \( fch \) is supposed to be an observed-controlled system with noisy output of state dimension 4 ([x;xchap] is of dimension 4). \( fch \) can be created with the macro \text{obscont1} or can be set to one of the two following string which gives pre computed examples
  - "bcomp": for a non-linear competition model.
  - "lcomp": for a linear example.
- \( abruit \): give the noise variance.
- \( xdim, npts, farrow \): See \text{portrait}

DESCRIPTION:

Phase portrait of dynamical systems.

SEE ALSO: \text{portrait} 406, \text{ode} 431, \text{obscont1} 405

2.4.4 **chaintest** a three-species food chain model

CALLING SEQUENCE:

\[
\text{chaintest}([f\_l,b1,odem,xdim,npts])
\]

\[
[xdot] = \text{chain}(t,x)
\]

\[
[z1] = \text{ch}_f1(u)
\]

\[
[z2] = \text{ch}_f2(u)
\]

PARAMETERS:

- \( f\_l \): the name of the macro which code the three-species food chain model (default value \text{chain}).
- \(odem, xdim, npts \): are optional arguments. Their meaning and syntax can be found in the \text{portr3d} help

DESCRIPTION:

A call to the function \text{chaintest}() will interactively display a phase portrait of a three-species food chain model given by:

\[
\begin{align*}
ff1 &= f1(x(1)) \\
ff2 &= f2(x(2)) \\
xdot1 &= x(1)*(1-x(1)) - ff1*x(2) \\
xdot2 &= ff1*x(2) - ff2*x(3) - 0.4*x(2) \\
xdot3 &= ff2*x(3) - 0.01*x(3)
\end{align*}
\]

and

\[
\begin{align*}
f1(u) &= 5*u/(1+b1*u) \\
f2(u) &= 0.1*u/(1+2*u)
\end{align*}
\]

The default value for \( b1 \) is 3.0.

The Scilab macros \text{chain}(t,x),f1(u),f2(u) code the dynamical system

SEE ALSO: \text{portr3d} 405, \text{ode} 431
2.4.5  gpeche ................................................. a fishing program

CALLING SEQUENCE :

[xk, ukp1] = gpeche(uk, pasg)
[ut] = pche(t)
[pdot] = pechep(t, p)

DESCRIPTION :

gpeche  Iterates a gradient method on a fishing problem Computes the trajectory associated to the command law uk  prints the cost value and computes a new control.

2.4.6  fusee  ............... a set of Scilab macro for a landing rocket problem

FUSEE :

[xdot] = fusee(t, x)

Dynamical motion equation for the rocket

FINIT :

finit()

Initialises the following parameters for rocket landing.

k  : The acceleration of the rocket engines
gamma : The moon gravity acceleration.
umax : the gaz ejection flow out.
mcap : the mass of the space capsule.
cpen : penalisation in the cost function of the final state.

FUSEEGRAD :

[ukp1] = fuseegrad(niter, ukp1, pasg)

niter : number of gradient iteration steps.
ukp1 : initial control value ( vector of sie 135 )
pasg : the gradient step value.

DESCRIPTION :
Iterate a gradient method and returns the computed control.

FUSEEP :

[pdot] = fuseep(t, p)

DESCRIPTION :
adjoint equation for the landing rocket problem.

POUSSE :

[ut] = pousse(t)

return the value of a piece wise constant control build on the discrete control uk

UBANG :

[uk] = ubang(tf, tcom)
FCOUT:
\[ [c, xk, pk, ukp1] = fcout(tf, uk, pasg) \]

DESCRIPTION:
returns a bang-bang control, 0 form time 0 to tcom and 1 form tcom to tf.

SFUSEE:
\[ [] = sfusee(tau, h0, v0, m0, Tf) \]

DESCRIPTION:
computes the rocket trajectory when a bang-bang control is used. \( \tau \) is the commutation time.

EQUAD:
\[ [xk, pk] = equad(tf, uk) \]

DESCRIPTION:
Computes the state and adjoint state of the rocket system for a given control \( ur \).

TRAJ:
\[ [xt] = traj(t) \]
returns a piece wise value of the mass evolution.

2.4.7 lotest __________________________ demo of the Lorenz attractor

CALLING SEQUENCE:
\[ [] = lotest([f_l,odem,xdim,npts,pinit]) \]
\[ y = lorenz(t, x) \]
\[ [] = ilo(sig, ro, beta) \]
\[ [] = ilof(sig, ro, beta) \]

PARAMETERS:
\( f_l \) : can be "loren" or lorenz. it is the name of the external for the Lorenz system. "loren" will use a Fortran coded version of the lorenz system and arnold will and loren will use a Scilab coded version. lorenz is the Scilab macro which code the lorenz system. This macro is loaded when calling lotest.
\( ilo, ilof \) are used to fix the parameters of the Fortran and Scilab code version of the Lorenz system. \( odem, xdim, npts \) are optional arguments. Their meaning and syntax can be found in the portr3d help

DESCRIPTION:
A call to the function lotest() will interactively display a phase portrait of a the following dynamical system
Obscont1 Scilab Function

\[
y(1) = \text{sig} \times (x(2) - x(1)); \\
y(2) = \text{ro} \times x(1) - x(2) - x(1) \times x(3); \\
y(3) = -\beta \times x(3) + x(1) \times x(2);
\]

See Also: портр3д 405, ode 431, chaintest 401, lotest 403

2.4.8 mine  a mining problem

Calling Sequence:

[cout, feed] = mine(n1, n2, uvect)

Parameters:

n1 : Number of discrete point for the state.

n2 : Number of time step

uvect : a row vector which gives the possible control value (integer values). For example, \( u = [-1, 0, 1] \)

means that at each step we come down one step or stay at the same level or rise one step.

cout(n1, n2) : The Bellman values.

feed(n1, n2) : The feedback Law.

Description:

Dynamic programming applied to an optimal extraction of ore in an opencast mine. The extraction is done as follows: the steam shovel move forward for \( k = 1, 2, \ldots, n2 \) at each step it takes the ore, then move up or down (or stay at the same level) according to the control value to reach another level at next step. The extraction process must maximise the following cost:

\[
\min_{k=1}^{n2-1} \left( f(x(k), k) + V_F(x, n2) \right)
\]

where \( x(k+1) = x(k) + u \times x(k) \) is the trajectory depth at step \( k \) \( (x=1 \) is the ground level). The instantaneous cost \( f(i, k) \) stands for the benefit of digging at depth \( i \) at position \( k \). It must be given as a Scilab macro \( f \).

\[
y = \text{ff}_\text{o}(x, k)
\]

and for efficiency \( f \) must accept and return column vectors for \( x \) and \( y \).

\( V_F(i, n2) \) is a final cost which is set so as to impose the steam shovel to be at ground level at position \( n2 \).

FF_o:

Calling Sequence:

[] = showcost(n1, n2, teta, alpha)

Description:

Shows a 3D representation of the instantaneous cost.
2.4.9  **obscont1**  ___________________________  a controlled-observed system

**CALLING SEQUENCE:**

```
[macr]=obscont1(sysn)
```

**PARAMETERS:**

sysn  : A Scilab string which gives the name of the controlled system.
gaincom, gainobs  : column vectors giving the requested gains
macr  : a new Scilab function which simulates the controlled observed system.

```
[xidot]=macr(t,x1,abruit,pas,n)
x1=[x;xchap],
```

**DESCRIPTION:**

This macros return a new function which computes the controlled observed version of a linearised system around the \((xe,ue)\) point.

Before calling this function, a noise vector \(br\) should be created. The equilibrium point \((xe,ue)\) should be given as a global Scilab. The linearised system \(f,g,h\) and the two gain matrices \(l,k\) are returned as global Scilab data.

2.4.10  **portr3d**  ___________________________  3 dimensional phase portrait.

**CALLING SEQUENCE:**

```
[]=portr3d(f,[odem,xdim,npts,pinit])
```

**PARAMETERS:**

f  : a Scilab external which gives the field of the dynamical system. Hence it can be a macro name which computes the field at time \(t\) and point \(x\) \([y]=f(t,x,u)\) or a list \(l\) \([y]=l(t,x,u)\) where \(l\) is a macro of type 

rest  : The other parameters are optional. If omitted they will be asked interactively
odem  : gives the integration method to use. The value "default" can be used, otherwise see ode for a complete set of possibilities
npts  : a vector of size \((2,10)\) \([\text{number-of-points,step}]\) gives the step for integration and the number of requested points. The solution will be calculated and drawn for \(t=0:step:(step*\text{number-of-points})\)
xdim  : \([xmin,xmax,ymin,ymax,zmin,zmax]\) the boundaries of the graphic frame.
pinit  : initial values for integration. A set of initial points can be given in a matrix

```
pinit = [x(1), x1(1),..., xn(1)
x(2), x1(2),..., xn(2)
x(3), x1(3),..., xn(3)].
```

**DESCRIPTION:**

Interactive integration and display of a 3 dimensional phase portrait of a dynamical system \(dx/dt=f(t,x,u)\)
(where \(u\) is an optional parameter)

**SEE ALSO:**  ode 431

Scilab Group  
April 1993  
405
2.4.11 portrait __________________________ 2 dimensional phase portrait.

CALLING SEQUENCE :

[] = portrait (f, [odem, xdim, npts, pinit])

PARAMETERS :

f : a Scilab external which gives the field of the dynamical system. Hence it can be a macro name which computes the field at time t and point x [y]=f(t,x,[u]) or a list list(f1,u1) where f1 is a macro of type [y]=f1(t,x,u) or a character string. The macro can be used to simulate a continuous or discrete system and in case of discrete system the second parameter must be set to 'discrete'

rest : The other parameters are optional. If omitted they will be asked interactively

odem : gives the integration method to use. The value "default" can be used, otherwise see ode for a complete set of possibilities

npts : a vector of size (2,10) [number-of-points,step] gives the step for integration and the number of requested points. The solution will be calculated and drawn for time=0:step*(number-of-points)

xdim : [xmin,xmax,ymin,ymax,zmin,zmax] the boundaries of the graphic frame.

pinit : initial values for integration. A set of initial points can be given in a matrix

\[
pinit = \begin{bmatrix}
  x0(1), & x1(1), & \ldots, & xn(1) \\
  x0(2), & x1(2), & \ldots, & xn(2) \\
  x0(3), & x1(3), & \ldots, & xn(3)
\end{bmatrix}.
\]

DESCRIPTION :

Interactive integration and display of a 2 dimensional phase portrait of a dynamical system dx/dt=f(t,x,[u]) (where u is an optional parameter)

EXAMPLE :

a = rand (2,2)
deff ('[ydot]=l_s(t,y)','ydot=a*y')
portrait(l_s)

SEE ALSO : ode 431

2.4.12 recur __________________________ a bilinear recurrent equation

CALLING SEQUENCE :

[y] = recur(x0, var, k, n) 
[integr] = logr(k, var)

DESCRIPTION :

computes solutions of a bilinear recurrent equation

\[x(i+1) = -x(i)*(k + sqrt(var)*br(i))\]

with initial value \(x0\) and driven by a white noise of variance \(var\). Trajectories are drawn and the empirical Lyapunov exponent is returned ( \(x(i)\) is not to much different from \(exp(y*i)\) )

A theoretical computation of the Lyapunov exponent is given by

\[[integr] = logr(k, var)\]
2.4.13 systems a collection of dynamical system

CALLING SEQUENCE:

[] = systems()

DESCRIPTION:
A call to this function will load into Scilab a set of macros which describes dynamical systems. Their parameters can be initiated by calling the routine tdinit().

BIOREACT:

[ydot] = biorecat(t,x)

a bioreactor model,

x(1) is the biomass concentration
x(2) is the sugar concentration

\[
\begin{align*}
xdot(1) &= \mu_{td}(x(2)) \cdot x(1) - \text{debit} \cdot x(1); \\
xdot(2) &= -k \cdot \mu_{td}(x(2)) \cdot x(1) - \text{debit} \cdot x(2) + \text{debit} \cdot x2in;
\end{align*}
\]

where \( \mu_{td} \) is given by

\[
\mu_{td}(x) = \frac{x}{1 + x};
\]

COMPET:

[xdot] = compet(t,x [,u])

a competition model. \( x(1), x(2) \) stands for two populations which grows on a same resource. \( 1/u \) is the level of that resource (1 is the default value).

\[
\begin{align*}
xdot(1) &= \frac{ppr}{ppk} \cdot x(1) - \frac{ppa}{ppk} \cdot x(1) \cdot x(2) \\
xdot(2) &= \frac{pps}{ppl} \cdot x(2) - \frac{ppb}{ppl} \cdot x(1) \cdot x(2)
\end{align*}
\]

The macro \( [x_e] = \text{equilcom}(ue) \) computes an equilibrium point of the competition model and a fixed level of the resource \( ue \) (default value is 1).

The macro \( [f, g, h, \text{linsy}] = \text{lincomp}([ue]) \) gives the linearisation of the competition model (with output \( y=x \)) around the equilibrium point \( x_e = \text{equilcom}(ue) \). This macro returns \( [f, g, h] \) the three matrices of the linearised system and \( \text{linsy} \) which is a Scilab macro \( [ydot] = \text{linsy}(t,x) \) which computes the dynamics of the linearised system.

CYCLLIM:

[xdot] = cycllim(t,x)

a model with a limit cycle

\[
xdot = a \cdot x + \text{geps} \cdot (1 - \|x\|^{**2}) \cdot x
\]

LINEAR:

[xdot] = linear(t,x)

a linear system

BLINPER:

[xdot] = linper(t,x)
a linear system with quadratic perturbations.

**POP:**

\[ \dot{x} = \text{pop}(t,x) \]

a fish population model

\[ \dot{x} = 10x(1-x/K) - \text{peche}(t)x \]

**PROIE:**

a Predator prey model with external insecticide.

\[ \dot{x} = \text{p}_p(t,x,u) \]

\[
\begin{align*}
x(1) & \quad \text{The prey (that we want to kill)} \\
x(2) & \quad \text{the predator (that we want to preserve)} \\
u & \quad \text{mortality rate due to insecticide which destroys both prey and predator (default value } u=0) \\
\end{align*}
\]

\[
\begin{align*}
\dot{x}(1) &= \text{ppr}x(1)(1-x(1)/\text{ppk}) - \text{ppa}x(1)x(2) - u x(1) \\
\dot{x}(2) &= -\text{ppm}x(2) + \text{ppb}x(1)x(2) - u x(2)
\end{align*}
\]

The macro \([x_e]=\text{equilpp}([u_e])\) computes the equilibrium point of the \(p_p\) system for the value \(u_e\) of the command. The default value for \(u_e\) is 0.

\[
\begin{align*}
x_e(1) &= (\text{ppm}+u_e)/\text{ppb}; \\
x_e(2) &= (\text{ppr}(1-x_e(1)/\text{ppk})-u_e)/\text{ppa};
\end{align*}
\]

**LINCOM:**

\[ \dot{x} = \text{lincom}(t,x,k) \]

linear system with a feedback

\[ \dot{x} = ax + b(-kx) \]

SEE ALSO: tdinit 409

2.4.14 tangent linearization of a dynamical system at an equilibrium point

**CALLING SEQUENCE:**

\([f,g,\text{newm}]=\text{tangent}(ff,x_e,[ue])\)

**PARAMETERS:**

\(ff\) : a string which gives the name of the Scilab macro which codes the system

\(x_e\) : column vector which gives the equilibrium point for the value \(u_e\) of the parameter

\(u_e\) : real value.

\(f,\ g\) : two matrices for the linearised system \(\dot{xdot}=f.dx + g.du\)

\(\text{newm}\) : A new macro of type [\(y]=\text{newm}(t,x,u)\) which computes the field of the linearised system (newm(t,x,ue)=0)

**DESCRIPTION:**

linearises around the equilibrium point \((x_e,u_e)\) the vector field of the dynamical system given by a Scilab macro \(ff, xdot=ff(t,x,[u])\). The dynamical system is supposed to be autonomous.
2.4.15  **tdinit**  interactive initialisation of the tdc's dynamical systems

**CALLING SEQUENCE :**

tdinit()

**DESCRIPTION :**

This macro can be used to interactively define the parameters needed by the dynamical systems described in systems:

- bioreactor model
- competition model
- system with limit cycle
- linear system
- quadratic model
- linear system with a feedback

**SEE ALSO :**  portrait 406, systems 407
2.5 Non-linear tools (optimization and simulation)
2.5.1 bvode boundary value problems for ODE

prey predatory model

**CALLING SEQUENCE:**

```plaintext
[z]=bvode(points,ncomp,m,aleft,aright,zeta,ipar,ltol,tol,fixpnt,...
fsub1,dfsub1,gsubi,dgsub1,guess1)
```

**PARAMETERS:**

- **z**: The solution of the ode evaluated on the mesh given by points
- **points**: an array which gives the points for which we want the solution
- **ncomp**: number of differential equations ($ncomp \leq 20$)
- **m**: a vector of size $ncomp$. $m(j)$ gives the order of the $j$-th differential equation

\[
mstar = \sum_{i=1}^{ncomp} m(i) \leq 40
\]

- **aleft**: left end of interval
- **aright**: right end of interval
- **zeta**: $zeta(j)$ gives j-th side condition point (boundary point). must have $zeta(j) \leq zeta(j+1)$. all side condition points must be mesh points in all meshes used, see description of ipar(11) and fixpnt below.

**ipar**: an integer array dimensioned at least 11. a list of the parameters in ipar and their meaning follows some parameters are renamed in bvode; these new names are given in parentheses.

- **ipar(1) (=nonlin)**
  - = 0 if the problem is linear
  - = 1 if the problem is nonlinear
- **ipar(2) (=k)**
  - number of collocation points per subinterval ($= k$) where $\max m(i) \leq k \leq 7$. if $ipar(2) = 0$ then bvode sets $k = \max(\max m(i) + 1, 5 - \max m(i))$
- **ipar(3) (=n)**
  - number of subintervals in the initial mesh. if $ipar(3) = 0$ then bvode arbitrarily sets $n = 5$.
- **ipar(4) (=ntol)**
  - number of solution and derivative tolerances. we require $0 < ntol < mstar$.
- **ipar(5) (=ndimf)**
  - dimension of fspace. its size provides a constraint on nmax. choose $ipar(5)$ according to the formula
  \[
ipar(5) \geq nmaxn_s, \quad \text{where} \quad n_s = 4 + 3mstar + (5 + k_d)k_dm + (2mstar - nrec)2mstar
\]
- **ipar(6) (=ndimi)**
  - dimension of ispace. its size provides a constraint on nmax. the maximum number of subintervals. choose $ipar(6)$ according to the formula
  \[
ipar(6) \geq nmaxn_t, \quad \text{where} \quad n_t = 3 + k_d m \quad k_dm = k_d + mstar \quad k_d = kncomp
\]
- **ipar(7) (=iprint)**
  - = -1 for full diagnostic printout
  - = 0 for selected printout
  - = 1 for no printout
- **ipar(8) (=iread)**
  - = 0 causes bvode to generate a uniform initial mesh.
  - = xx Other values are not implemented yet in Scilab
  - = 1 if the initial mesh is provided by the user. it is defined in fspace as follows: the mesh
    \[
    aleft = x(1) < x(2) < \ldots < x(n) < x(n+1) = aright
    \]
    will occupy $fspace(1), \ldots, fspace(n+1)$. the user needs to supply only the interior mesh points $fspace(j) = x(j), \ j = 2, \ldots, n$. 

Scilab Group June 1993
= 2 if the initial mesh is supplied by the user as with ipar(8)=1, and in addition no adaptive mesh selection is to be done.

ipar(9) (= iguess)
= 0 if no initial guess for the solution is provided.
= 1 if an initial guess is provided by the user in subroutine guess.
= 2 if an initial mesh and approximate solution coefficients are provided by the user in fspace. (the former and new mesh are the same).
= 3 if a former mesh and approximate solution coefficients are provided by the user in fspace, and the new mesh is to be taken twice as coarse; i.e., every second point from the former mesh.
= 4 if in addition to a former initial mesh and approximate solution coefficients, a new mesh is provided in fspace as well. (see description of output for further details on iguess = 2, 3, and 4.)

ipar(10) = number of fixed points in the mesh other than

ipar(11) = number of fixed points in the mesh other than aleft and aright. (= nfxpnt, the dimension of fixpnt) the code requires that all side condition points other than aleft and aright (see description of zeta) be included as fixed points in fixpnt.

ltol an array of dimension ipar(4). ltol(j) = 1 specifies that the j-th tolerance in tol controls the error in the l-th component of z(u). also require that

\[ l \leq \text{ltol}(1) < \text{ltol}(2) < ... < \text{ltol}(\text{ntol}) \leq mstar \]

tol an array of dimension ipar(4). tol(j) is the error tolerance on the ltol(j)-th component of z(u). thus, the code attempts to satisfy for j=1,...,ntol on each subinterval

\[ |z(v) - z(u)|_{ltol(j)} \leq \text{tol}(j) * |z(u)|_{ltol(j)} + \text{tol}(j) \]

if v(x) is the approximate solution vector.

fixpnt an array of dimension ipar(11). it contains the points, other than aleft and aright, which are to be included in every mesh.

externals The function fsub, dfsub, gsub, dgsb, guess are Scilab externals i.e. functions (see syntax below) or the name of a Fortran subroutine (character string) with specified calling sequence or a list. An external as a character string refers to the name of a Fortran subroutine. The Fortran coded function interface to bvode are specified in the file fcol.f.

fsub name of subroutine for evaluating

\[ f(x, z(u(x))) = (f_1, \ldots, f_{\text{ncomp}})^T \]

at a point x in (aleft, aright). it should have the heading [f] = fsub(x, z) where f is the vector containing the value of fi(x, z(u)) in the i-th component and

\[ z(u(x)) = (z_1, \ldots, z_{mstar})^T \]

is defined as above under purpose.

dfsub name of subroutine for evaluating the Jacobian of f(x, z(u)) at a point x. it should have the heading [df] = dfsub(x, z) where z(u(x)) is defined as for fsub and the (ncomp) by (mstar) array df should be filled by the partial derivatives of f, viz, for a particular call one calculates

\[ df(i,j) = df_i / dz_j, \quad i = 1, \ldots, \text{ncomp}, \quad j = 1, \ldots, \text{mstar}. \]

gsub name of subroutine for evaluating the i-th component of

\[ g(x, z(u(x))) = g_i(zeta(i), z(u(zeta(i)))) \]

at a point x = zeta(i) where 1 \leq i \leq mstar. it should have the heading [g] = gsub(i, z) where z(u) is as for fsub, and i and g = gi are as above. note that in contrast to f in fsub, here only one value per call is returned in g.

Scilab Group

June 1993

412
The Scilab function `bvode` is used for solving multi-point boundary value problems for a mixed order system of ODEs given by:

\[ g_j(zeta; z(u(zeta(j)))) = 0 \quad j = 1, \ldots, \text{mstar} \]

where \( z(u(x)) \) is the exact solution vector, \( u_m(x) \) is the \( m \)-th derivative of \( u \), and \( f_i(x, z(u(x))) \) is a (generally) nonlinear function of \( x \) and \( u \).

**Guess**

The name of the subroutine for initial approximation is `guess(x)`, and the vector of the \( m \)-th derivatives of \( u \) is represented by \( \text{dmval}(u(x)) \). It should have the heading:

\[ [z, \text{dmval}] = \text{guess}(x) \]

where \( z(u) \) is as for `fsub`, \( m \) as for `gsub`, and the \( m \)-vector of \( m \)-th derivatives of \( u \) is specified for any \( x \), \( \text{aleft} \leq x \leq \text{aright} \).

**Example**

```scilab
deff('df=dfsub(x,z)','df=[0,0,-6/x**2,-6/x]

deff('f=fsub(x,z)','f=(1 -6*x**2*z(4)-6*x*z(3))/x**3

deff('g=gsub(i,z)','g=[z(1),z(3),z(1),z(3)];g=g(i)

deff('dg=dgsub(i,z)',
    ['dg=[1,0,0,0;0,0,1,0;1,0,0,0;0,0,1,0
    dg=dg(i,:)

deff('u=trusol(x)',
    u=0*ones(4,1);
    u(1) = 0.25*(10*log(2)-3)*(1-x) + 0.5 *((1/x + (3+x)*log(x) - x)
    u(2) = -0.25*(10*log(2)-3) + 0.5 *(-1/x**2 + (3+x)/x + log(x)
    u(3) = 0.5*(-6/x**4 - 1/x/x + 6/x**3)

fixpnt=0;m=4;
ncomp=1;aleft=1;aright=2;
zeta=[1,1,2,2];
ipar=zeros(1,11);
ipar(3)=1;ipar(4)=2;ipar(5)=2000;ipar(6)=200;ipar(7)=1;ltol=[1,3];tol=[1.e-11,1.e-11];res=aleft:0.1:aright;
z=bvode(res,ncomp,m,aleft,aright,zeta,ipar,ltol,tol,fixpnt,...
fsub,dfsub,gsub,dgsub,guess)
```

**Notes**

- The order of the differential equations satisfies \( 1 \leq m(i) \leq 4 \).
- The boundary points satisfy:
  \[ \text{aleft} \leq zeta(1) \leq \ldots \leq zeta(\text{mstar}) \leq \text{aright} \]

**Description**

This package solves a multi-point boundary value problem for a mixed order system of ODEs given by:

\[ u^{(m(i))}_i = f(x, z(u(x))) \quad i = 1, \ldots, \text{ncomp} \quad \text{aleft} < x < \text{aright} \]

where \( u = (u_1, u_2, \ldots, u_{\text{ncomp}}) \) is the exact solution vector, \( u^{(m(i))}_i \) is the \( m(i) \)th derivative of \( u_i \).
2.5.2 colnew ____________________________ boundary value problems for ODE

CALLING SEQUENCE:
This function has been renamed bvode.

2.5.3 dasrt ____________________________ DAE solver with zero crossing

CALLING SEQUENCE:
[r, nn, [,hd]] = dasrt(x0, t0, t [, atol, [rtol]], res [, jac], ng, surf [, info] [,hd])

PARAMETERS:

x0 : is either y0 (ydot0 is estimated by dassl with zero as first estimate) or the matrix [y0
     ydot0]. g(t, y0, ydot0) must be equal to zero. If you only know an estimate of ydot0 set info(7)=1
y0 : real column vector of initial conditions.
ydot0 : real column vector of the time derivative of y at t0 (may be an estimate).
t0 : real number is the initial instant.
t : real scalar or vector. Gives instants for which you want the solution. Note that you can get solution at each dassl's step point by setting info(2)=1.
nn : a vector with two entries [times num] times is the value of the time at which the surface is
crossed, num is the number of the crossed surface
atol, rtol : real scalars or column vectors of same size as y. atol, rtol give respectively absolute
and relative error tolerances of solution. If vectors the tolerances are specified for each component of
y.
res : external (function or list or string). Computes the value of g(t, y, ydot).
function : Its calling sequence must be [r, ires]=res(t, y, ydot) and res must return the
residue r=g(t, y, ydot) and error flag ires. ires = 0 if res succeeds to compute r, -1 if residue is locally not defined for (t, y, ydot), -2 if parameters are out of admissible range.
list : it must be as follows:

list(res, x1, x2, ...)

where the calling sequence of the function res is now

r=res(t, y, ydot, x1, x2, ...)

res still returns r=g(t, y, ydot) as a function of (t, y, ydot, x1, x2, ...).
string : it must refer to the name of a fortran subroutine (see source code of fresd.f).
jac : external (function or list or string). Computes the value of dg/dy+cj*dg/dydot for a given
value of parameter cj

AUTHOR : u. ascher, department of computer science, university of british columbia, vancouver, b. c.,
canada v6t 1w5
       g. bader, institut f. angewandte mathematik university of heidelberg im neuenheimer feld 294d-6900
       heidelberg 1

Fotran subroutine colnew.f
SEE ALSO : fort 43, link 303, external 38, ode 431, dassl 416
Scilab Group June 1993 414
function : Its calling sequence must be \( r = \text{jac}(t, y, ydot, cj) \) and the \( \text{jac} \) function must return \( r = \frac{dg(t, y, ydot)}{dy} + cj \frac{dg(t, y, ydot)}{dydot} \) where \( cj \) is a real scalar.

list : it must be as follows

\[
\text{list(jac,x1,x2,...)}
\]

where the calling sequence of the function \( \text{jac} \) is now

\[
r = \text{jac}(t, y, ydot, x1, x2,...)
\]

\( \text{jac} \) still returns \( \frac{dg}{dy} + cj \frac{dg}{dydot} \) as a function of \( (t, y, ydot, cj, x1, x2,\ldots) \).

character string : it must refer to the name of a fortran subroutine (see source code of \( \text{jacdd}.f \)).

surf : external (function or list or string). Computes the value of the column vector \( \text{surf}(t, y) \) with \( ng \) components. Each component defines a surface.

function : Its calling sequence must be \( \text{surf}(t, y) \)

list : it must be as follows

\[
\text{list(surf,x1,x2,...)}
\]

where the calling sequence of the function \( \text{surf} \) is now

\[
r = \text{surf}(t, y, x1, x2,...)
\]

character string : it must refer to the name of a fortran subroutine (see source code of \( \text{fsurfd}.f \)) in directory \text{SCDIR/default}.

info : list which contains 7 elements, default value is \( \text{list([],0,[],[],[],0,0)} \).

\( \text{info(1)} \) : real scalar which gives the maximum time for which \( g \) is allowed to be evaluated or an empty matrix \([\,]\) if no limits imposed for time.

\( \text{info(2)} \) : flag which indicates if \( \text{dassl} \) returns its intermediate computed values (\( \text{flag}=1 \)) or only the user specified time point values (\( \text{flag}=0 \)).

\( \text{info(3)} \) : 2 components vector which give the definition \([ml, mu]\) of band matrix computed by \( \text{jac} \);

\( r(i - j + ml + mu + 1, j) = \frac{dg(i)/dy(j) + cj*dg(i)/dydot(j)}{} \). If \( \text{jac} \) returns a full matrix set \( \text{info(3)}=[\,] \).

\( \text{info(4)} \) : real scalar which gives the maximum step size. Set \( \text{info(4)}=[\,] \) if no limitation.

\( \text{info(5)} \) : real scalar which gives the initial step size. Set \( \text{info(5)}=[\,] \) if not specified.

\( \text{info(6)} \) : set \( \text{info(6)}=1 \) if the solution is known to be non negative, else set \( \text{info(6)}=0 \).

\( \text{info(7)} \) : set \( \text{info(7)}=1 \) if \( y_{dot0} \) is just an estimation, \( \text{info(7)}=0 \) if \( g(t_0, y_0, y_{dot0}) = 0 \).

hd : real vector which allows to store the \text{dassl} context and to resume integration.

\( r \) : real matrix. Each column is the vector \([t; x(t); x_{dot}(t)]\) where \( t \) is time index for which the solution had been computed.

DESCRIPTION :
Solution of the implicit differential equation

\[
\begin{align*}
g(t, y, y_{dot}) &= 0 \\
y(t_0) &= y_0 \quad \text{and} \quad y_{dot}(t_0) = y_{dot0}
\end{align*}
\]

Returns the surface crossing instants and the number of the surface reached in \( nn \).

Detailed examples can be found in \text{SCIDIR/tests/dassldasrt.tst}.

EXAMPLE :

//dy/dt = ((2*log(y)+8)/t -5)*y,  y(1) = 1,  1<=t<=6
//g1 = ((2*log(y)+8)/t - 5)*y
//g2 = log(y) - 2.2491
y0=1;t=2:6;t0=1;y0d=3;
atol=1.d-6;rtol=0;ng=2;
dassl Scilab Function

```scilab
deff('[delta,ires]=res1(t,y,ydot)','ires=0;delta=ydot-((2*log(y)+8)/t-5)*y')
deff('[rts]=gr1(t,y)','rts=[((2*log(y)+8)/t-5)*y;log(y)-2.2491]')

[yy,nn]=dasrt([y0,y0d],t0,t,atol,rtol,res1,ng,gr1);
// (Should return nn=[2.4698972 2])
```

See Also: 
ode 431, dassl 416, impl 421, fort 43, link 303, external 38

### 2.5.4 dassl differential algebraic equation

**CALLING SEQUENCE:**

```scilab
[r [,hd]]=dassl(x0,t0,t [,atol,[rtol]],res [,jac] [,info] [,hd])
```

**PARAMETERS:**

- **x0**: is either `y0` (`ydot0` is estimated by dassl with zero as first estimate) or the matrix `[y0 ydot0]`. `g(t,y0,ydot0)` must be equal to zero. If you only know an estimate of `ydot0` set `info(7)=1`.
- **y0**: real column vector of initial conditions.
- **ydot0**: real column vector of the time derivative of `y` at `t0` (may be an estimate).
- **t0**: real number is the initial instant.
- **t**: real scalar or vector. Gives instants for which you want the solution. Note that you can get solution at each dassl's step point by setting `info(2)=1`.
- **atol,rtol**: real scalars or column vectors of same size as `y`. `atol,rtol` give respectively absolute and relative error tolerances of solution. If vectors the tolerances are specified for each component of `y`.
- **res**: external (function or list or string). Computes the value of `g(t,y,ydot)`.
- **function**: Its calling sequence must be `[r,ires]=res(t,y,ydot)` and `res` must return the residue `r=g(t,y,ydot)` and error flag `ires`. `ires = 0` if `res` succeeds to compute `r`, `-1` if residue is locally not defined for `(t,y,ydot)`, `-2` if parameters are out of admissible range.
- **list**: it must be as follows:

```
list(res,x1,x2,...)
```

where the calling sequence of the function `res` is now

```scilab
r=res(t,y,ydot,x1,x2,...)
```

`res` still returns `r=g(t,y,ydot)` as a function of `(t,y,ydot,x1,x2,...)`.

- **string**: it must refer to the name of a fortran subroutine (see source code of of Ex-dassl.f in routines/default).  

- **jac**: external (function or list or string). Computes the value of `dg/dy+cj*dg/dydot` for a given value of parameter `cj`.
- **function**: Its calling sequence must be `r=jac(t,y,ydot,cj)` and the `jac` function must return `r=dg(t,y,ydot)/dy+cj*dg(t,y,ydot)/dydot` where `cj` is a real scalar.
- **list**: it must be as follows

```
list(jac,x1,x2,...)
```

where the calling sequence of the function `jac` is now

```scilab
r=jac(t,y,ydot,x1,x2,...)
```

`jac` still returns `dg/dy+cj*dg/dydot` as a function of `(t,y,ydot,cj,x1,x2,...)`.
datafit Scilab Function

datafit string: it must refer to the name of a fortran subroutine (see source code of Ex-dassl.f in routines/default).

info: list which contains 7 elements, default value is list([],0,[],[],[],0,0);
info(1): real scalar which gives the maximum time for which g is allowed to be evaluated or an empty matrix [] if no limits imposed for time.
info(2): flag which indicates if dassl returns its intermediate computed values (flag=1) or only the user specified time point values (flag=0).
info(3): 2 components vector which give the definition [ml,mu] of band matrix computed by jac:
\[ r(i - j + ml + mu + 1, j) = "dg(i)/dy(j)+cj*dg(i)/dydot(j)". \] If jac returns a full matrix set info(3)=[].
info(4): real scalar which gives the maximum step size. Set info(4)=[] if no limitation.
info(5): real scalar which gives the initial step size. Set info(5)=[] if not specified.
info(6): set info(6)=1 if the solution is known to be non negative, else set info(6)=0.
info(7): set info(7)=1 if ydot0 is just an estimation, info(7)=0 if g(t0,y0,ydot0)=0.
hd: real vector which allows to store the dassl context and to resume integration
r: real matrix. Each column is the vector \[ [t; x(t); xdot(t)] \] where t is time index for which the solution had been computed

DESCRIPTION:
Solution of the implicit differential equation
\[ g(t,y,ydot)=0 \]
\[ y(t0)=y0 \quad \text{and} \quad ydot(t0)=ydot0 \]

Detailed examples are given in SCIDIR/tests/dassldasrt.tst

EXAMPLES:

deff('[r,ires]=chemres(t,y,yd)',[
   'r(1)=-0.04*y(1)+1d4*y(2)*y(3)-yd(1);',
   'r(2)=0.04*y(1)-1d4*y(2)*y(3)-3d7*y(2)*y(2)-yd(2);',
   'r(3)=y(1)+y(2)+y(3)-1;',
   'ires=0']);
deff('[pd]=chemjac(x,y,yd,cj)',[
   'pd=[-0.04-cj , 1d4*y(3) , 1d4*y(2);',
   '0.04 , -1d4*y(3)-2*3d7*y(2)-cj , -1d4*y(2);]',
   '1,1,1])

y0=[1;0;0];
yd0=[-0.04;0.04;0];
t=[1.d-5:0.02:.4,0.41:.1:4,4:100,4000,40000,4d5,4d6,4d7,4d8,4d9,4d10];

y=dassl([y0,yd0],0,t,chemres);
info=list([],0,[],[],[],0,0);
info(2)=1;
y=dassl([y0,yd0],0,4d10,chemres,info);
y=dassl([y0,yd0],0,4d10,chemres,chemjac,info);

SEE ALSO: ode 431, dasrt 414, impl 421, fort 43, link 303, external 38

2.5.5 datafit Parameter identification based on measured data

CALLING SEQUENCE:
[p, err] = datafit([imp, ] G [,DG], Z [,W], [contr], p0, [algo], [df0, [mem]],
[work], [stop], ['in'])

PARAMETERS:
imp : scalar argument used to set the trace mode. imp=0 nothing (except errors) is reported, imp=1
initial and final reports, imp=2 adds a report per iteration, imp>2 add reports on linear search.
Warning, most of these reports are written on the Scilab standard output.
G : function descriptor (e=G(p,z), e: ne x 1, p: np x 1, z: nz x 1)
DG : partial of G wrt p function descriptor (optional; S=DG(p,z), S: ne x np)
Z : matrix [z_1,z_2,...,z_n] where z_i (nz x 1) is the ith measurement
W : weighting matrix of size ne x ne (optional; default no ponderation)
contr : 'b', binf, bsup with binf and bsup real vectors with same dimension as p0. binf
and bsup are lower and upper bounds on p.
p0 : initial guess (size np x 1)
algo : 'qn' or 'gc' or 'nd'. This string stands for quasi-Newton (default), conjugate gradient or
non-differentiable respectively. Note that 'nd' does not accept bounds on x.
df0 : real scalar. Guessed decreasing of f at first iteration. (df0=1 is the default value).
mem : integer, number of variables used to approximate the Hessian, (algo='gc' or 'nd'). Default
value is around 6.
stop : sequence of optional parameters controlling the convergence of the algorithm. stop= 'ar', nap,
[iter [,epsg [,epsf [,epsx]]]]
"ar" : reserved keyword for stopping rule selection defined as follows:
ap : maximum number of calls to fun allowed.
it : maximum number of iterations allowed.
epsg : threshold on gradient norm.
epsf : threshold controlling decreasing of f
epsx : threshold controlling variation of x. This vector (possibly matrix) of same size as x0 can be
used to scale x.
"in" : reserved keyword for initialization of parameters used when fun in given as a Fortran routine
(see below).
p : Column vector, optimal solution found
err : scalar, least square error.

DESCRIPTION:
datafit is used for fitting data to a model. For a given function G(p,z), this function finds the best
vector of parameters p for approximating G(p,z_i)=0 for a set of measurement vectors z_i
Vector p is
found by minimizing G(p,z_1)^T W G(p,z_1) + G(p,z_2)^T W G(p,z_2) + ... + G(p,z_n)^T W G(p,z_n)
datafit is an improved version of fitdat.

EXAMPLE:
deff('y=FF(x)','y=a*(x-b)+c*x.*x')
X=[]; Y=[];
a=34; b=12; c=14; for x=0:.1:3, Y=[Y,FF(x)+100*(rand()-0.5)]; X=[X,x];end
Z=[Y;X];
deff('e=G(p,z)','a=p(1),b=p(2),c=p(3),y=z(1),x=z(2),e=y-FF(x)')
[p, err] = datafit(G, Z, [3; 5; 10])
xset('window',0)
xbasc();
plot2d(X',Y',-1)
plot2d(X',FF(X')',5,'002')
a=p(1), b=p(2), c=p(3); plot2d(X', FF(X')', 12, '002')
// same problme with a known
deff('e=G(p,z,a)','b=p(1), c=p(2), y=z(1), x=z(2), e=y-FF(x)')
[p, err] = datafit(list(G, a), Z, [5; 10])

Scilab Group
Feb 1999  418
a=34; b=12; c=14;
deff('s=DG(p,z)','y=z(1), x=z(2), s=-[x-p(2),-p(1),x*x]')
[p, err] = datafit(G, DG, Z, [3; 5; 10])
xset('window', 1)
xbasc();
plot2d(X', Y', -1)
plot2d(X', FF(X)', 5, '002')
a=p(1), b=p(2), c=p(3); plot2d(X', FF(X)', 12, '002')

SEE ALSO: optim 438, leastsq 427

2.5.6 derivative- derivative

CALLING SEQUENCE:

J = derivative(f, x0, h)
[J, J2] = derivative(f, x0, h)

PARAMETERS:

f : Scilab function f: \( \mathbb{R}^n \rightarrow \mathbb{R}^p \)
x0 : real column vector (of dimension n)
h : (small) positive real number (default is 1.d-7)
J : real \( p \times n \) Jacobian matrix
J2 : real \( p \times (p^n) \) matrix

DESCRIPTION:

Approximate derivatives of a function \( f: \mathbb{R}^n \rightarrow \mathbb{R}^p \).

\[
f(x) = f(x_0) + D_1f(x_0) \cdot dx + 1/2 \cdot D_2f(x_0) \cdot (dx \cdot dx) + ...
\]

\( J = D_1f(x_0) \); \( J_2 = D_2f(x_0) \)

Derivatives are evaluated by finite difference: \( f_j(x+h)-f_j(x)/h \). Caution: h should be carefully chosen...

EXAMPLE:

deff('y=f(x)', 'y=[\sin(x(1)) \cdot \cos(x(2));x(1)^2 \cdot x(2)^2;1+x(1) \cdot x(2)^2]')
x0=[1;2]; eps=0.001; h=eps*[2;5];
[J, J2] = derivative(f, x0);
[f(x0+h)-f(x0)-J*h , 0.5*J2*(h.*h)]

SEE ALSO: mtlb_diff ??, derivat 487

2.5.7 fit_dat fit_dat Parameter identification based on measured data

CALLING SEQUENCE:

[p, err] = fit_dat(G, p0, Z [, W] [, pmin, pmax] [, DG])

PARAMETERS:

G : Scilab function (e=G(p,z), e: nExp, p: npx1, z: nzx1)
p0 : initial guess (size npx1)
Z : matrix [z_1, z_2, ..., z_n] where z_j (nzx1) is the ith measurement
**DESCRIPTION:**

`fit_dat` is used for fitting data to a model. For a given function $G(p,z)$, this function finds the best vector of parameters $p$ for approximating $G(p,z_i) = 0$ for a set of measurement vectors $z_i$. Vector $p$ is found by minimizing $\sum_{i=1}^{n} (G(p,z_i)'W(p,z_i) + G(p,z_2)'W(p,z_2) + \ldots + G(p,z_n)'W(p,z_n))$.

**EXAMPLE:**

```scilab
def('y=FF(x)','y=a*(x-b)+c*x.*x')
X=[ ]; Y=[ ];
a=34; b=12; c=14; for x=0:.1:3, Y=[Y,FF(x)+100*(rand()-.5)]; X=[X,x]; end
Z=[Y;X];
def('e=G(p,z)','a=p(1),b=p(2),c=p(3),y=z(1),x=z(2),e=y-FF(x)')
[p,err]=fit_dat(G,[3;5;10],Z)
xset('window',0)
xbasc();
plot2d(X',Y',-1)
plot2d(X',FF(X)',5,'002')
a=p(1),b=p(2),c=p(3);plot2d(X',FF(X)',12,'002')
```

**SEE ALSO:** optim 438, datafit 417

### 2.5.8 `fsolve` find a zero of a system of n nonlinear functions

**CALLING SEQUENCE:**

```
[x [,v [,info]]]=fsolve(x0,fct [,fjac] [,tol])
```

**PARAMETERS:**

- $x0$: real vector (initial value of function argument).
- `fct`: external (i.e function or list or string).
- `fjac`: external (i.e function or list or string).
- `tol`: real scalar. precision tolerance: termination occurs when the algorithm estimates that the relative error between $x$ and the solution is at most tol. (tol=1.d-10 is the default value).
- $x$: real vector (final value of function argument, estimated zero).
- `v`: real vector (value of function at $x$).
- `info`: termination indicator
  - 0: improper input parameters.
  - 1: algorithm estimates that the relative error between $x$ and the solution is at most tol.
  - 2: number of calls to fcn reached
  - 3: tol is too small. No further improvement in the approximate solution $x$ is possible.
4 : iteration is not making good progress.

DESCRIPTION :
find a zero of a system of n nonlinear functions in n variables by a modification of the powell hybrid
method. Jacobian may be provided.

0 = fct(x) w.r.t x.

fct is an "external". This external returns v=fct(x) given x.

The simplest calling sequence for fct is:

[v]=fct(x).

If fct is a character string, it refers to a C or Fortran routine which must be linked to Scilab. Fortran
calling sequence must be

fct(n,x,v,iflag)
integer n,iflag
double precision x(n),v(n)

and C Calling sequence must be

fct(int *n, double x[],double v[],int *iflag)

Incremental link is possible (help link).

jac is an "external". This external returns v=d(fct)/dx (x) given x.

The simplest calling sequence for jac is:

[v]=jac(x).

If jac is a character string, it refers to a to a C or Fortran routine which must be linked to Scilab calling
sequences are the same as those for fct. Note however that v must be a nxn array.

EXAMPLES :

// A simple example with fsolve
a=[1,7;2,8];b=[10;11];
deff('[y]=fsol1(x)', 'y=a*x+b');
deff('[y]=fsolj1(x)', 'y=a');
[xres]=fsolve([100;100],fsol1);
a*xres+b
[xres]=fsolve([100;100],fsol1,fsolj1);
a*xres+b
// See routines/default/Ex-fsolve.f
[xres]=fsolve([100;100], ’fsol1’,’fsolj1’,1.e-7);
a*xres+b

SEE ALSO: external 38 , quapro 440 , linpro 429 , optim 438

2.5.9 impl __________________________ differential algebraic equation

DESCRIPTION :

y=impl([type],y0,ydot0,t0,t [,atol, [rtol]],res,adda [,jac])

PARAMETERS :

y0, ydot0 : real vectors or matrix (initial conditions).
t0 : real scalar (initial time).
t : real vector (times at which the solution is computed).
res, adda : externals (function or character string or list).
type : string 'adams' or 'stiff'
atol, rtol : real scalar or real vector of the same size as y.
jac : external (function or character string or list).

DESCRIPTION :
Solution of the linear implicit differential equation
\[ A(t,y) \frac{dy}{dt} = g(t,y), \quad y(t_0) = y_0 \]
t_0 is the initial instant, y_0 is the vector of initial conditions Vector ydot_0 of the time derivative of y at t_0 must also be given. r The input res is an external i.e. a function with specified syntax, or the name a Fortran subroutine or a C function (character string) with specified calling sequence or a list.
If res is a function, its syntax must be as follows:
\[ r = res(t,y,ydot) \]
where t is a real scalar (time) and y and ydot are real vector (state and derivative of the state). This function must return \( r = g(t,y) - A(t,y) \cdot ydot \).
If res is a character string, it refers to the name of a Fortran subroutine or a C function. See SCIDIR/routines/default/ for an example to do that.
res can also be a list: see the help of ode.
The input adda is also an external.
If adda is a function, its syntax must be as follows:
\[ r = adda(t,y,p) \]
and it must return \( r = A(t,y) + p \) where p is a matrix to be added to \( A(t,y) \).
If adda is a character string, it refers to the name of a Fortran subroutine or a C function. See SCIDIR/routines/default/Ex-impl.f for an example to do that.
adda can also be a list: see the help of ode.
The input jac is also an external.
If jac is a function, its syntax must be as follows:
\[ j = jac(t,y,ydot) \]
and it must return the Jacobian of \( r = g(t,y) - A(t,y) \cdot ydot \) with respect to y.
If jac is a character string, it refers to the name of a Fortran subroutine or a C function. See SCIDIR/routines/default/ for an example to do that.
jac can also be a list: see the help of ode.

EXAMPLES :
y = impl([1;0;0],[-0.04;0.04;0],0,0.4,'resid','aplusp');
// Using hot restart
//[x1,w,iw]=impl([1;0;0],[-0.04;0.04;0],0,0.2,'resid','aplusp');
// hot start from previous call
//[x1]=impl([1;0;0],[-0.04;0.04;0],0.2,0.4,'resid','aplusp',w,iw);
//maxi(abs(x1-x))

SEE ALSO: dassl 416, ode 431, external 38

2.5.10 int2d __________ definite 2D integral by quadrature and cubature method

CALLING SEQUENCE :
[I,err]=int2d(X,Y,f [,params])

PARAMETERS :
Scilab Group April 1993 422
int3d Scilab Function

X : a 3 by N array containing the abscissae of the vertices of the N triangles.
Y : a 3 by N array containing the ordinates of the vertices of the N triangles.

f : external (function or list or string) defining the integrand \( f(u,v) \);
params : real vector \([tol, iclose, maxtri, mevals, iflag]\). default value is \([1.d-10, 1, 50, 4000, 1]\).
tol : the desired bound on the error. If iflag=0, tol is interpreted as a bound on the relative error; if iflag=1, the bound is on the absolute error.
iclose : an integer parameter that determines the selection of LQM0 or LQM1 methods. If iclose=1 then LQM1 is used. Any other value of iclose causes LQM0 to be used. LQM0 uses function values only at interior points of the triangle. LQM1 is usually more accurate than LQM0 but involves evaluating the integrand at more points including some on the boundary of the triangle. It will usually be better to use LQM1 unless the integrand has singularities on the boundary of the triangle.

maxtri : the maximum number of triangles in the final triangulation of the region
mevals : the maximum number of function evaluations to be allowed. This number will be effective in limiting the computation only if it is less than \( 94*\text{maxtri} \) when LQM1 is specified or \( 56*\text{maxtri} \) when LQM0 is specified.

iflag : I : the integral value
e : the estimated error

DESCRIPTION :
int2d computes the two-dimensional integral of a function \( f \) over a region consisting of \( n \) triangles. A total error estimate is obtained and compared with a tolerance - tol - that is provided as input to the subroutine. The error tolerance is treated as either relative or absolute depending on the input value of iflag. A 'Local Quadrature Module' is applied to each input triangle and estimates of the total integral and the total error are computed. The local quadrature module is either subroutine LQM0 or subroutine LQM1 and the choice between them is determined by the value of the input variable iclose.

If the total error estimate exceeds the tolerance, the triangle with the largest absolute error is divided into two triangles by a median to its longest side. The local quadrature module is then applied to each of the subtriangles to obtain new estimates of the integral and the error. This process is repeated until either (1) the error tolerance is satisfied, (2) the number of triangles generated exceeds the input parameter maxtri, (3) the number of integrand evaluations exceeds the input parameter mevals, or (4) the function senses that roundoff error is beginning to contaminate the result.

EXAMPLES :
X=[0,0;1,1;1,0];
Y=[0,0;0,1;1,1];
deff('z=f(x,y)','z=cos(x+y)')
[I,e]=int2d(X,Y,f)
// computes the integrand over the square [0 1]x[0 1]

SEE ALSO : intc 425 , intl 426 , int3d 423 , intg 425 , mesh2d 569

REFERENCES :
Fortran routine twodq, Authors: Kahaner, D.K., N.B.S., Rechard, O.W., N.B.S., Barnhill, Robert, Univ. of Utah

2.5.11 int3d __________ definite 3D integral by quadrature and cubature method

CALLING SEQUENCE :

[result, err] = int3d(X, Y, Z, f [, nf[, params]])

PARAMETERS :
X : a 4 by NUMTET array containing the abscissae of the vertices of the NUMTET tetrahedrons.
**int3d** Scilab Function

Y : a NUMTET array containing the ordinates of the vertices of the NUMTET tetrahedrons.
Z : a NUMTET array containing the third coordinates of the vertices of the NUMTET tetrahedrons.
f : external (function or list or string) defining the integrand \( f(xyz, nf) \), where \( xyz \) is the vector of a point coordinates and \( nf \) the number functions
nf : the number of function to integrate (default is 1)
params : real vector [minpts, maxpts, epsabs, epsrel]. default value is [0, 1000, 0.0, 1.d-5].
epsabs : Desired bound on the absolute error.
epsrel : Desired bound on the relative error.
minpts : Minimum number of function evaluations.
maxpts : Maximum number of function evaluations. The number of function evaluations over each subregion is 43
result : the integral value, or vector of the integral values.
err : Estimates of absolute errors.

**DESCRIPTION :**

The function calculates an approximation to a given vector of definite integrals

\[
\int_{a}^{b} \int_{c}^{d} \int_{e}^{f} (F, F', \ldots, F) \, dx(3) \, dx(2) \, dx(1),
\]

where the region of integration is a collection of NUMTET tetrahedrons and where

\[
F = F(J(1), J(2), J(3)), \quad J = 1, 2, \ldots, NUMFUN.
\]

A globally adaptive strategy is applied in order to compute approximations \( \text{result}(k) \) hopefully satisfying, for each component of \( I \), the following claim for accuracy:

\[
\text{ABS}(I(K) - \text{RESULT}(K)) \leq \text{MAX}(\text{EPSABS}, \text{EPSREL} \times \text{ABS}(\text{RESULT}(K))
\]

\( \text{int3d} \) repeatedly subdivides the tetrahedrons with greatest estimated errors and estimates the integrals and the errors over the new subtetrahedrons until the error request is met or \( \text{MAXPTS} \) function evaluations have been used.

A 43 point integration rule with all evaluation points inside the tetrahedron is applied. The rule has polynomial degree 8.

If the values of the input parameters EPSABS or EPSREL are selected great enough, an integration rule is applied over each tetrahedron and the results are added up to give the approximations \( \text{RESULT}(K) \).

No further subdivision of the tetrahedrons will then be applied.

When \( \text{int3d} \) computes estimates to a vector of integrals, all components of the vector are given the same treatment. That is, \( I(F_j) \) and \( I(F_k) \) for

\( j \) not equal to \( k \) are estimated with the same subdivision of the region of integration. For integrals with enough similarity, we may save time by applying \( \text{int3d} \) to all integrands in one call. For integrals that varies continuously as functions of some parameter, the estimates produced by \( \text{int3d} \) will also vary continuously when the same subdivision is applied to all components. This will generally not be the case when the different components of the integrals require clearly different subdivisions.

On the other hand this feature should be used with caution when the different components of the integrals require clearly different subdivisions.

**EXAMPLES :**

```scilab
X=[0;1;0;0];
Y=[0;0;1;0];
Z=[0;0;0;1];
deff('v=f(xyz,numfun)','v=exp(xyz''*xyz)')
[RESULT,ERROR]=int3d(X,Y,Z,'int3dex')
// computes the integrand exp(x*x+y*y+z*z) over the
//tetrahedron (0.,0.,0.), (1.,0.,0.), (0.,1.,0.), (0.,0.,1.)
```

Scilab Group April 1993 424
intg Scilab Function

SEE ALSO: intc 425, intl 426, int3d 423

REFERENCES:
Fortran routine dqtet.f

Authors:
Jarle Berntsen, The Computing Centre,
University of Bergen, Thormohlens gt. 55,
N-5008 Bergen, Norway
Phone.. 47-5-544055
Email.. jarle@eik.ii.uib.no

Ronald Cools, Dept. of Computer Science,
Katholieke Universiteit Leuven, Celestijnenlaan 200A,
B-3030 Heverlee, Belgium
Phone.. 32-16-201015 (3562)
Email.. ronald@cs.kuleuven.ac.be

Terje O. Espelid, Department of Informatics,
University of Bergen, Thormohlens gt. 55,
N-5008 Bergen, Norway
Phone.. 47-5-544180
Email.. terje@eik.ii.uib.no

2.5.12  intc ---------------------------------- Cauchy integral

CALLING SEQUENCE:
[y]=intc(a,b,f)

PARAMETERS:
a, b : two complex numbers
f : "external" function

DESCRIPTION:
If f is a complex-valued function, intc(a,b,f) computes the integral from a to b of f(z)dz
along the straight line a b of the complex plane.

SEE ALSO: intg 425, intl 426

AUTHOR : F. D.

2.5.13  intg ---------------------------------- definite integral

CALLING SEQUENCE:
[v, err]=intg(a,b,f [,ea [,er]])

PARAMETERS:
a, b : real numbers
f : external (function or list or string)
ea, er : real numbers
ea : absolute error required on the result. Default value: 0
er : relative error required on the result. Default value: 1.d-8

Scilab Group April 1993 425
DESCRIPTION:
\[ \text{intg}(a, b, f) \] evaluates the definite integral from \( a \) to \( b \) of \( f(t) \) \( dt \). The evaluation hopefully satisfies following claim for accuracy: \( \text{abs}(I-v) \leq \max(ea, er*abs(I)) \) where \( I \) stands for the exact value of the integral.

If \( f \) is a function its definition must be as follows \( y = f(t) \)

If \( f \) is a list the list must be as follows: \( \text{list}(f, x1, x2,...) \) where \( f \) is a function with calling sequence \( f(t, x1, x2,...) \).
If \( f \) is a string it refers to a the name of a Fortran subroutine (see source code of \( \text{fintg.f} \))

EXAMPLE:
\[
\text{deff('}[y]=f(x)\'_,'y=x*sin(30*x)/sqrt(1-((x/(2*%pi))ˆ2))')\]
\[
\text{exact}=-2.5432596188;
\text{abs(exact-intg(0,2*%pi,f))}
\]
\[
// See file routines/default/Ex-intg.f
\text{abs(exact-intg(0,2*%pi,'intgex'))}
\]

SEE ALSO: \( \text{intc 425}, \text{intl 426}, \text{inttrap 192}, \text{intsplin 191}, \text{ode 431} \)

### 2.5.14 \( \text{intl} \) \( \text{Cauchy integral} \)

CALLING SEQUENCE:

\[
[y]=\text{intl}(a,b,z0,r,f)
\]

PARAMETERS:

\( z0 \) : complex number
\( a, b \) : two real numbers
\( r \) : positive real number
\( f \) : "external" function

DESCRIPTION:

If \( f \) is a complex-valued function, \( \text{intl}(a, b, z0, r, f) \) computes the integral of \( f(z) \) \( dz \) along the curve in the complex plane defined by \( z0 + r*\exp(%i*t) \) for \( a \leq t \leq b \) (part of the circle with center \( z0 \) and radius \( r \) with phase between \( a \) and \( b \))

SEE ALSO: \( \text{intc 425} \)

AUTHOR: F. D.

### 2.5.15 \( \text{karmarkar} \) \( \text{karmarkar algorithm} \)

CALLING SEQUENCE:

\[
[x1]=\text{karmarkar}(a,b,c,x0)
\]

PARAMETERS:

\( a \) : matrix (n,p)
\( b \) : n - vector
\( c \) : p - vector
\( x0 \) : initial vector

Scilab Group April 1993 426
leastsq Scilab Function

eps : threshold (default value : 1.d-5)
gamma : descent step 0<gamma<1 , default value : 1/4
x1 : solution
crit : value of c'*x1

DESCRIPTION :
Computes x which minimizes

c'*x

under constraints:

a*x = b
x>=0

EXAMPLE :

// n=10;p=20;
// a=rand(n,p);c=rand(p,1);x0=abs(rand(p,1));b=a*x0;x1=karmarkar(a,b,c,x0);

2.5.16 leastsq _________________ Solves non-linear least squares problems

CALLING SEQUENCE :

[f,xopt]=leastsq([imp,] fun [,Dfun],x0)
[f,[xopt, [gradopt]]=leastsq(fun [,Dfun],[contr],x0,['algo'],[df0,[mem]],
, [stop],[‘in’])

PARAMETERS :

imp : scalar argument used to set the trace mode. imp=0 nothing (except errors) is reported, imp=1 initial and final reports, imp=2 adds a report per iteration, imp>2 add reports on linear search.
Warning, most of these reports are written on the Scilab standard output.
fun : external, i.e Scilab function or string (fun is the function defining the least square problem: see below.

x0 : real vector (initial value of variable to be minimized).

f : value of optimal least square value.

xopt : best value of x found.

contr : ’b’, binf,bsup with binf and bsup real vectors with same dimension as x0. binf and bsup are lower and upper bounds on x.

algo : A string with possible values :’qn’ or ’gc’ or ’nd’ . This string stands for quasi-Newton (default), conjugate gradient or non-differentiable respectively. Note that ’nd’ does not accept bounds on x).

df0 : real scalar. Guessed decreasing of f at first iteration. (df0=1 is the default value).

mem : integer, number of variables used to approximate the Hessian, (algo=’gc’ or ’nd’). Default value is around 6.

stop : sequence of optional parameters controlling the convergence of the algorithm. stop= ’ar’,nap,
, [iter [,epsg [,epsf [,epsx]]]]
"ar" : reserved keyword for stopping rule selection defined as follows:

nap : maximum number of calls to fun allowed.

iter : maximum number of iterations allowed.

epsg : threshold on gradient norm.

epsf : threshold controlling decreasing of f

epsx : threshold controlling variation of x. This vector (possibly matrix) of same size as x0 can be used to scale x.
"in" : reserved keyword for initialization of parameters used when \texttt{fun} in given as a Fortran routine (see below).

\texttt{gradopt} : gradient of \texttt{fun} at \texttt{xopt}

\textbf{DESCRIPTION :}

Non-linear optimization routine for programs without constraints or with bound constraints:

\[
\min \sum (f(x))^2 \text{ w.r.t } x.
\]

\texttt{fun} is an "external" i.e. function, or list or Fortran routine (see "external"). This external must return \((f(x))\) for a given \(x\).

If \texttt{fun} is a function, the calling sequence for \texttt{fun} must be:

\[
[f]=\texttt{fun}(x, [\text{optional parameters}]).
\]

Here, \texttt{fun} is a function from \(\mathbb{R}^n\) to \(\mathbb{R}^m\) which returns \(f(x)\), a real vector (value of function at \(x\)).

If \texttt{fun} is defined by a Fortran or C routine first argument must be a list: ist\(\texttt{fun\_name,m,...}\) If \texttt{fun\_name} is a character string, it refers to the name of the routine which must be linked to Scilab.

Here, the generic calling sequence for the Fortran subroutine is: subroutine fun\((m,n,x,td,f)\)

\(n\) is the dimension of \(x\), \(x\) is an \(n\) vector, \(td\) are working arrays which may also be used to pass parameters.

If \texttt{fun} is given as a Fortran routine, it is possible to initialize parameters or to send Scilab variables to this routine using sequence of arguments \texttt{\'td\', valtd}. Then, the Fortran function \texttt{fun\((m,n,x,f,td)\)} is evaluated with \texttt{td=valtd}. Thus, the Scilab variables \texttt{valtd} are sent to the Fortran function \texttt{fun}.

\texttt{Dfun} is an "external". This external must return a matrix \(g\) such as \((g(i,j)=dfi/dxj)\) for a given \(x\).

If \texttt{Dfun} is a function, the calling sequence for \texttt{fun} must be:

\[
[g]=\texttt{Dfun}(x, [\text{optional parameters}]).
\]

If \texttt{Dfun} is defined by a Fortran or C routine first argument must be a list: ist\(\texttt{fun\_name,m,...}\) If \texttt{fun\_name} is a character string, it refers to the name of the routine which must be linked to Scilab.

Here, the generic calling sequence for the Fortran subroutine is: subroutine dfun\((m,n,x,td,g)\)

\textbf{EXAMPLES :}

\[
a=\text{rand}(3,2);b=[1;1;1];x0=[1;-1];
deff('f=fun(x,a,b)','f=a*x-b');
deff('g=dfun(x,a,b)','g=a');
\]

\[
[f,xopt]=\text{leastsq}(\text{fun},x0) \quad //\text{Simplest call}
xopt-a\backslash b \quad //\text{compare with linear algebra solution}
\]

\[
[f,xopt]=\text{leastsq}(\text{fun},d\text{fun},x0) \quad //\text{specify gradient}
\]

\[
[f,xopt]=\text{leastsq}(\text{list}(\text{fun},[1\ 2\;3\ 4],[1;2]),x0)
deff('f=fun(x,a,b)','f=\exp(a*x)-b');
deff('g=dfun(x,a,b)','g=a.*\{(\exp(a*x)+\text{ones}(1,\text{size}(a,2)))\}');
\]

\[
[f,xopt]=\text{leastsq}(\text{list}(\text{fun},[1\ 2\;3\ 4],[1;2]),x0)
\]

\textbf{SEE ALSO :} \texttt{external 38, quapro 440, linpro 429}

\text{Scilab Group} \quad \text{April 1993}
2.5.17 linpro ............................. linear programming solver

CALLING SEQUENCE:

\[
[x, 	ext{lagr}, f] = \text{linpro}(p, C, b [, x0])
\]
\[
[x, 	ext{lagr}, f] = \text{linpro}(p, C, b, ci, cs [, x0])
\]
\[
[x, 	ext{lagr}, f] = \text{linpro}(p, C, b, ci, cs, me [, x0])
\]
\[
[x, 	ext{lagr}, f] = \text{linpro}(p, C, b, ci, cs, me, x0 [, imp])
\]

PARAMETERS:

\(p\) : real column vector (dimension \(n\))
\(C\) : real matrix (dimension \((me + md) \times n\)) (If no constraints are given, you can set \(C = [\]\))
\(b\) : RHS column vector (dimension \((me + md)\)) (If no constraints are given, you can set \(b = [\]\))
\(ci\) : column vector of lower-bounds (dimension \(n\)). If there are no lower bound constraints, put \(ci = [\]\). If some components of \(x\) are bounded from below, set the other (unconstrained) values of \(ci\) to a very large negative number (e.g. \(ci(j) = -(% \text{eps})^(-1)\)).
\(cs\) : column vector of upper-bounds. (Same remarks as above).
\(me\) : number of equality constraints (i.e. \(C(j,:) \times = b(j)\)).
\(x0\) : either an initial guess for \(x\) or one of the character strings ‘v’ or ‘g’. If \(x0=’v’\) the calculated initial feasible point is a vertex. If \(x0=’g’\) the calculated initial feasible point is arbitrary.
\(imp\) : verbose option (optional parameter) (Try \(imp=7, 8,...\)) warning the message are output in the window where scilab has been started.
\(x\) : optimal solution found.
\(f\) : optimal value of the cost function (i.e. \(f=p^*x\)).
\(lagr\) : vector of Lagrange multipliers. If lower and upper-bounds \(ci, cs\) are provided, \(lagr\) has \(n + me + md\) components and \(lagr(n+1 : n + me + md)\) is the Lagrange vector associated with the bound constraints and \(lagr(n+1 : n + me + md)\) is the Lagrange vector associated with the linear constraints. (If an upper-bound (resp. lower-bound) constraint \(i\) is active \(lagr(i)\) is \(> 0\) (resp. \(< 0\)). If no bounds are provided, \(lagr\) has only \(me + md\) components.

DESCRIPTION:

\(x, lagr, f\) = linpro \((p, C, b [, x0])\)
Minimize \(p^*x\) under the constraints \(C^*x <= b\)
\(x, lagr, f\) = linpro \((p, C, b, ci, cs [, x0])\)
Minimize \(p^*x\) under the constraints \(C^*x <= b , ci <= x <= cs\)
\(x, lagr, f\) = linpro \((p, C, b, ci, cs, me [, x0])\)
Minimize \(p^*x\) under the constraints
\(C(j,:) \times = b(j)\), \(j=1,...,me\)
\(C(j,:) \times <= b(j)\), \(j=me+1,...,me+md\)
\(ci <= x <= cs\)

If no initial point is given the program computes a feasible initial point which is a vertex of the region of feasible points if \(x0=’v’\).
If \(x0=’g’\), the program computes a feasible initial point which is not necessarily a vertex. This mode is advisable when the quadratic form is positive definite and there are a few constraints in the problem or when there are large bounds on the variables that are security bounds and very likely not active at the optimal solution.

EXAMPLE:

//Find \(x\) in \(R^6\) such that:
//\(C1^*x = b1\) (3 equality constraints i.e me=3)
\(C1= [1,-1,1,0,3,1;\)
\(-1,0,-3,-4,5,6;\)
2, 5, 3, 0, 1, 0];
b1=[1; 2; 3];
// C2*x <= b2 (2 inequality constraints)
C2=[0,1,0,1,2,-1;
   -1,0,2,1,1,0];
b2=[-1; 2.5];
// with x between ci and cs:
cl=[-1000; -10000; 0; -1000; -1000; -1000];
cs=[10000; 100; 1.5; 100; 100; 1000];
// and minimize \( p' * x \) with
p=[1; 2; 3; 4; 5; 6];
// No initial point is given: x0='v';
C=[C1; C2]; b=[b1; b2] ; me=3; x0='v';
[x, lagr, f]=linpro(p, C, b, ci, cs, me, x0)
// Lower bound constraints 3 and 4 are active and upper bound
// constraint 5 is active --> lagr(3:4) < 0 and lagr(5) > 0.
// Linear (equality) constraints 1 to 3 are active --> lagr(7:9) <> 0

SEE ALSO: quapro 440

AUTHOR: E. Casas, C. Pola Mendez

2.5.18 lmisolver ________________ linear matrix inequation solver

CALLING SEQUENCE:

[XLISTF[, OPT]] = lmisolver(XLIST0, evalfunc [,options])

PARAMETERS:

XLIST0 : a list of containing initial guess (e.g. XLIST0=list(X1, X2, ..., Xn))
evalfunc : a Scilab function ("external" function with specific syntax)
XLISTF : a list of matrices (e.g. XLIST0=list(X1, X2, ..., Xn))
options : optional parameter. If given, options is a real row vector with 5 components [Mbound, abstol, nu, maxiters, ME]

The syntax the function evalfunc must be as follows:
[LME, LMI, OBJ]=evalfunc(X) where X is a list of matrices, LME, LMI are lists and OBJ a real scalar.

DESCRIPTION:
lmisolver solves the following problem:
minimize \( f(X1, X2, ..., Xn) \) a linear function of Xi’s
under the linear constraints: \( Gi(X1, X2, ..., Xn) = 0 \) for i=1,...,p and LMI (linear matrix inequalities) constraints:
\( Hj(X1, X2, ..., Xn) > 0 \) for j=1,...,q
The functions f, G, H are coded in the Scilab function evalfunc and the set of matrices Xi’s in the list X (i.e. X=list(X1, ..., Xn)).
The function evalfunc must return in the list LME the matrices G1(X), ..., Gp(X) (i.e. LME(i)=Gi(X1, ..., Xn), i=1,...,p).
evalfunc must return in the list LMI the matrices H1(X0), ..., Hq(X) (i.e. LMI(j)=Hj(X1, ..., Xn), j=1,...,q).
evalfunc must return in OBJ the value of f(X) (i.e. OBJ=f(X1, ..., Xn)).

lmisolver returns in XLISTF, a list of real matrices, i.e. XLIST=list(X1, X2, ..., Xn) where the Xi’s solve the LMI problem:
Defining Y, Z and cost by:
[Y, Z, cost] = evalfunc(XLIST), Y is a list of zero matrices, Y=list(Y1, ..., Yp), Y1=0, Y2=0, ..., Yp=0.
Z is a list of square symmetric matrices, Z=list(Z1, ..., Zq), which are semi positive definite Z1>0, Z2>0, ..., Zq>0 (i.e. spec(Z(j)) > 0).

Scilab Group April 1993 430
ode Scilab Function

cost is minimized. lmisolver can also solve LMI problems in which the Xi’s are not matrices but lists of matrices. More details are given in the documentation of LMITOOL.

EXAMPLE :

//Find diagonal matrix X (i.e. X=diag(diag(X), p=1) such that
//A1'*X+X*A1+Q1 < 0, A2'*X+X*A2+Q2 < 0 (q=2) and trace(X) is maximized
n=2;A1=rand(n,n);A2=rand(n,n);
Xs=diag(1:n);Q1=-(A1'*Xs+Xs*A1+0.1*eye());
Q2=-(A2'*Xs+Xs*A2+0.2*eye());
deff('{[LME,LMI,OBJ]=evalf(Xlist).Face=X=Xlist(1),LME=X-diag(diag(X));...LMI=list(-(A1'*X+X*A1+Q1),-(A2'*X+X*A2+Q2)),OBJ= -sum(diag(X) )}');
X=lmisolver(list(zeros(A1)),evalf);X=X(1)

SEE ALSO: lmitool 431

2.5.19 lmitool tool for solving linear matrix inequations

CALLING SEQUENCE :

lmitool()

lmitool(filename)

txt=lmitool(probname,varlist,datalist)

PARAMETERS :

filename : a string referring to a .sci function
probname : a string containing the name of the problem
varlist : a string containing the names of the unknown matrices (separated by commas if there are more than one)
datalist : a string containing the names of data matrices (separated by commas if there are more than one)
txt : a string providing information on what the user should do next

DESCRIPTION :

lmitool() or lmitool(filename) is used to define interactively a LMI problem. In the non interactive mode, txt=lmitool(probname,varlist,datalist) generates a file in the current directory. The name of this file is obtained by adding .sci to the end of probname. This file is the skeleton of a solver function and the corresponding evaluation function needed by lmisolver.

SEE ALSO: lmisolver 430

2.5.20 ode ordinary differential equation solver

CALLING SEQUENCE :

y=ode(y0,t0,t,f)
[y,w,iw]=ode(['type'],y0,t0,t [,rtol [,atol]],f [,jac] [,w,iw])
[y,rd,w,iw]=ode("root",y0,t0,t [,rtol [,atol]],f [,jac],ng,g [,w,iw])
y=ode("discrete",y0,k0,kvect,f)
ode Scilab Function

PARAMETERS:

- **y0**: real vector or matrix (initial conditions).
- **t0**: real scalar (initial time).
- **t**: real vector (times at which the solution is computed).
- **f**: external (function or character string or list).
- **type**: one of the following character string: "adams" "stiff" "rk" "rkf" "fix" "discrete" "roots"
- **rtol, atol**: real constants or real vectors of the same size as y.
- **jac**: external (function or character string or list).
- **w, iw**: real vectors.
- **ng**: integer.
- **g**: external (function or character string or list).
- **k0**: integer (initial time). kvect : integer vector.

DESCRIPTION:

ode is the standard function for solving explicit ODE systems defined by:
\[ \frac{dy}{dt}=f(t,y) \quad y(t_0)=y_0. \]

It is an interface to various solvers, in particular to ODEPACK. The type of problem solved and the method used depend on the value of the first optional argument type which can be one of the following strings:

- **<not given>**: lsoda solver of package ODEPACK is called by default. It automatically selects between nonstiff predictor-corrector Adams method and stiff Backward Differentiation Formula (BDF) method. It uses nonstiff method initially and dynamically monitors data in order to decide which method to use.
- **"adams"**: This is for nonstiff problems. lsode solver of package ODEPACK is called and it uses the Adams method.
- **"stiff"**: This is for stiff problems. lsode solver of package ODEPACK is called and it uses the BDF method.
- **"rk"**: Adaptive Runge-Kutta of order 4 (RK4) method.
- **"rkf"**: The Shampine and Watts program based on Fehlberg’s Runge-Kutta pair of order 4 and 5 (RKF45) method is used. This is for non-stiff and mildly stiff problems when derivative evaluations are inexpensive. This method should generally not be used when the user is demanding high accuracy.
- **"fix"**: Same solver as "rkf", but the user interface is very simple, i.e. only rtol and atol parameters can be passed to the solver. This is the simplest method to try.
- **"root"**: ODE solver with rootfinding capabilities. The lsodar solver of package ODEPACK is used. It is a variant of the lsoda solver where it finds the roots of a given vector function. See help on ode_root for more details.
- **"discrete"**: Discrete time simulation. See help on ode_discrete for more details.

In this help we only describe the use of ode for standard explicit ODE systems.

The simplest call of ode is: \( y=\text{ode}(y0,t0,t,f) \) where \( y0 \) is the vector of initial conditions, \( t0 \) is the initial time, \( t \) is the vector of times at which the solution \( y \) is computed and \( y \) is the solution vector \( y=[y(t(1)), y(t(2)), \ldots] \).

The input \( f \) to \text{ode} is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list.

If \( f \) is a function, its syntax must be as follows:

\[
ydot = f(t,y)
\]

where \( t \) is a real scalar (time) and \( y \) a real vector (state). This function is the RHS of the differential equation \( dy/dt=f(t,y) \).

If \( f \) is a character string, it refers to the name of a Fortran subroutine or a C function, i.e. if \text{ode}(y0,t0,t,"fex") is the command, then the subroutine \text{fex} is called. This routine must have the following calling sequence: \( f(n,t,y,ydot) \). It can be dynamically linked to Scilab by the link function. Examples of such programs can be seen in the files SCIDIR/routines/default/README and SCIDIR/routines/default/Ex-ode.f.
ode Scilab Function

The \( f \) argument can also be a list: if \( \text{ode}(y0,t0,t,lst) \) is the command, then \( lst \) must be a list with the following structure:

\[
\text{lst} = \text{list}(f,u1,u2,...,un)
\]

where \( f \) is a function with syntax:

\[
\text{ydot} = f(t,y,u1,u2,...,un)
\]

this allows to use parameters as the arguments of \( f \).

The function \( f \) can return a \( pxq \) matrix instead of a vector. With this matrix notation, we solve the \( n=p+q \) ODE’s system \( \frac{dY}{dt}=F(t,Y) \) where \( Y \) is a \( pxq \) matrix. Then initial conditions, \( Y0 \), must also be a \( pxq \) matrix and the result of \( \text{ode} \) is the \( p \times q(T+1) \) matrix \( [Y(t_0),Y(t_1),...,Y(t_T)] \).

Optional parameters can be given for the error of the solution: \( rtol \) and \( atol \) are threshold for relative and absolute estimated errors. The estimated error on \( y(i) \) is:

\[
rtol(i) \cdot \text{abs}(y(i)) + atol(i)
\]

and integration is carried out as far as this error is small for all components of the state. If \( rtol \) and/or \( atol \) is a constant \( rtol(i) \) and/or \( atol(i) \) are set to this constant value. Default values for \( rtol \) and \( atol \) are respectively \( rtol=1.d-5 \) and \( atol=1.d-7 \) for most solvers and \( rtol=1.d-3 \) and \( atol=1.d-4 \) for "rfk" and "fix".

For stiff problems, it is better to give the Jacobian of the RHS function as the optional argument \( jac \). It is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list. If \( jac \) is a function the syntax should be as follows:

\[
J = jac(t,y)
\]

where \( t \) is a real scalar (time) and \( y \) a real vector (state). The result matrix \( J \) must evaluate to \( df/dx \) i.e. \( J(k,i) = \frac{df_k}{dx_i} \) with \( f_k = k\text{th component of } f \).

If \( jac \) is a character string it refers to the name of a Fortran subroutine or a C function, with the following calling sequence: \( jac(n,t,y,ml,mu,J,nrpd) \). In most cases you have not to refer \( ml, mu \) and \( nrpd \) (see source code in \( \text{SCIDIR/routines/default/Ex-ode.f} \) for an example).

If \( jac \) is a list the same conventions as for \( f \) apply.

Optional arguments \( w \) and \( iw \) are vectors for storing information returned by the integration routine. When these vectors are provided in RHS of \( \text{ode} \) the integration re-starts with the same parameters as in its previous stop.

More options can be given to ODEPACK solvers by using \( \text{%ODEOPTIONS} \) variable. See \( \text{odeoptions help} \).

**EXAMPLE**:

// Simple one dimension ODE
// dy/dt=y^2-y*\sin(t)+\cos(t), y(0)=0
deff("[ydot]=f(t,y)","ydot=y^2-y*\sin(t)+\cos(t)"")
y0=0; t0=0; t=0:0.1:%pi;
y=ode(y0,t0,t,f)
plot(t,y)

// Simulation of dx/dt = A x(t) + B u(t) with u(t)=\sin(\omega t),
// x0=[1;0]
// \text{solution } x(t) \text{ desired at } t=0.1, 0.2, 0.5, 1.
// \text{A and u function are passed to RHS function in a list}.
// \text{B and omega are passed as global variables}
deff("[xdot]=linear(t,x,A,u)","xdot=A*x+B*u(t)"")
deff("[ut]=u(t)","ut=\sin(\omega t)"")
A=[1 1; 0 2]; B=[1;1]; omega=5;
ode([1;0],0,[0.1,0.2,0.5,1],list(linear,A,u))

Scilab Group February 1998 433
ode_root Scilab Function

// Matrix notation
// Integration of the Riccati differential equation
// Xdot=A'*X + X*A - X'*B*X + C , X(0)=Identity
// Solution at t=[1,2]
deff("[Xdot]=ric(t,X)","Xdot=A'*X+X*A-X'*B*X+C")
A=[1,1;0,2]; B=[1,0;0,1]; C=[1,0;0,1];
t0=0; t=0:0.1:%pi;
X=ode(eye(A),0,t,ric)

// Computation of exp(A)
A=[1,1;0,2];
deff("[xdot]=f(t,x)","xdot=A*x");
ode(eye(A),0,1,f)
od("adams",eye(A),0,1,f)
// with stiff matrix, Jacobian given
A=[10,0;0,-1];
deff("[xdot]=f(t,x)","xdot=A*x");
deff("[J]=Jacobian(t,y)","J=A")
od("stiff",[0;1],0,1,f,Jacobian)

SEE ALSO: ode_discrete 434, ode_root 435, dassl 416, impl 421, odedc 436, odeoptions 437, csim 329, ltitr 347, rtitr 358

2.5.21 ode_discrete ordinary differential equation solver, discrete time simulation

CALLING SEQUENCE:

y=ode("discrete",y0,k0,kvect,f)

PARAMETERS:

y0 : real vector or matrix (initial conditions).
t0 : real scalar (initial time).
f : external i.e. function or character string or list.
k0 : integer (initial time).
kvect : integer vector.

DESCRIPTION:
With this syntax (first argument equal to "discrete") ode computes recursively y(k+1)=f(k,y(k)) from an initial state y(k0) and returns y(k) for k in kvect. kvect(1) must be greater than or equal to k0.

Other arguments and other options are the same as for ode, see the ode help.

EXAMPLE:

y1=[1;2;3]; deff("yp=a_function(k,y)","yp=A*y+B*u(k)")
A=diag([0.2,0.5,0.9]); B=[1;1;1]; u=1:10; n=5;
y=ode("discrete",y1,1,1:n,a_function);
y(:,2)-(A*y1+B*u(1))
// Now y evaluates at [y3,y5,y7,y9]
y=ode("discrete",y1,1,3:2:9,a_function)

SEE ALSO: ode 431

Scilab Group February 1998 434
ode_root ordinary differential equation solver with root finding

CALLING SEQUENCE:

\[ y, rd[, w, iw] = \text{ode}("root", y0, t0, t[, rtol [, atol]], f[, jac], ng, g[, w, iw]) \]

PARAMETERS:

- \( y0 \): real vector or matrix (initial conditions).
- \( t0 \): real scalar (initial time).
- \( t \): real vector (times at which the solution is computed).
- \( f \): external i.e. function or character string or list.
- \( rtol, atol \): real constants or real vectors of the same size as \( y \).
- \( jac \): external i.e. function or character string or list.
- \( w, iw \): real vectors.
- \( ng \): integer.
- \( g \): external i.e. function or character string or list.

DESCRIPTION:

With this syntax (first argument equal to "root") \text{ode} computes the solution of the differential equation \( \frac{dy}{dt} = f(t, y) \) until the state \( y(t) \) crosses the surface \( g(t, y) = 0 \).

\( g \) should give the equation of the surface. It is an external i.e. a function with specified syntax, or the name of a Fortran subroutine or a C function (character string) with specified calling sequence or a list.

If \( g \) is a function the syntax should be as follows:

\[ z = g(t, y) \]

where \( t \) is a real scalar (time) and \( y \) a real vector (state). It returns a vector of size \( ng \) which corresponds to the \( ng \) constraints. If \( g \) is a character string it refers to the name of a Fortran subroutine or a C function, with the following calling sequence: \( g(n, t, y, ng, gout) \) where \( ng \) is the number of constraints and \( gout \) is the value of \( g \) (output of the program). If \( g \) is a list the same conventions as for \( f \) apply (see \text{ode} help).

Output \( rd \) is a \( 1 \times k \) vector. The first entry contains the stopping time. Other entries indicate which components of \( g \) have changed sign. \( k \) larger than 2 indicates that more than one surface (\((k-1)\) surfaces) have been simultaneously traversed.

Other arguments and other options are the same as for \text{ode}, see the \text{ode} help.

EXAMPLE:

// Integration of the differential equation
// \( \frac{dy}{dt} = y \), \( y(0) = 1 \), and finds the minimum time \( t \) such that \( y(t) = 2 \)
\[
\text{deff("[ydot]=f(t,y)","ydot=y")}
\text{deff("[z]=g(t,y)","z=y-2")}
\]
\[
y0=1; ng=1;
[y, rd]=\text{ode}("roots", y0, 0, 2, f, ng, g)
\]
\[
\text{deff("[z]=g(t,y)","z=y-[2;2;33]")}
[y, rd]=\text{ode}("roots", 1, 0, 2, f, 3, g)
\]

SEE ALSO: dasrt 414, ode 431

Scilab Group April 1993 435
CALLING SEQUENCE:

\[ yt = \text{odedc}(y0, nd, stdel, t0, t, f) \]

PARAMETERS:

\( y0 \) : real column vector (initial conditions), \( y0 = [y0c; y0d] \) where \( y0d \) has \( nd \) components.
\( nd \) : integer, dimension of \( y0d \)
\( stdel \) : real vector with one or two entries, \( stdel = [h, \delta] \) (with \( \delta = 0 \) as default value).
\( t0 \) : real scalar (initial time).
\( t \) : real (row) vector, instants where \( yt \) is calculated.
\( f \) : external i.e. function or character string or list with calling sequence: \( yp = f(t, yc, yd, flag) \).

DESCRIPTION:

\[ y = \text{odedc}([y0c; y0d], nd, [h, \delta], t0, t, f) \] computes the solution of a mixed discrete/continuous system. The discrete system state \( yd_k \) is embedded into a piecewise constant \( yd(t) \) time function as follows:

\[ yd(t) = yd_k \text{ for } t \text{ in } [t_k = \delta + k \cdot h, t_{k+1} = \delta + (k+1) \cdot h] \quad (\text{with } \delta = h \cdot \delta) \]

The simulated equations are now:

\[ \frac{dyc}{dt} = f(t, yc(t), yd(t), 0), \text{ for } t \text{ in } [t_k, t_{k+1}] \]

\[ yc(t0) = y0c \]

and at instants \( t_k \) the discrete variable \( yd \) is updated by:

\[ yd(t_{k+}) = f(yc(t_{k-}), yd(t_{k-}), 1) \]

Note that, using the definition of \( yd(t) \) the last equation gives

\[ yd_k = f(t_k, yc(t_k-), yd(t_{k-1}), 1) \quad (yc \text{ is time-continuous: } yc(t_{k-}) = yc(tk)) \]

The calling parameters of \( f \) are fixed: \( ycd = f(t, yc, yd, flag) \); this function must return either the derivative of the vector \( yc \) if \( flag = 0 \) or the update of \( yd \) if \( flag = 1 \).
\( ycd = \text{dot}(yc) \) must be a vector with same dimension as \( yc \) if \( flag = 0 \) and \( ycd = \text{update}(yd) \) must be a vector with same dimension as \( yd \) if \( flag = 1 \).
\( t \) is a vector of instants where the solution \( y \) is computed.
\( y \) is the vector \( y = [y(t(1)), y(t(2)), ...] \). This function can be called with the same optional parameters as the \( \text{ode} \) function (provided \( nd \) and \( stdel \) are given in the calling sequence as second and third parameters). In particular integration flags, tolerances can be set. Optional parameters can be set by the \( \text{odeoptions} \) function.
An example for calling an external routine is given in directory \( \text{SCIDIR/default/fydot2.f} \)

External routines can be dynamically linked (see link).

EXAMPLE:

```scilab
//Linear system with switching input
def('xdu=phis(t,x,u,flag)', 'if flag==0 then xdu=A*x+B*u; else xdu=1-u;end');
x0=[1;1];A=[-1,2;-2,-1];B=[1;2];u=0;nu=1;stdel=[1,0];u0=0;t0:0.05:10;
xu=odedc([x0;u0],nu,stdel,0,phis);x=xu(1:2,:);u=xu(3,:);
x=2;
plot2d1('onn','t','x',[1:nx],'161');
plot2d2('onn','t','u',[nx+1:nx+nu],'000');
//Fortran external (see fydot2.f):
```
odeoptions Scilab Function

norm(xu-odedc([x0;u0],nu,stdel,0,t,’phis’),1)
//Sampled feedback
//
// | xcdot=fc(t,xc,u)  
// (system) | y=hc(t,xc)
//
// | xd+=fd(xd,y)     
// (feedback) | u=hd(t,xd)
//
deff(’xcd=f(t,xc,xd,iflag)’,...
// [’if iflag==0 then ’
// ’ xcd=fc(t,xc,e(t)-hd(t,xd));’
// ’else ’
// ’ xcd=fd(xd,hc(t,xc));’
// ’end’);
A=[-10,2,3;4,-10,6;7,8,-10];B=[1;1;1];C=[1,1,1];
Ad=[1/2,1;0,1/20];Bd=[1;1];Cd=[1,1,1];
deff(’st=e(t)’,’st=sin(3*t)’)
deff(’xdot=fc(t,x,u)’,’xdot=A*x+B*u’)
deff(’y=hc(t,x)’,’y=C*x’)
deff(’xp=fd(x,y)’,’xp=Ad*x + Bd*y’)
deff(’u=hd(t,x)’,’u=Cd*x’)
h=0.1;t0=0;t=0:0.1:2;
x0c=[0;0;0];x0d=[0;0];nd=2;
xcd=odedc([x0c;x0d],nd,h,t0,t,’fcd1’) // Fast calculation (see fydot2.f)
plot2d([t’,t’,t’],xcd(1:3,:));
xset("window",2);plot2d2("gnn",[t’,t’],xcd(4:5,:));
xset("window",0);

SEE ALSO:  ode 431,  odeoptions 437,  csim 329,  external 38

2.5.24  odeoptions __________________________________ set options for ode solvers

CALLING SEQUENCE:
odeoptions()

DESCRIPTION:
This function interactively displays a command which should be executed to set various options of ode solvers. The global variable %ODEOPTIONS sets the options.

CAUTION: the ode function checks if this variable exists and in this case it uses it. For using default values you should clear this variable. Note that odeoptions does not create this variable. To create it you must execute the command line displayed by odeoptions.

The variable %ODEOPTIONS is a vector with the following elements:

[itask,tcrit,h0,hmax,hmin,jactyp,mxstep,maxordn,maxords,ixpr,ml,mu]

The default value is:
[1,0,0,%inf,0,2,500,12,5,0,-1,-1]

Scilab Group  February 1998  437
The meaning of the elements is described below.

- **itask**: 1: normal computation at specified times
  2: computation at mesh points (given in first row of output of ode)
  3: one step at one internal mesh point and return
  4: normal computation without overshooting
  5: one step, without passing tcrit, and return

- **tcrit**: assumes itask equals 4 or 5, described above

- **h0**: first step tried

- **hmax**: max step size

- **hmin**: min step size

- **jactype**: 0: functional iterations, no jacobian used ("adams" or "stiff" only)
  1: user-supplied full jacobian
  2: internally generated full jacobian
  3: internally generated diagonal jacobian ("adams" or "stiff" only)
  4: user-supplied banded jacobian (see ml and mu below)
  5: internally generated banded jacobian (see ml and mu below)

- **maxordn**: maximum non-stiff order allowed, at most 12

- **maxords**: maximum stiff order allowed, at most 5

- **ixpr**: print level, 0 or 1

- **ml, mu**: If jactype equals 4 or 5, ml and mu are the lower and upper half-bandwidths of the banded jacobian: the band is the i,j's with i-ml <= j <= ny-1. If jactype equals 4 the jacobian function must return a matrix J which is ml+mu+1 x ny (where ny=dim of y in ydot=f(t,y)) such that column 1 of J is made of mu zeros followed by df1/dy1, df2/dy1, df3/dy1, ... (1+ml possibly non-zero entries) column 2 is made of mu-1 zeros followed by df1/dx2, df2/dx2, etc

**SEE ALSO**: ode 431

### 2.5.25 optim .......................... non-linear optimization routine

**CALLING SEQUENCE:**

```plaintext
[f,xopt]=optim(costf,x0)
[f,[xopt,[gradopt,[work]]]]=optim(costf,[contr],x0,['algo'],[df0,[mem]],
[work],[stop],[‘in’],[imp=iflag])
```

**PARAMETERS:**

- **costf**: external, i.e Scilab function or string (costf is the cost function: see below its calling sequence (Scilab or Fortran)).
- **x0**: value of initial vector of variable to be minimized.
- **f**: value of optimal cost (f=costf(xopt))
- **xopt**: best value of x found.
- **contr**: 'b',binf,bsup with binf and bsup real vectors with same dimension as x0. binf and bsup are lower and upper bounds on x.
- "algo": 'qn' or 'gc' or 'nd'. This string stands for quasi-Newton (default), conjugate gradient or non-differentiable respectively. Note that 'nd' does not accept bounds on x.
- **df0**: real scalar. Guessed decreasing of f at first iteration. (df0=1 is the default value).
- **mem**: integer, number of variables used to approximate the Hessian, (algo='gc' or 'nd'). Default value is around 6.
- **stop**: sequence of optional parameters controlling the convergence of the algorithm. stop= 'ar',nap,
  [iter [,epsg [,epsf [,epsx]]]]
- "ar" : reserved keyword for stopping rule selection defined as follows:
  - **nap**: maximum number of calls to costf allowed.
  - **iter**: maximum number of iterations allowed.
  - **epsg**: threshold on gradient norm.
  - **epsf**: threshold controlling decreasing of f
  - **epsx**: threshold controlling variation of x. This vector (possibly matrix) of same size as x0 can be used to scale x.

Scilab Group April 1993 438
"in" : reserved keyword for initialization of parameters used when costf in given as a Fortran routine (see below).

"imp=iflag" : named argument used to set the trace mode. iflag=0 nothing (except errors) is reported, iflag=1 initial and final reports, iflag=2 adds a report per iteration, iflag>=2 add reports on linear search. Warning, most of these reports are written on the Scilab standard output.

gradopt : gradient of costf at xopt
work : working array for hot restart for quasi-Newton method. This array is automatically initialized by optim when optim is invoked. It can be used as input parameter to speed-up the calculations.

DESCRIPTION :
Non-linear optimization routine for programs without constraints or with bound constraints:

min costf(x) w.r.t x.

  costf is an "external" i.e function, or list or Fortran routine (see "external"). This external must return f (costf(x)) and g (gradient of costf) given x.

If costf is a function, the calling sequence for costf must be:

[f, g, ind]=costf(x, ind).

  Here, costf is a function which returns f, value (real number) of cost function at x, and g, gradient vector of cost function at x. The variable ind is used by optim and is described below.

If ind=2 (resp. 3, 4), costf must provide f (resp. g, f and g).
If ind=1 nothing is computed (used for display purposes only).
On output, ind<0 means that f cannot be evaluated at x and ind=0 interrupts the optimization.
If costf is a character string, it refers to the name of a Fortran routine which must be linked to Scilab (see examples in the routines foptim.f and e.g. genros.f in the directory SCIDIR/default)
Dynamic link of Fortran routine is also possible (help link).
Here, the generic calling sequence for the Fortran subroutine is: function costf(ind, n, x, f, g, ti, tr, td)
ind has the same meaning as above if set to 0,1,2 but the values ind=10 and ind=11 are now valid. These values are used for initializations (see below).
n is the dimension of x, x is an n vector, ti, tr, td are working arrays.
The Fortran function costf must return f and the vector g, given x, ind, n, ti, tr, td.
If costf is given as a Fortran routine, it is possible to initialize parameters or to send Scilab variables to this routine.
This facility is managed by the parameter 'in'.
If the string 'in' is present, initialization is done by Fortran: optim makes two calls to the Fortran function costf, once with ind=10 and once with ind=11. In this case, for ind=10, costf must set the dimensions nti, ntr, ntd of ti, tr, td in the common/nird/nti, ntr, ntd and, for ind=11, costf must initialize the vectors ti, tr, td (integer vector, real vector, double precision vector respectively).
In the calling sequence of optim, the string 'in' can be replaced by 'ti', valti, 'td', valtd. Then, the Fortran function costf(ind, x, f, g, ti, tr, td) is evaluated with ti=valti and td=valtd whatever the value of ind. Thus, the Scilab variables valti and valtd (integer vector and real vector) are sent to the Fortran function costf.
It is also possible to save the content of the working arrays ti and td. This is possible by adding the strings 'si' and/or 'sd' at the ned of the calling sequence of optim. Then, the output variables must be: [f, [x, [g]], [to]], [ti], [td]].

EXAMPLES :

xref=[1; 2; 3]; x0=[1; -1; 1];
deff('[f, g, ind]=cost(x, ind)', 'f=0.5*norm(x-xref)^2, g=x-xref');
[f, xopt]=optim(cost, x0); //Simplest call
[f, xopt, gopt]=optim(cost, x0, 'gc'); // By conjugate gradient
[f, xopt, gopt]=optim(cost, x0, 'nd'); // Seen as non differentiable
[f, xopt, gopt]=optim(cost, 'b', [-1; 0; 2], [0.5; 1; 4], x0); // Bounds on x
[f, xopt, gopt]=optim(cost, 'b', [-1; 0; 2], [0.5; 1; 4], x0, 'gc'); // Bounds on x
quapro Scilab Function

\[ [f, xopt, gopt] = \text{optim}(\text{cost}', 'b', [-1; 0; 2], [0.5; 1; 4], x0, 'qc', 'ar', 3) \]
// Here, 3 calls to cost are allowed.
// Now calling the Fortran subroutine "genros" in SCIDIR/default/Ex-optim.f
// See also the link function for dynamically linking an objective function
\[ [f, xopt, gopt] = \text{optim}('genros', [1; 2; 3]) \]  // Rosenbrock’s function

SEE ALSO:  external 38, quapro 440, linpro 429, datafit 417, leastsq 427

2.5.26 quapro __________________________ linear quadratic programming solver

CALLING SEQUENCE:

\[ [x, lagr, f] = \text{quapro}(Q, p, C, b [, x0]) \]
\[ [x, lagr, f] = \text{quapro}(Q, p, C, b, ci, cs [, x0]) \]
\[ [x, lagr, f] = \text{quapro}(Q, p, C, b, ci, cs, me [, x0]) \]
\[ [x, lagr, f] = \text{quapro}(Q, p, C, b, ci, cs, me, x0 [, imp]) \]

PARAMETERS:

- \( Q \): real symmetric matrix (dimension \( n \times n \)).
- \( p \): real (column) vector (dimension \( n \)).
- \( C \): real matrix (dimension \( (me + md) \times n \)) (If no constraints are given, you can set \( C = [] \)).
- \( b \): RHS column vector (dimension \( (me + md) \)) (If no constraints are given, you can set \( b = [] \)).
- \( ci \): column vector of lower-bounds (dimension \( n \)). If there are no lower bound constraints, put \( ci = [] \). If some components of \( x \) are bounded from below, set the other (unconstrained) values of \( ci \) to a very large negative number (e.g. \( ci(j) = -(% \text{eps})^(-1) \)).
- \( cs \): column vector of upper-bounds. (Same remarks as above).
- \( me \): number of equality constraints (i.e. \( C(1:me,:) \times x = b(1:me) \)).
- \( x0 \): either an initial guess for \( x \) or one of the character strings ‘v’ or ‘g’. If \( x0='v' \) the calculated initial feasible point is a vertex. If \( x0='g' \) the calculated initial feasible point is arbitrary.
- \( imp \): verbose option (optional parameter) (Try \( imp=7, 8, \ldots \)). warning the message are output in the window where scilab has been started.
- \( x \): optimal solution found.
- \( f \): optimal value of the cost function (i.e. \( f=0.5*x'*Q*x+p' \)).
- \( lagr \): vector of Lagrange multipliers. If lower and upper-bounds \( ci, cs \) are provided, \( lagr \) has \( n + me + md \) components and \( lagr(1:n) \) is the Lagrange vector associated with the bound constraints and \( lagr(n+1:n+me+md) \) is the Lagrange vector associated with the linear constraints. (If an upper-bound (resp. lower-bound) constraint \( i \) is active \( lagr(i) \) is \( >0 \) (resp. \( <0 \)). If no bounds are provided, \( lagr \) has only \( me + md \) components.

DESCRIPTION:

\[ [x, lagr, f] = \text{quapro}(Q, p, C, b [, x0]) \]
Minimize \( 0.5*x'*Q*x + p'*x \)
under the constraint
\[ C*x = b \]
\[ [x, lagr, f] = \text{quapro}(Q, p, C, b, ci, cs [, x0]) \]
Minimize \( 0.5*x'*Q*x + p'*x \)
under the constraints
\[ C*x = b \quad ci <= x <= cs \]
\[ [x, lagr, f] = \text{quapro}(Q, p, C, b, ci, cs, me [, x0]) \]
Minimize \( 0.5*x'*Q*x + p'*x \)
under the constraints

Scilab Group  April 1993 440
semidef Scilab Function

\[
C(j,:) \times = b(j), \quad j=1,\ldots,me
\]
\[
C(j,:) \times \leq b(j), \quad j=me+1,\ldots,me+md
\]
\[
ci \leq x \leq cs
\]

If no initial point is given the program computes a feasible initial point which is a vertex of the region of feasible points if \( x_0='v' \).

If \( x_0='g' \), the program computes a feasible initial point which is not necessarily a vertex. This mode is advisable when the quadratic form is positive definite and there are few constraints in the problem or when there are large bounds on the variables that are just security bounds and very likely not active at the optimal solution.

Note that \( Q \) is not necessarily non-negative, i.e. \( Q \) may have negative eigenvalues.

**EXAMPLE :**

```plaintext
//Find \( x \) in \( \mathbb{R}^6 \) such that:
//\( C1\times x = b1 \) (3 equality constraints i.e me=3)
C1 = [1,-1,1,0,3,1;
     -1,0,-3,-4,5,6;
     2,5,3,0,1,0];
b1 = [1;2;3];
//\( C2\times x \leq b2 \) (2 inequality constraints)
C2 = [0,1,0,1,2,-1;
     -1,0,2,1,1,0];
b2 = [-1;2.5];
//with \( x \) between \( ci \) and \( cs \):
\( ci = [-10000;-10000;0;-1000;-1000;0]; cs = [10000;100;1.5;100;100;1000];
//and minimize \( 0.5\times x'^*Q\times x + p'^*x \) with
p = [1;2;3;4;5;6]; Q = eye(6,6);
//No initial point is given;
C = [C1; C2];
b = [b1; b2];
me = 3;
[x, lagr, f] = quapro(Q, p, C, b, ci, cs, me)
//Only linear constraints (1 to 4) are active (lagr(1:6)=0):
[x, lagr, f] = quapro(Q, p, C, b, [], [], me)  //Same result as above
```

**SEE ALSO :** linpro 429, optim 438

 AUTHOR : E. Casas, C. Pola Mendez

### 2.5.27 **semidef**

**semidefinite programming**

**CALLING SEQUENCE :**

```plaintext
[x, Z, ul, info] = semidef(x0, Z0, F, blck_szs, c, options)
```

**PARAMETERS :**

- \( x0 : m \times 1 \) real column vector (must be strictly primal feasible, see below)
- \( Z0 : L \times 1 \) real vector (compressed form of a strictly feasible dual matrix, see below)
- \( F : L \times (m+1) \) real matrix
- \( blck\_szs : p \times 2 \) integer matrix (sizes of the blocks) defining the dimensions of the (square) diagonal blocks size(\( F(j) = blck\_szs(j) \)) \( j=1,\ldots,m+1. \)
- \( c : m \times 1 \) real vector
- \( options : \) row vector with five entries \([nu, abstol, reltol, 0, maxiters]\)
- \( ul : \) row vector with two entries
DESCRIPTION:
\[ [x, Z, u, \text{info}] = \text{semidef}(x_0, Z_0, F, \text{blckszs}, c, \text{options}) \]
solves semidefinite program:

\[
\begin{align*}
\text{minimize} & \quad c'x \\
\text{subject to} & \quad F_0 + x_1 F_1 + \cdots + x_m F_m \geq 0
\end{align*}
\]

and its dual

\[
\begin{align*}
\text{maximize} & \quad -\text{Tr} F_0 Z \\
\text{subject to} & \quad \text{Tr} F_i Z = c_i, \ i = 1, \ldots, m \\
& \quad Z \succeq 0
\end{align*}
\]

exploiting block structure in the matrices \( F_j \).

It interfaces L. Vandenberghe and S. Boyd's cvx program.

The \( F_j \)'s matrices are stored columnwise in \( F \) in compressed format: if \( F_j^{i,j} \), \( i = 0, \ldots, m \), \( j = 1, \ldots, L \) denote the \( j \)th (symmetric) diagonal block of \( F_j \), then

\[
F = \begin{bmatrix}
\text{pack}(F_0^{1,1}) & \text{pack}(F_1^{1,1}) & \cdots & \text{pack}(F_m^{1,1}) \\
\text{pack}(F_0^{1,2}) & \text{pack}(F_1^{1,2}) & \cdots & \text{pack}(F_m^{1,2}) \\
\cdots & \cdots & \cdots & \cdots \\
\text{pack}(F_0^{L,1}) & \text{pack}(F_1^{L,1}) & \cdots & \text{pack}(F_m^{L,1})
\end{bmatrix}
\]

where \( \text{pack}(M) \), for symmetric \( M \), is the vector \( [M(1,1); M(1,2); \ldots; M(1,n); M(2,2); M(2,3); \ldots; M(2,n); \ldots] \) (obtained by scanning columnwise the lower triangular part of \( M \)).

\( \text{blckszs} \) gives the size of block \( j \), i.e., \( \text{size}(F_j^{i,j}) = \text{blckszs}(j) \).

\( Z \) is a block diagonal matrix with \( L \) blocks \( Z^0, \ldots, Z^{L-1} \). \( Z^j \) has size \( \text{blckszs}[j] \) times \( \text{blckszs}[j] \). Every block is stored using packed storage of the lower triangular part.

The \( 2 \) vector \( u \) contains the primal objective value \( c'x \) and the dual objective value \( -\text{Tr} F_0 Z \).

The entries of \( \text{options} \) are respectively:

- \( \text{nu} \) = a real parameter which controls the rate of convergence.
- \( \text{abstol} \) = absolute tolerance.
- \( \text{reltol} \) = relative tolerance (has a special meaning when negative).
- \( \text{tv} \) = target value, only referenced if \( \text{reltol} < 0 \).
- \( \text{iters} \) = on entry: maximum number of iterations \( \geq 0 \), on exit: the number of iterations taken.

\( \text{info} \) returns 1 if maxiters exceeded, 2 if absolute accuracy is reached, 3 if relative accuracy is reached, 4 if target value is reached, 5 if target value is not achievable; negative values indicate errors.

Convergence criterion:

1. \( \text{maxiters} \) is exceeded
2. duality gap is less than \( \text{abstol} \)
3. primal and dual objective are both positive and duality gap is less than \( \text{reltol} \times \text{dual objective} \)
   or primal and dual objective are both negative and duality gap is less than \( \text{reltol} \times \text{minus the primal objective} \)
4. \( \text{reltol} \) is negative and primal objective is less than \( \text{tv} \) or dual objective is greater than \( \text{tv} \)

EXAMPLE:

\[
F_0 = \begin{bmatrix}
2, 1, 0, 0; \\
1, 2, 0, 0; \\
0, 0, 3, 1 \\
0, 0, 1, 3
\end{bmatrix}
\]

\[
F_1 = \begin{bmatrix}
1, 2, 0, 0; \\
2, 1, 0, 0; \\
0, 0, 1, 3
\end{bmatrix}
\]

Scilab Group
April 1993
F2 = [2, 2, 0, 0; 2, 2, 0, 0; 0, 0, 3, 4; 0, 0, 4, 4];
blck_szs = [2, 2];
F01 = F0(1:2, 1:2); F02 = F0(3:4, 3:4);
F11 = F1(1:2, 1:2); F12 = F1(3:4, 3:4);
F21 = F2(1:2, 1:2); F22 = F2(3:4, 3:4);
x0 = [0; 0]
Z0 = 2 * F0;
Z01 = Z0(1:2, 1:2); Z02 = Z0(3:4, 3:4);
FF = [[F01(:); F02(:)], [F11(:); F12(:)], [F21(:); F22(:)]]
ZZ0 = [[Z01(:); Z02(:)]]
c = [trace(F1 * Z0); trace(F2 * Z0)];
options = [10, 1.d-10, 1.d-10, 0, 50];
[x, Z, ul, info] = semidef(x0, pack(ZZ0), pack(FF), blck_szs, c, options)
w = vec2list(unpack(Z, blck_szs), [blck_szs; blck_szs]); Z = sysdiag(w(1), w(2))
c' * x + trace(F0 * Z)
spec(F0 + F1 * x(1) + F2 * x(2))
trace(F1 * Z) - c(1)
trace(F2 * Z) - c(2)
2.6 Signal Processing toolbox
2.6.1  %asn ______________________________ elliptic integral

CALLING SEQUENCE:
[y]=%asn(x,m)

PARAMETERS:
x : upper limit of integral (x>0) (can be a vector)
m : parameter of integral (0<m<1)
y : value of the integral

DESCRIPTION:
Calculates the elliptic integral

\[ K = \int_0^x \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}} \]

If x is a vector, y is a vector of same dimension as x.

EXAMPLE:
m=0.8;z=%asn(1/sqrt(m),m);K=real(z);Ktilde=imag(z);
x2max=1/sqrt(m);
x1=0:0.05:1;x2=1:((x2max-1)/20):x2max;x3=x2max:0.05:10;
x=[x1,x2,x3];
y=%asn(x,m);
rect=[0,-Ktilde,1.1*K,2*Ktilde];
plot2d(real(y)',imag(y)',1,'011',' ',rect)
//
deff('y=f(t)','y=1/sqrt((1-t^2)*(1-m*t^2))');
intg(0,0.9,f)-%asn(0.9,m)  //Works for real case only!

AUTHOR: F.D.

2.6.2  %k ________________________________ Jacobi’s complete elliptic integral

CALLING SEQUENCE:
[K]=%k(m)

PARAMETERS:
m : parameter of the elliptic integral 0<m<1 (m can be a vector)
K : value of the elliptic integral from 0 to 1 on the real axis

DESCRIPTION:
Calculates Jacobi’s complete elliptic integral of the first kind:

\[ K = \int_0^1 \frac{dt}{\sqrt{(1-t^2)(1-mt^2)}} \]

EXAMPLE:
m=0.4;
%asn(1,m)
%k(m)

REFERENCES:
Abramowitz and Stegun page 598
SEE ALSO:  %asn 445

AUTHOR: F.D.
2.6.3 %sn  
Jacobi's elliptic function

CALLING SEQUENCE:

\[ y = \text{sn}(x, m) \]

PARAMETERS:
- \( x \): a point inside the fundamental rectangle defined by the elliptic integral; \( x \) is a vector of complex numbers
- \( m \): parameter of the elliptic integral (0 < \( m \) < 1)
- \( y \): result

DESCRIPTION:
Jacobi's sn elliptic function with parameter \( m \): the inverse of the elliptic integral for the parameter \( m \).
The amplitude am is computed in fortran and the addition formulas for elliptic functions are applied

EXAMPLE:

```plaintext
m=0.36;
K=%k(m);
P=4*K;  //Real period
real_val=0:(P/50):P;
plot(real_val,real(%sn(real_val,m)));
xbasc();
KK=%k(1-m);
Ip=2*KK;
ima_val1=0:(Ip/50):KK-0.001;
ima_val2=(KK+0.05):(Ip/25):(Ip+KK);
z1=%sn(%i*ima_val1,m);z2=%sn(%i*ima_val2,m);
plot2d([ima_val1’,ima_val2’],[imag(z1)’,imag(z2)’]);
xgrid(3)
```

SEE ALSO: %asn 445, %k 445

AUTHOR: F. D.

2.6.4 analpf  
create analog low-pass filter

CALLING SEQUENCE:

\[ [hs,pols,zers,gain]=\text{analpf}(n,fdesign,rp,omega) \]

PARAMETERS:
- \( n \): positive integer: filter order
- \( fdesign \): string: filter design method: 'butt' or 'cheb1' or 'cheb2' or 'ellip'
- \( rp \): 2-vector of error values for cheb1, cheb2 and ellip filters where only \( rp(1) \) is used for cheb1 case, only \( rp(2) \) is used for cheb2 case, and \( rp(1) \) and \( rp(2) \) are both used for ellip case.
  - for cheb1 filters \( 1-rp(1)<\text{ripple}<1 \) in passband
  - for cheb2 filters \( 0<\text{ripple}<rp(2) \) in stopband
  - for ellip filters \( 1-rp(1)<\text{ripple}<1 \) in passband \( 0<\text{ripple}<rp(2) \) in stopband
- \( omega \): cut-off frequency of low-pass filter in Hertz
- \( hs \): rational polynomial transfer function
- \( pols \): poles of transfer function
zeros: zeros of transfer function

gain: gain of transfer function

DESCRIPTION:
Creates analog low-pass filter with cut-off frequency at omega.
hs=gain*poly(zers, 's')/poly(pols, 's')

EXAMPLE:

//Evaluate magnitude response of continuous-time system
hs=analpf(4, 'cheb1', [.1 0], 5)
fr=0:.1:15;
hf=freq(hs(2), hs(3), %i*fr);
hm=abs(hf);
plot(fr, hm)

AUTHOR: C. B.

2.6.5 buttmag .......................................................... response of Butterworth filter

CALLING SEQUENCE:

[h]=buttmag(order, omegac, sample)

PARAMETERS:

order : integer: filter order
omegac : real: cut-off frequency in Hertz
sample : vector of frequency where buttmag is evaluated
h : Butterworth filter values at sample points

DESCRIPTION:
squared magnitude response of a Butterworth filter omegac = cutoff frequency; sample = sample of frequencies

EXAMPLE:

//squared magnitude response of Butterworth filter
h=buttmag(13, 300, 1:1000);
mag=20*log(h)'/log(10);
plot2d((1:1000)', mag, [2], "011", "", [0, -180, 1000, 20])

AUTHOR: F. D.

2.6.6 casc .............................................................. cascade realization of filter from coefficients

CALLING SEQUENCE:

[cels]=casc(x, z)

PARAMETERS:

x : (4xN)-matrix where each column is a cascade element, the first two column entries being the numerator coefficients and the second two column entries being the denominator coefficients
z : string representing the cascade variable
cels : resulting cascade representation

Scilab Group April 1993 447
DESCRIPTION:
Creates cascade realization of filter from a matrix of coefficients (utility function).

EXAMPLE:
\[
x=[1,2,3;4,5,6;7,8,9;10,11,12] \
cels=casc(x,'z')
\]

2.6.7 cepstrum

calculation

CALLING SEQUENCE:
fresp = cepstrum(w,mag)

PARAMETERS:
\[w\] : positive real vector of frequencies (rad/sec)
\[mag\] : real vector of magnitudes (same size as \[w\])
\[fresp\] : complex vector

DESCRIPTION:
\[fresp = cepstrum(w,mag)\] returns a frequency response \[fresp(i)\] whose magnitude at frequency \[w(i)\] equals \[mag(i)\] and such that the phase of freq corresponds to a stable and minimum phase system. \[w\] needs not to be sorted, but minimal entry should not be close to zero and all the entries of \[w\] should be different.

EXAMPLE:
\[
w=0.1:0.1:5;mag=1+abs(sin(w)); \\
fresp=cepstrum(w,mag); \\
plot2d([w',w'],[mag(:),abs(fresp)])
\]

SEE ALSO: frfit 459

2.6.8 cheb1mag

response of Chebyshev type 1 filter

CALLING SEQUENCE:
\[h2]=cheb1mag(n,omegac,epsilon,sample)\]

PARAMETERS:
\[n\] : integer : filter order
\[omegac\] : real : cut-off frequency
\[epsilon\] : real : ripple in pass band
\[sample\] : vector of frequencies where \[cheb1mag\] is evaluated
\[h2\] : Chebyshev I filter values at sample points

DESCRIPTION:
Square magnitude response of a type 1 Chebyshev filter.
\[omegac=passband edge.\]
\[epsilon\] : such that \[1/(1+epsilon^2)=passband ripple.\]
\[sample\] : vector of frequencies where the square magnitude is desired.

EXAMPLE:
\//Chebyshev; ripple in the passband \\
\[n=13;epsilon=0.2;omegac=3;sample=0:0.05:10; \\
h=cheb1mag(n,omegac,epsilon,sample); \\
plot(sample,h,['frequencies','magnitude'])\]

SEE ALSO: buttmag 447
### 2.6.9 cheb2mag  
-------

**response of type 2 Chebyshev filter**

**CALLING SEQUENCE:**

\[
[h2]=\text{cheb2mag}(n,\text{omegar},A,\text{sample})
\]

**PARAMETERS:**

- \(n\): integer; filter order
- \(\text{omegar}\): real scalar; cut-off frequency
- \(A\): attenuation in stop band
- \(\text{sample}\): vector of frequencies where cheb2mag is evaluated
- \(h2\): vector of Chebyshev II filter values at sample points

**DESCRIPTION:**

Square magnitude response of a type 2 Chebyshev filter. 
\(\text{omegar}\) = stopband edge, \(\text{sample}\) = vector of frequencies where the square magnitude \(h2\) is desired.

**EXAMPLE:**

```plaintext
//Chebyshev; ripple in the stopband
n=10;\text{omegar}=6;A=1/0.2;\text{sample}=0.0001:0.05:10;
\text{h2}=\text{cheb2mag}(n,\text{omegar},A,\text{sample});
\text{plot}(\text{sample},\log(\text{h2})/\log(10),'frequencies','magnitude in dB')
//Plotting of frequency edges
\text{minval}=(-\text{maxi}(-\log(\text{h2})))\text{/log(10)};
\text{plot2d([\text{omegar};\text{omegar}],[\text{minval};0],[2],"000");}
//Computation of the attenuation in dB at the stopband edge
\text{attenuation}=\log(\text{A*A})/\log(10);
\text{plot2d}(\text{sample}',\text{attenuation*ones(}\text{sample}')',[5],"000")
```

**SEE ALSO:** cheb1mag 448

### 2.6.10 chepol  
-------

**Chebychev polynomial**

**CALLING SEQUENCE:**

\[
[Tn]=\text{chepol}(n,\text{var})
\]

**PARAMETERS:**

- \(n\): integer; polynomial order
- \(\text{var}\): string; polynomial variable
- \(Tn\): polynomial in the variable \(\text{var}\)

**DESCRIPTION:**

Recursive implementation of Chebyshev polynomial. 
\(Tn=2*poly(0,\text{var})*\text{chepol}(n-1,\text{var})-\text{chepol}(n-2,\text{var})\)
with \(T0=1\) and \(T1=poly(0,\text{var})\).

**EXAMPLE:**

```plaintext
\text{chepol}(4,'x')
```

**AUTHOR:** F. D.

Scilab Group  
April 1993  
449
### 2.6.11 convol

**Calling Sequence:**

```
[y] = convol(h, x)
[y, e1] = convol(h, x, e0)
```

**Parameters:**

- `x, h`: input sequences (h is a "short" sequence, x a "long" one)
- `e0`: old tail to overlap add (not used in first call)
- `y`: output of convolution
- `e1`: new tail to overlap add (not used in last call)

**Description:**

Calculates the convolution `y = h*x` of two discrete sequences by using the fft. Overlap add method can be used.

**Use of Overlap Add Method:** For `x=[x1,x2,...,xNm1,xN]` First call is `[y1,e1]=convol(h,x1);` Subsequent calls: `[yk,ek]=convol(h,xk,ekm1);` Final call: `[yN]=convol(h,xN,ekm1);` Finally `y=[y1,y2,...,yNm1,yN]`

**Example:**

```plaintext
x = [1, 2, 3];
h1 = [1, 0, 0, 0, 0]; h2 = [0, 1, 0, 0, 0]; h3 = [0, 0, 1, 0, 0];
x1 = convol(h1, x), x2 = convol(h2, x), x3 = convol(h3, x),
convol(h1+h2+h3, x)
p1 = poly(x, 'x', 'coeff')
p2 = poly(h1+h2+h3, 'x', 'coeff')
p1*p2
```

**See Also:** corr 450, fft 457, pspect 472

**Author:** F. D. C. Bunks Date 3 Oct. 1988

### 2.6.12 corr

**Calling Sequence:**

```
[cov, Mean] = corr(x, [y], nlags)
[cov, Mean] = corr('fft', xmacro, [ymacro], n, sect)
```

**Parameters:**

- `x`: a real vector
- `y`: a real vector, default value x.
- `nlags`: integer, number of correlation coefficients desired.
- `xmacro`: a scilab external (see below).
- `ymacro`: a scilab external (see below), default value xmacro
- `n`: an integer, total size of the sequence (see below).
- `sect`: size of sections of the sequence (see below).

**See Also:** corr 450, fft 457, pspect 472

**Author:** F. D. C. Bunks Date 3 Oct. 1988

Scilab Group April 1993 450
xi : a real vector
yi : a real vector, default value xi.
cov : real vector, the correlation coefficients
Mean : real number or vector, the mean of x and if given y

DESCRIPTION :
Computes

\[ \text{cov}(m) = \frac{1}{n} \sum_{k=1}^{n-m} (x(k) - \text{xmean}) (y(k+m) - \text{ymean}) \]

for \( m=0,..,nlag-1 \) and two vectors \( x=[x(1),..,x(n)] \) \( y=[y(1),..,y(n)] \)

Note that if x and y sequences are differents \( \text{corr}(x,y,...) \) is different with \( \text{corr}(y,x,...) \)

Short sequences:
\[ [\text{cov}, \text{Mean}] = \text{corr}(x, [y], nlags) \]
returns the first nlags correlation coefficients and Mean = \( \text{mean}(x) \) (mean of \( [x,y] \) if y is an argument). The sequence x (resp. y) is assumed real, and x and y are of same dimension n.

Long sequences:
\[ [\text{cov}, \text{Mean}] = \text{corr}('fft', xmacro, [ymacro], n, sect) \]
Here xmacro is either
- a function of type \( [xx]=xmacro(sect, istart) \) which returns a vector xx of dimension nsect containing the part of the sequence with indices from istart to istart+sect-1.
- a fortran subroutine which performs the same calculation. (See the source code of dgetx for an example). n = total size of the sequence. sect = size of sections of the sequence. sect must be a power of 2. cov has dimension sect. Calculation is performed by FFT.

"Updating method":
\[ \begin{align*}
[w, xu] &= \text{corr('updt', x1, [y1], w0)} \\
[w, xu] &= \text{corr('updt', x2, [y2], w, xu)} \\
&\ldots \\
w &= \text{corr('updt', xk, [yk], w, xu)}
\end{align*} \]

With this calling sequence the calculation is updated at each call to corr.

\( w0 = 0 \cdot \text{ones}(1, 2 \cdot nlag); \)
\( nlag = \text{power of 2}. \)

\( x1, x2, \ldots \) are parts of x such that \( x=[x1, x2, \ldots] \) and sizes of xi a power of 2. To get nlag coefficients a final fft must be performed \( c = \text{ffft}(w, 1)/n; \text{cov} = c(1:nlag) \) (n is the size of x (y)). Caution: this calling sequence assumes that \( \text{xmean} = \text{ymean} = 0 \).

EXAMPLE :
\[ x = \pi/10: \pi/10: 102.4*\pi; \]
\[ \text{rand('seed')}; \text{rand('normal')}; \]
\[ y = [.8*\sin(x) + .8*\sin(2*x) + \text{rand(x)}; .8*\sin(x) + .8*\sin(1.99*x) + \text{rand(x)}]; \]
\[ c = []; \]
\[ \text{for } j=1:2, \text{for } k=1:2, c = [c; \text{corr(y(k,:), y(j,:), 64)}]; \text{end}; \text{end}; \]
\[ c = \text{matrix}(c, 2, 128); \text{cov} = []; \]
for j=1:64, cov=[cov; c(:,(j-1)*2+1:2*j)]; end;
rand('unif')
//
rand('normal'); x=rand(1,256); y=-x;
deff('z=xx(inc,is)','z=x(is:is+inc-1)');
deff('z=yy(inc,is)','z=y(is:is+inc-1)');
[c,mxy]=corr(x,y,32);
x=x-mxy(1)*ones(x); y=y-mxy(2)*ones(y); // centring
c1=corr(x,y,32); c2=corr(x,32);
norm(c1+c2,1)
[c3,m3]=corr('fft', xx, yy, 256, 32);
norm(c1-c3,1)
[c4,m4]=corr('fft', xx, 256, 32);
norm(m3,1), norm(m4,1)
norm(c3-c1,1), norm(c4-c2,1)
x1=x(1:128); x2=x(129:256);
y1=y(1:128); y2=y(129:256);
w0=0*ones(1:64); // 32 coeffs
[w1,xu]=corr('u', x1, y1, w0); w2=corr('u', x2, y2, w1, xu);
z=real(fft(w2,1))/256; c5=z(1:32);
norm(c5-c1,1)
[w1,xu]=corr('u', x1, w0); w2=corr('u', x2, w1, xu);
z=real(fft(w2,1))/256; c6=z(1:32);
norm(c6-c2,1)
rand('unif')
// test for Fortran or C external
//
deff('y=xmacro(sec, ist)', 'y=sin(ist:(ist+sec-1))');
x=xmacro(100,1);
[c1,m1]=corr(x, 2^3);
[cc,mm]=corr('fft', xmacro, 100, 2^3);
[cc2,mm2]=corr('fft', 'corexx', 100, 2^3);
[maxi(abs(cc-cc1)), maxi(abs(mm-mm1)), maxi(abs(cc-cc2)), maxi(abs(mm-mm2))]
deff('y=ymacro(sec, ist)', 'y=cos(ist:(ist+sec-1))');
y=ymacro(100,1);
[c1,m1]=corr(x, y, 2^3);
[cc,mm]=corr('fft', xmacro, ymacro, 100, 2^3);
[cc2,mm2]=corr('fft', 'corexy', 'corey', 100, 2^3);
[maxi(abs(cc-cc1)), maxi(abs(mm-mm1)), maxi(abs(cc-cc2)), maxi(abs(mm-mm2))]

SEE ALSO: fft 457

2.6.13 cspect __________ spectral estimation (correlation method)

CALLING SEQUENCE:

[sm, cwp]=cspect(nlags, ntp, wtype, x, y, wpar)

PARAMETERS:

x : data if vector, amount of input data if scalar
y : data if vector, amount of input data if scalar

Scilab Group April 1993
DESCRIPTION:
Spectral estimation using the correlation method. Cross-spectral estimate of \( x \) and \( y \) is calculated when both \( x \) and \( y \) are given. Auto-spectral estimate of \( x \) is calculated if \( y \) is not given.

EXAMPLE:
```scilab
czt Scilab Function
nlags : number of correlation lags (positive integer)
npt : number of transform points (positive integer)
wttype : string : 're','tr','hm','hn','kr','ch' (window type)
wpar : optional window parameters for wttype='kr', wpar>0 and for wttype='ch', 0 < wpar(1)
        < .5, wpar(2) > 0
sm : power spectral estimate in the interval [0,1]
cwp : calculated value of unspecified Chebyshev window parameter

DESCRIPTION:
Spectral estimation using the correlation method. Cross-spectral estimate of \( x \) and \( y \) is calculated when both \( x \) and \( y \) are given. Auto-spectral estimate of \( x \) is calculated if \( y \) is not given.

EXAMPLE:
```scilab
czt Scilab Function
```
ell1mag Scilab Function

a=.7*exp(%i*%pi/6);
[frr,bds]=xgetech(); //preserve current context
rect=[-1.2,-1.2*sqrt(2),1.2,1.2*sqrt(2)];
t=2*%pi*(0:179)/179;xsetech([0,0,0.5,1]);
plot2d(sin(t)’,cos(t)’,[2],”012”,’’,rect);
plot2d([0 real(a)]’,[0 imag(a)]’,[3],”000”)
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0]);
w0=.93*exp(-%i*%pi/15);w=exp(-(0:9)*log(w0));z=a*w;
zh=real(z);zh=imag(z);
plot2d(zh’,zh’,[5],”000”)
xsetech([0.5,0,0.5,1]);
plot2d(sin(t)’,cos(t)’,[2],”012”,’’,rect);
plot2d([0 real(a)]’,[0 imag(a)]’,[-1],”000”)
xsegs([-1.0,0;1.0,0],[0,-1.0;0,1.0]);
w0=w0/(0.93*.93);w=exp(-(0:9)*log(w0));z=a*w;
zh=real(z);zh=imag(z);
plot2d(zh’,zh’,[5],”000”)
xsetech(frr,bds); //restore context

AUTHOR: C. Bunks

2.6.15 dft ________________________________ discrete Fourier transform

CALLING SEQUENCE:

[xf]=dft(x,flag);

PARAMETERS:

x : input vector
flag : indicates dft (flag=-1) or idft (flag=1)
xf : output vector

DESCRIPTION:

Function which computes dft of vector x.

EXAMPLE:

n=8;omega = exp(-2*%pi*%i/n);
j=0:n-1;F=omega.^(j’*j); //Fourier matrix
x=1:8;x=x(:);
F*x
fft(x,-1)
dft(x,-1)
inv(F)*x
fft(x,1)
dft(x,1)

SEE ALSO: fft 457

AUTHOR: C. B.
2.6.16    ell1mag -------------------------------- magnitude of elliptic filter

CALLING SEQUENCE :

\[ v = \text{ell1mag}(\text{eps}, m1, z) \]

PARAMETERS :

- eps : passband ripple \(= 1/(1+\text{eps}^2) \)
- \( m1 \) : stopband ripple \(= 1/(1+(\text{eps}^2)/m1) \)
- \( z \) : sample vector of values in the complex plane
- \( v \) : elliptic filter values at sample points

DESCRIPTION :

Function used for squared magnitude of an elliptic filter. Usually \( m1 = \text{eps} \times \text{eps} / (A^2 - 1) \). Returns \( v = \text{real}(\text{ones}(z)/(\text{ones}(z) + \text{eps} \times \text{eps} \times \text{s} \times \text{s})) \) for \( \text{s} = \text{sn}(z, m1) \).

EXAMPLE :

deff('([alpha,beta]=alpha_beta(n,m,m1)',...
'  if 2*int(n/2)=n then, beta=K1; else, beta=0;end;...
  alpha=%k(1-m1)/%k(1-m);')
epsilon=0.1;A=10; //ripple parameters
m1=(epsilon*epsilon)/(A*A-1);n=5;omegac=6;
m=find_freq(epsilon,A,n);omegar = omegac/sqrt(m)
%k(1-m1)*%k(m)/(%k(m1)*%k(1-m))-n //Check...
[alpha,beta]=alpha_beta(n,m,m1)
alpha*%asn(1,m)-n*%k(m1) //Check
sample=0:0.01:20;
//Now we map the positive real axis into the contour...
z=alpha*%asn(sample/omegac,m)+beta*ones(sample);
plot(sample,ell1mag(epsilon,m1,z))

SEE ALSO :    buttmag 447

2.6.17    eqfir ------------------------------------- minimax approximation of FIR filter

CALLING SEQUENCE :

\[ [hn] = \text{eqfir}(nf, \text{bedge}, \text{des}, \text{wate}) \]

PARAMETERS :

- nf : number of output filter points desired
- \text{bedge} : Mx2 matrix giving a pair of edges for each band
- \text{des} : M-vector giving desired magnitude for each band
- \text{wate} : M-vector giving relative weight of error in each band
- \text{hn} : output of linear-phase FIR filter coefficients

DESCRIPTION :

Minimax approximation of multi-band, linear phase, FIR filter

EXAMPLE :

\[ \text{hn} = \text{eqfir}(33, [0 .2; .25 .35; .4 .5],[0 1 0],[1 1 1]); \]
\[ [\text{hm}, \text{fr}] = \text{frmag}(\text{hn}, 256); \]
\[ \text{plot}(\text{fr}, \text{hm}), \]

AUTHOR : C. B.
2.6.18  eqiir  ______________________________ Design of iir filters

CALLING SEQUENCE:

[cells, fact, zzeros, zpoles] = eqiir(ftype, approx, om, deltap, deltas)

PARAMETERS:

ftype : filter type ("lp", "hp", "sb", "bp")
approx : design approximation ("butt", "cheb1", "cheb2", "ellip")
om : 4-vector of cutoff frequencies (in radians) om=[om1, om2, om3, om4]. 0 <= om1 <= om2 <= om3 <= om4 <= pi. When ftype='lp' or 'hp', om3 and om4 are not used and may be set to 0.
deltap : ripple in the passband. 0 <= deltap <= 1
deltas : ripple in the stopband. 0 <= deltas <= 1
cells : realization of the filter as second order cells
fact : normalization constant
zzeros : zeros in the z-domain
zpoles : poles in the z-domain

DESCRIPTION:

Design of iir filter interface with eqiir (syredi)
The filter obtained is \( h(z) = \text{fact} \times \text{product of the elements of cells} \).
That is \( hz = \text{fact} \times \text{prod(cells(2))./prod(cells(3))} \)

EXAMPLE:

[cells, fact, zzeros, zpoles] = ...
eqiir('lp', 'ellip', [2*pi/10, 4*pi/10], 0.02, 0.001)
transfer = fact*poly(zzeros, 'z')/ poly(zpoles, 'z')

SEE ALSO:  eqfir 455,  iir 463

2.6.19  faurre  ______________________ filter computation by simple Faurre algorithm

CALLING SEQUENCE:

[P, R, T] = faurre(n, H, F, G, R0)

PARAMETERS:

n : number of iterations.
H, F, G : estimated triple from the covariance sequence of y.
R0 : E(yk*yk')
P : solution of the Riccati equation after n iterations.
R, T : gain matrix of the filter.

DESCRIPTION:

This function computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices R and T of the filter model. The algorithm tries to compute the solution P as the growing limit of a sequence of matrices Pn such that

\[
\begin{align*}
P_{n+1} &= F^*P_n*F' + (G-F^*P_n*h')*(R_0-H*P_n*H') * (G' - H*P_n*F')^{-1} \\
P_0 &= G*R_0 * G'
\end{align*}
\]

Scilab Group  April 1993  456
Note that this method may not converge, especially when F has poles near the unit circle. Use preferably the srfaur function.

**Author:** G. Le V.

**SEE ALSO:** srfaur 476, linquist ??, phc 471

### 2.6.20 ffilt

**coefficients of FIR low-pass**

**CALLING SEQUENCE:**

\[ [x] = \text{ffilt}(ft, n, fl, fh) \]

**PARAMETERS:**

- \( ft \) : filter type where \( ft \) can take the values "lp" for low-pass filter, "hp" for high-pass filter, "bp" for band-pass filter, and "sb" for stop-band filter.
- \( n \) : integer (number of filter samples desired)
- \( fl \) : real (low frequency cut-off)
- \( fh \) : real (high frequency cut-off)
- \( x \) : vector of filter coefficients

**DESCRIPTION:**

Get \( n \) coefficients of a FIR low-pass, high-pass, band-pass, or stop-band filter. For low and high-pass filters one cut-off frequency must be specified whose value is given in \( fl \). For band-pass and stop-band filters two cut-off frequencies must be specified for which the lower value is in \( fl \) and the higher value is in \( fh \)

**Author:** C. B.

### 2.6.21 fft

**fast Fourier transform.**

**CALLING SEQUENCE:**

- \( [x] = \text{fft}(a, -1) \)
- \( [x] = \text{fft}(a, 1) \)
- \( x = \text{fft}(a, -1, \text{dim}, \text{incr}) \)
- \( x = \text{fft}(a, 1, \text{dim}, \text{incr}) \)

**PARAMETERS:**

- \( x \) : real or complex vector. Real or complex matrix (2-dim fft)
- \( a \) : real or complex vector.
- \( \text{dim} \) : integer
- \( \text{incr} \) : integer

**DESCRIPTION:**

Short syntax (one or two dimensional fft):
- \( x = \text{fft}(a, -1) \) gives a direct transform (the \(-1\) refers to the sign of the exponent..., NOT to "inverse"), that is

\[
x(k) = \sum_{m=1}^{n} a(m) e^{\pi(i(m - 1)(k - 1)/n)}
\]
for k varying from 1 to n (n=size of vector a).
  a=fft(x,1) performs the inverse transform normalized by 1/n.
  (fft(fft(:,,-1),1) is identity).
When the first argument given to fft is a matrix a two-dimensional FFT is performed.
Long syntax (multidimensional FFT):  x=fft(a,-1,dim,incr) allows to perform an multidimen-
sional fft.
If a is a real or complex vector implicitly indexed by x1,x2,...,xp i.e. a(x1,x2,...,xp) where x1
lies in 1..dim1, x2 in 1.. dim2,... one gets a p-dimensional FFT p by calling p times fft
as follows
  a1=fft(a,-1,dim1,incr1)
  a2=fft(a1,-1,dim2,incr2) ...

  where dimi is the dimension of the current variable w.r.t which one is integrating and incri is the
increment which separates two successive xi elements in a.
In particular,if a is an nxm matrix, x=fft(a,-1) is equivalent to the two instructions:
  a1=fft(a,-1,m,1) and x=fft(a1,-1,n,m).
if a is an hypermatrix (see hypermat) fft(a,flag) performs the N dimensional fft of a.
EXAMPLE :
  a=[1;2;3];n=size(a,'*');
  norm(1/n*exp(2*%i*%pi*(0:n-1).'.*(0:n-1)/n)*a -fft(a,1))
  norm(exp(-2*%i*%pi*(0:n-1).'.*(0:n-1)/n)*a -fft(a,-1))

SEE ALSO:  corr 450

2.6.22 filter .................................................................................................................. modelling filter

CALLING SEQUENCE :

[y,xt]=filter(n,F,H,Rt,T)

PARAMETERS :

n : number of computed points.
F, H : relevant matrices of the Markovian model.
Rt, T : gain matrices.
y : output of the filter.
xt : filter process.

DESCRIPTION :
This function computes the modelling filter
SEE ALSO:  faurre 456

AUTHOR : G. Le V.

2.6.23 find_freq ........................................ parameter compatibility for elliptic filter design

CALLING SEQUENCE :

[m]=find_freq(epsilon,A,n)

PARAMETERS :

epsilon : passband ripple
A : passband ripple
n : number of computed points.
A : stopband attenuation
n : filter order
m : frequency needed for construction of elliptic filter

DESCRIPTION:
Search for \( m \) such that
\[
\frac{n}{K(1-m)K(m)} = \frac{m_1}{(A^2 - 1)(\epsilon^2 / A^2 - 1)}
\]
with
\[
m_1 = \frac{\epsilon \epsilon}{A^2 - 1}
\]
If \( m = \frac{\omega_r^2}{\omega_c^2} \), the parameters \( \epsilon, A, \omega_c, \omega_r \) and \( n \) are then compatible for defining a prototype elliptic filter. Here, \( K = K(m) \) is the complete elliptic integral with parameter \( m \).

SEE ALSO: \( %k \) 445

AUTHOR: F. D.

2.6.24 findm  ____________________________ for elliptic filter design

CALLING SEQUENCE:

\[
[m] = \text{findm}(\chi)
\]

DESCRIPTION:
Search for \( m \) such that \( \chi = \frac{K(1-m)}{K(m)} \) (For use with \text{findfreq}).

SEE ALSO: \( %k \) 445

AUTHOR: F. D.

2.6.25 frfit  ____________________________ frequency response fit

CALLING SEQUENCE:

\[
\text{sys} = \text{frfit}(w, f, \text{order})
\]
\[
[num, den] = \text{frfit}(w, f, \text{order})
\]
\[
\text{sys} = \text{frfit}(w, f, \text{order}, \text{weight})
\]
\[
[num, den] = \text{frfit}(w, f, \text{order}, \text{weight})
\]

PARAMETERS:
- \( w \): positive real vector of frequencies (Hz)
- \( f \): complex vector of frequency responses (same size as \( w \))
- \( \text{order} \): integer (required order, degree of \( \text{den} \))
- \( \text{weight} \): positive real vector (default value \( \text{ones}(w) \)).
- \( \text{num}, \text{den} \): stable polynomials

DESCRIPTION:
\[
\text{sys} = \text{frfit}(w, f, \text{order}, \text{weight})
\]
returns a bi-stable transfer function \( G(s) = \frac{\text{num}}{\text{den}} \)
of given order such that its frequency response \( G(w(i)) \) matches \( f(i) \), i.e. \( \text{freq}(\text{num}, \text{den}, \text{i} \cdot w) \) should be close to \( f(i) \). \( \text{weight}(i) \) is the weight given to \( w(i) \).

EXAMPLE:

\[
w = 0.01:0.01:2; s = \text{poly}(0, 's');
\]
\[
G = \text{syslin}('c', 2*(s^2+0.1*s+2), (s^2+s+1)*(s^2+0.3*s+1));
\]
\[
f = \text{repfreq}(G, w);
\]
\[
Gid = \text{frfit}(w, f, 4);
\]
\[
fres = \text{repfreq}(Gid, w);
\]
\[
\text{bode}(w, [f; \text{fres}; \text{fresfit}])
\]

SEE ALSO: \( \text{frep2tf} \) 338, \( \text{factors} \) 488, \( \text{cepstrum} \) 448, \( \text{mrfit} \) 470, \( \text{freq} \) 339, \( \text{calfrq} \) 324
### 2.6.26  **frmag** __magnitude of FIR and IIR filters__

**CALLING SEQUENCE:**

\[ [\text{xm}, \text{fr}] = \text{frmag}(\text{num}, \text{den}, \text{npts}) \]

**PARAMETERS:**

- \text{npts} : integer (number of points in frequency response)
- \text{xm} : mvector of magnitude of frequency response at the points \text{fr}
- \text{fr} : points in the frequency domain where magnitude is evaluated
- \text{num} : if \text{den} is omitted vector coefficients/polynomial/rational polynomial of filter
- \text{num} : if \text{den} is given vector coefficients/polynomial of filter numerator
- \text{den} : vector coefficients/polynomial of filter denominator

**DESCRIPTION:**

Calculates the magnitude of the frequency responses of FIR and IIR filters. The filter description can be one or two vectors of coefficients, one or two polynomials, or a rational polynomial.

**AUTHOR:** C. B.

### 2.6.27  **fsfirlin** __design of FIR, linear phase filters, frequency sampling technique__

**CALLING SEQUENCE:**

\[ [\text{hst}] = \text{fsfirlin}(\text{hd}, \text{flag}) \]

**PARAMETERS:**

- \text{hd} : vector of desired frequency response samples
- \text{flag} : is equal to 1 or 2, according to the choice of type 1 or type 2 design
- \text{hst} : vector giving the approximated continuous response on a dense grid of frequencies

**DESCRIPTION:**

Function for the design of FIR, linear phase filters using the frequency sampling technique.

**EXAMPLE:**

```
// Example of how to use the fsfirlin macro for the design of an FIR filter by a frequency sampling technique.
// Two filters are designed: the first (response hst1) with abrupt transitions from 0 to 1 between passbands and stop bands; the second (response hst2) with one sample in each transition band (amplitude 0.5) for smoothing.

hd=[zeros(1,15) ones(1,10) zeros(1,39)]; // desired samples
hst1=fsfirlin(hd,1); // filter with no sample in the transition
hd(15)=.5;hd(26)=.5; // samples in the transition bands
hst2=fsfirlin(hd,1); // corresponding filter
pas=1/prod(size(hst1))*0.5;
fg=0:pas:.5; // normalized frequencies grid
plot2d([1 1]*fg(1:257),', [hst1' hst2']); // 2nd example
```

**AUTHOR:** G. Le Vey

---

*Scilab Group April 1993 460*
hd=[0*ones(1,15) ones(1,10) 0*ones(1,39)]; //desired samples
hst1=fsfirlin(hd,1); //filter with no sample in the transition
hd(15)=.5;hd(26)=.5; //samples in the transition bands
hst2=fsfirlin(hd,1); //corresponding filter
pas=1/prod(size(hst1))*0.5;
fg=0:pas:0.5; //normalized frequencies grid
n=prod(size(hst1))
plot(fg(1:n),hst1);
plot2d(fg(1:n)',hst2',[3],"000");

SEE ALSO: ffilt, wfir

2.6.28 group  group delay for digital filter

CALLING SEQUENCE:
[tg,fr]=group(npts,a1i,a2i,b1i,b2i)

PARAMETERS:

npts : integer : number of points desired in calculation of group delay
a1i  : in coefficient, polynomial, rational polynomial, or cascade polynomial form this variable is the
      transfer function of the filter. In coefficient polynomial form this is a vector of coefficients (see
      below).
a2i  : in coeff poly form this is a vector of coeffs
b1i  : in coeff poly form this is a vector of coeffs
b2i  : in coeff poly form this is a vector of coeffs
tg  : values of group delay evaluated on the grid fr
fr  : grid of frequency values where group delay is evaluated

DESCRIPTION:
Calculate the group delay of a digital filter with transfer function h(z).
The filter specification can be in coefficient form, polynomial form, rational polynomial form, cascade
polynomial form, or in coefficient polynomial form.
In the coefficient polynomial form the transfer function is formulated by the following expression
h(z)=prod(a1i+a2i*z+z**2)/prod(b1i+b2i*z+zˆ2)

EXAMPLE:
z=poly(0,'z');
h=z/((z-.5);
[tg,fr]=group(100,h);
plot(fr,tg)

AUTHOR: C. B.

2.6.29 hank  covariance to hankel matrix

CALLING SEQUENCE:
[hk]=hank(m,n,cov)

PARAMETERS:
m  : number of bloc-rows
**n** : number of bloc-columns  
**cov** : sequence of covariances; it must be given as : [R0 R1 R2...Rk]  
**hk** : computed hankel matrix  

**DESCRIPTION :**  
this function builds the hankel matrix of size (m*d, n*d) from the covariance sequence of a vector process  

**EXAMPLE :**  

// Example of how to use the hank macro for  
// building a Hankel matrix from multidimensional  
// data (covariance or Markov parameters e.g.)  
//  
// This is used e.g. in the solution of normal equations  
// by classical identification methods (Instrumental Variables e.g.)  
//  
// 1) let's generate the multidimensional data under the form :  
// C=[c_0 c_1 c_2 .... c_n]  
// where each bloc c_k is a d-dimensional matrix (e.g. the k-th correlation  
// of a d-dimensional stochastic process X(t) [c_k = E(X(t) X'(t+k)], '  
// being the transposition in scilab)  
//  
// we take here d=2 and n=64  
//  
c=rand(2,2*64)  
//  
// generate the hankel matrix H (with 4 bloc-rows and 5 bloc-columns)  
// from the data in c  
//  
H=hank(4,5,c);  
//  
**SEE ALSO :**  
toeplitz 227  

### 2.6.30 **hilb**  
**Hilbert transform**  

**CALLING SEQUENCE :**  

[xh]=hilb(n[,wtype][,par])  

**PARAMETERS :**  

- **n** : odd integer : number of points in filter  
- **wtype** : string : window type ('re','tr','hn','hm','kr','ch') (default = 're')  
- **par** : window parameter for wtype='kr' or 'ch' default par=[0 0] see the function window for more help  

**DESCRIPTION :**  
returns the first n points of the Hilbert transform centred around the origin. That is, xh=(2/(n*pi))*(sin(n*pi/2))^2.  

**EXAMPLE :**  

plot(hilb(51))  

**AUTHOR :**  
C. B.
CALLING SEQUENCE:

\[ hz = \text{iir}(n, \text{ftype}, \text{fdesign}, \text{frq}, \text{delta}) \]

PARAMETERS:

- \( n \): filter order (pos. integer)
- \( \text{ftype} \): string specifying the filter type ‘lp’, ‘hp’, ‘bp’, ‘sb’
- \( \text{fdesign} \): string specifying the analog filter design = ‘butt’, ‘cheb1’, ‘cheb2’, ‘ellip’
- \( \text{frq} \): 2-vector of discrete cut-off frequencies (i.e., \( 0 < \text{frq} < 0.5 \)). For \( \text{lp} \) and \( \text{hp} \) filters only \( \text{frq}(1) \) is used. For \( \text{bp} \) and \( \text{sb} \) filters \( \text{frq}(1) \) is the lower cut-off frequency and \( \text{frq}(2) \) is the upper cut-off frequency.
- \( \text{delta} \): 2-vector of error values for \( \text{cheb1} \), \( \text{cheb2} \), and \( \text{ellip} \) filters where only \( \text{delta}(1) \) is used for \( \text{cheb1} \) case, only \( \text{delta}(2) \) is used for \( \text{cheb2} \) case, and \( \text{delta}(1) \) and \( \text{delta}(2) \) are both used for \( \text{ellip} \) case. \( 0 < \text{delta}(1), \text{delta}(2) < 1 \)
- For \( \text{cheb1} \) filters \( 1 - \text{delta}(1) < \text{ripple} < 1 \) in passband
- For \( \text{cheb2} \) filters \( 0 < \text{ripple} < \text{delta}(2) \) in stopband
- For \( \text{ellip} \) filters \( 1 - \text{delta}(1) < \text{ripple} < 1 \) in passband and \( 0 < \text{ripple} < \text{delta}(2) \) in stopband.

DESCRIPTION:

function which designs an iir digital filter using analog filter designs.

EXAMPLE:

\[
\text{hz}=\text{iir}(3, 'bp', 'ellip', [0.15 .25], [.08 .03]);
\]

\[
[\text{hzm}, \text{fr}]=\text{frmag}(\text{hz}, 256);
\]

\[
\text{plot2d}(\text{fr}, \text{hzm})
\]

\[
\text{xtitle}('\text{Discrete IIR filter band pass } 0.15<\text{fr}<0.25 ','','');
\]

\[
\text{q} = \text{poly}(0,'q'); \quad \quad \text{ //to express the result in terms of the } ...\]

\[
\text{hzd} = \text{horner}(\text{hz}, 1/\text{q}) \quad \text{ //delay operator } q=z^{-1}
\]

SEE ALSO: eqfir 455, eqiir 456

AUTHOR: C. B.

2.6.32 iirgroup ____________________ group delay Lp IIR filter optimization

CALLING SEQUENCE:

\[
[\text{lt, grad}]=\text{iirgroup}(p, r, \theta, \omega, \omega, \text{wt}, \text{td})
\]

\[
[\text{cout, grad, ind}]=\text{iirlp}(x, \text{ind}, p, [\text{flag}], \lambda, \omega, \text{ad}, \text{wa}, \text{td}, \text{wt})
\]

PARAMETERS:

- \( r \): vector of the module of the poles and the zeros of the filters
- \( \theta \): vector of the argument of the poles and the zeros of the filters
- \( \omega \): frequencies where the filter specifications are given
- \( \text{wt} \): weighting function for and the group delay
- \( \text{td} \): desired group delay
- \( \text{lt}, \text{ grad} \): criterium and gradient values

DESCRIPTION:

optimization of IIR filters for the Lp criterium for the the group delay. (Rabiner & Gold pp270-273).
2.6.33  **iirlp**  

Lp IIR filter optimization

**CALLING SEQUENCE:**

```plaintext
[cost, grad, ind] = iirlp(x, ind, p, [flag], lambda, omega, ad, wa, td, wt)
```

**PARAMETERS:**

- `x`: 1X2 vector of the module and argument of the poles and the zeros of the filters
- `flag`: string: ’a’ for amplitude, ’gd’ for group delay; default case for amplitude and group delay.
- `omega`: frequencies where the filter specifications are given
- `wa, wt`: weighting functions for the amplitude and the group delay
- `lambda`: weighting (with 1-lambda) of the costs (’a’ and ’gd’ for getting the global cost.
- `ad, td`: desired amplitude and group delay

**DESCRIPTION:**

optimization of IIR filters for the Lp criterium for the amplitude and/or the group delay. (Rabiner & Gold pp270-273).

2.6.34  **intdec**  

Changes sampling rate of a signal

**CALLING SEQUENCE:**

```plaintext
[y] = intdec(x, lom)
```

**PARAMETERS:**

- `x`: input sampled signal
- `lom`: For a 1D signal this is a scalar which gives the rate change. For a 2D signal this is a 2-vector of sampling rate changes `lom=(col rate change, row rate change)`
- `y`: Output sampled signal

**DESCRIPTION:**

Changes the sampling rate of a 1D or 2D signal by the rates in `lom`

**AUTHOR:** C. B.

2.6.35  **jmat**  

row or column block permutation

**CALLING SEQUENCE:**

```plaintext
[j] = jmat(n, m)
```

**PARAMETERS:**

- `n`: number of block rows or block columns of the matrix
- `m`: size of the (square) blocks

**DESCRIPTION:**

This function permutes block rows or block columns of a matrix
CALLING SEQUENCE:

\[ [x1, p1, x, p] = \text{kalm}(y, x0, p0, f, g, h, q, r) \]

PARAMETERS:

\( f, g, h \) : current system matrices  
\( q, r \) : covariance matrices of dynamics and observation noise  
\( x0, p0 \) : state estimate and error variance at \( t=0 \) based on data up to \( t=-1 \)  
\( y \) : current observation  
Output from the function is:

\( x1, p1 \) : updated estimate and error covariance at \( t=1 \) based on data up to \( t=0 \)  
\( x \) : updated estimate and error covariance at \( t=0 \) based on data up to \( t=0 \)

DESCRIPTION:

function which gives the Kalman update and error variance

AUTHOR: C. B.

2.6.37 lattn  
recursive solution of normal equations

CALLING SEQUENCE:

\[ [la, lb] = \text{lattn}(n, p, cov) \]

PARAMETERS:

n : maximum order of the filter  
p : fixed dimension of the MA part. If \( p = -1 \), the algorithm reduces to the classical Levinson recursions.  
cov : matrix containing the \( R_k \)'s (\( d \times d \) matrices for a \( d \)-dimensional process). It must be given the following way

\[ \text{cov} = \begin{bmatrix} R_0 \\ R_1 \\ R_2 \\ \vdots \\ R_{n\text{lag}} \end{bmatrix} \]

la : list-type variable, giving the successively calculated polynomials (degree 1 to degree \( n \)), with coefficients \( A_k \)

DESCRIPTION:

solves recursively on \( n \) (\( p \) being fixed) the following system (normal equations), i.e. identifies the AR part (poles) of a vector ARMA\((n,p)\) process

\[ \begin{pmatrix} I & -A_1 & -A_2 & \cdots & -A_n \\ R_p & R_{p+1} & \cdots & R_{p+n-1} \\ \vdots & \vdots & \ddots & \vdots \\ R_{p+1-n} & R_{p+2-n} & \cdots & R_p \end{pmatrix} = 0 \]

where \{\( R_k; k=1, n\text{lag} \)\} is the sequence of empirical covariances

AUTHOR: G. Le V.
2.6.38 lattp

CALLING SEQUENCE:

[la, lb] = lattp(n, p, cov)

DESCRIPTION:

see lattn

AUTHOR: G. Levey

2.6.39 lev

Yule-Walker equations (Levinson’s algorithm)

CALLING SEQUENCE:

[ar, sigma2, rc] = lev(r)

PARAMETERS:

r: correlation coefficients
ar: auto-Regressive model parameters
sigma2: scale constant
rc: reflection coefficients

DESCRIPTION:

resolve the Yule-Walker equations

\[
\begin{pmatrix}
R_0 & R_1 & \ldots & R_{N-1} \\
R_1 & R_0 & \ldots & R_{N-2} \\
\vdots & \vdots & \ddots & \vdots \\
R_{N-1} & R_{N-2} & \ldots & R_0
\end{pmatrix}
\begin{pmatrix}
ar_1 \\
ar_2 \\
\vdots \\
ar_{N-1}
\end{pmatrix}
= 
\begin{pmatrix}
\sigma_2 \\
0 \\
\vdots \\
0
\end{pmatrix}
\]

where \( R_i = r(i-1) \). using Levinson’s algorithm.

AUTHOR: C. B.

2.6.40 levin

Toeplitz system solver by Levinson algorithm (multidimensional)

CALLING SEQUENCE:

[la, sig] = levin(n, cov)

PARAMETERS:

n: maximum order of the filter
cov: matrix containing the \( R_k \) (d x d matrices for a d-dimensional process). It must be given the following way:

\[
\begin{pmatrix}
R_0 \\
R_1 \\
R_2 \\
\vdots \\
R_{n \times g}
\end{pmatrix}
\]
levin Scilab Function

la : list-type variable, giving the successively calculated Levinson polynomials (degree 1 to n), with coefficients Ak

sig : list-type variable, giving the successive mean-square errors.

DESCRIPTION:
function which solves recursively on n the following Toeplitz system (normal equations)

\[
\begin{pmatrix}
R_1 & R_2 & \ldots & R_n \\
R_0 & R_1 & \ldots & R_{n-1} \\
R_{-1} & R_0 & \ldots & R_{n-2} \\
\vdots & \vdots & \ldots & \vdots \\
R_{-n} & R_{-n} & \ldots & R_1 \\
R_{1-n} & R_{2-n} & \ldots & R_0 \\
\end{pmatrix}
(I - A_1 \ldots - A_n) = 0
\]

where \(\{R_k; k=1,n_{lag}\}\) is the sequence of \(n_{lag}\) empirical covariances

EXAMPLE:

//We use the 'levin' macro for solving the normal equations
//on two examples: a one-dimensional and a two-dimensional process.
//We need the covariance sequence of the stochastic process.
//This example may usefully be compared with the results from
//the 'phc' macro (see the corresponding help and example in it)
//
//1) A one-dimensional process
//-------------------------

//We generate the process defined by two sinusoids (1Hz and 2 Hz)
//in additive Gaussian noise (this is the observed process);
//the simulated process is sampled at 10 Hz (step 0.1 in t, underafter).
//
t1=0:.1:100;rand('normal');
y1=sin(2*%pi*t1)+sin(2*%pi*2*t1);y1=y1+rand(y1);plot(t1,y1);

//covariance of y1

nlag=128;
c1=corr(y1,nlag);
c1=c1';//c1 needs to be given columnwise (see the section PARAMETERS of this help)

//compute the filter for a maximum order of n=10
//la is a list-type variable each element of which
//containing the filters of order ranging from 1 to n; (try varying n)
//in the d-dimensional case this is a matrix polynomial (square, d X d)
//sig gives, the same way, the mean-square error
//
n=15;
[la1,sig1]=levin(n,c1);

//verify that the roots of 'la' contain the
//frequency spectrum of the observed process y
//(remember that y is sampled -in our example
//at 10Hz (T=0.1s) so that we need to retrieve

AUTHOR: G. Le Vey
//the original frequencies (1Hz and 2 Hz) through
//the log and correct scaling by the frequency sampling
//we verify this for each filter order
//for i=1:n, s1=roots(la1(i));s1=log(s1)/2/%pi/.1;
//
//now we get the estimated poles (sorted, positive ones only !)
//s1=sort(imag(s1));s1=s1(1:i/2);end;
//
//the last two frequencies are the ones really present in the observed
//process --> the others are "artifacts" coming from the used model size.
//This is related to the rather difficult problem of order estimation.
//
//2) A 2-dimensional process
// -----------------------

// [4 frequencies 1, 2, 3, and 4 Hz, sampled at 0.1 Hz :
// y_1 = sin(2*Pi*t) + sin(2*Pi*2*t) + Gaussian noise
// y = |
// | y_2 |
// |
// |
// 2: y_1=sin(t2)+sin(2*t2)+rand(t2);sin(3*t2)+sin(4*t2)+rand(t2)];
c2=[];
for j=1:2, for k=1:2, c2=[c2;corr(y2(k,:),y2(j,:),nlag)];end;end;
c2=matrix(c2,2,128);cov=[];
for j=1:64,cov=[cov;c2(:,(j-1)*d+1:j*d)];end; // covar. columnwise

//in the multidimensional case, we have to compute the
//roots of the determinant of the matrix polynomial
//(easy in the 2-dimensional case but tricky if d>=3 !).
//We just do that here for the maximum desired
//filter order (n); mp is the matrix polynomial of degree n
//
[l2a2,s2g2]=levin(n,c2);
mp=l2a2(n);determinant=mp(1,1)*mp(2,2)-mp(1,2)*mp(2,1);
s2=roots(determinant);s2=log(s2)/2/%pi/0.1; // same trick as above for 1D process
s2=sort(imag(s2));s2=s2(1:d*n/2); // just the positive ones !

// There the order estimation problem is seen to be much more difficult !
// many artifacts ! The 4 frequencies are in the estimated spectrum
// but beneath many non relevant others.

SEE ALSO: phc 471

2.6.41  lgfft __________________________ utility for fft

CALLING SEQUENCE:
[y]=lgfft(x)

PARAMETERS:

x : real or complex vector

DESCRIPTION:
returns the lowest power of 2 larger than size(x) (for FFT use).

2.6.42 lindquist ____________________________ Lindquist’s algorithm

CALLING SEQUENCE:

[P,R,T]=lindquist(n,H,F,G,R0)

PARAMETERS:

n : number of iterations.
H, F, G : estimated triple from the covariance sequence of y.
R0 : E(yk*yk')
P : solution of the Riccati equation after n iterations.
R, T : gain matrices of the filter.

DESCRIPTION:
computes iteratively the minimal solution of the algebraic Riccati equation and gives the matrices R and T of the filter model, by the Lindquist’s algorithm.

AUTHOR: G. Le V.

SEE ALSO: srfaur 476, faurre 456, phc 471

2.6.43 mese ________________________ maximum entropy spectral estimation

CALLING SEQUENCE:

[sm,fr]=mese(x [,npts]);

PARAMETERS:

x : Input sampled data sequence
npts : Optional parameter giving number of points of fr and sm (default is 256)
sm : Samples of spectral estimate on the frequency grid fr
fr : npts equally spaced frequency samples in [0, .5]

DESCRIPTION:
Calculate the maximum entropy spectral estimate of x

AUTHOR: C. B.

Scilab Group April 1993 469
2.6.44 mfft  ______________________________ multi-dimensional fft

**CALLING SEQUENCE:**

\[ [xk]=mfft(x,flag,dim) \]

**PARAMETERS:**

- **x**: \( x(i,j,k,...) \) input signal in the form of a row vector whose values are arranged so that the \( i \) index runs the quickest, followed by the \( j \) index, etc.
- **flag**: (-1) FFT or (1) inverse FFT
- **dim**: dimension vector which gives the number of values of \( x \) for each of its indices
- **xk**: output of multidimensional fft in same format as for \( x \)

**DESCRIPTION:**

FFT for a multi-dimensional signal

For example for a three dimensional vector which has three points along its first dimension, two points along its second dimension and three points along its third dimension the row vector is arranged as follows

\[
x=[x(1,1,1),x(2,1,1),x(3,1,1), \]
\[
x(1,2,1),x(2,2,1),x(3,2,1), \]
\[
x(1,1,2),x(2,1,2),x(3,1,2), \]
\[
x(1,2,2),x(2,2,2),x(3,2,2), \]
\[
x(1,1,3),x(2,1,3),x(3,1,3), \]
\[
x(1,2,3),x(2,2,3),x(3,2,3)]
\]

and the dim vector is: \( \text{dim}=[3,2,3] \)

**AUTHOR**: C. B.

2.6.45 mrfit  ______________________________ frequency response fit

**CALLING SEQUENCE:**

\[
sys=mrfit(w,mag,order) \\
[num,den]=mrfit(w,mag,order) \\
sys=mrfit(w,mag,order,weight) \\
[num,den]=mrfit(w,mag,order,weight)
\]

**PARAMETERS:**

- **w**: positive real vector of frequencies (Hz)
- **mag**: real vector of frequency responses magnitude (same size as \( w \))
- **order**: integer (required order, degree of \( \text{den} \))
- **weight**: positive real vector (default value \( \text{ones}(w) \)).
- **num**, **den**: stable polynomials

**DESCRIPTION:**

\[ \text{sys=mrfit}(w,\text{mag},\text{order},\text{weight}) \] returns a bi-stable transfer function \( G(s)=\text{sys}=\text{num}/\text{den}, \) of of given \( \text{order} \) such that its frequency response magnitude \( \text{abs}(G(w(i))) \) matches \( \text{mag}(i) \) i.e. \( \text{abs}(\text{freq}(\text{num},\text{den},\%i*w)) \) should be close to \( \text{mag} \). \( \text{weight}(i) \) is the weight given to \( w(i) \).

**EXAMPLE:**

Scilab Group  
April 1993  
470
w=0.01:0.01:2;s=poly(0,'s');
G=syslin('c',2*(s^2+0.1*s+2),(s^2+s+1)*(s^2+0.3*s+1)); // syslin('c',Num,Den);
fresp=repfreq(G,w);
mag=abs(fresp);
Gid=mrfit(w,mag,4);
frespfit=repfreq(Gid,w);
plot2d([w',w'],[mag(:),abs(frespfit(:))])

SEE ALSO:  cepstrum 448,  frfit 459,  freq 339,  calfrq 324

2.6.46  phc  ________________ Markovian representation

CALLING SEQUENCE:

[H,F,G]=phc(hk,d,r)

PARAMETERS:

hk : hankel matrix
d : dimension of the observation
r : desired dimension of the state vector for the approximated model
H, F, G : relevant matrices of the Markovian model

DESCRIPTION:

Function which computes the matrices H, F, G of a Markovian representation by the principal hankel component approximation method, from the hankel matrix built from the covariance sequence of a stochastic process.

EXAMPLE:

//This example may usefully be compared with the results from
//the 'levin' macro (see the corresponding help and example)
//We consider the process defined by two sinusoids (1Hz and 2 Hz)
//in additive Gaussian noise (this is the observation);
//the simulated process is sampled at 10 Hz.
//t=0:.1:100;rand('normal');
y=sin(2*%pi*t)+sin(2*%pi*2*t);y=y+rand(y);plot(t,y)
//covariance of y
//nlag=128;
c=corr(y,nlag);
//hankel matrix from the covariance sequence
//(we can choose to take more information from covariance
//by taking greater n and m; try it to compare the results !
//n=20;m=20;
h=hank(n,m,c);
//compute the Markov representation (mh,mf,mg)
//We just take here a state dimension equal to 4 :
/this is the rather difficult problem of estimating the order!
//Try varying ns!
//(the observation dimension is here equal to one)
ns=4;
[mh,mf,mg]=phc(h,1,ns);
//
//verify that the spectrum of mf contains the
//frequency spectrum of the observed process y
//(remember that y is sampled -in our example
//at 10Hz (T=0.1s) so that we need
//to retrieve the original frequencies through the log
//and correct scaling by the frequency sampling)
//
s=spec(mf);s=log(s);
s=s/2/%pi/.1;
//
//now we get the estimated spectrum
imag(s),
//

SEE ALSO: levin 466

2.6.47 p spect __________________________ cross-spectral estimate between 2 series

CALLING SEQUENCE:

[sm,cwp]=pspect(sec_step,sec_leng,wtype,x,y,wpar)

PARAMETERS:

x : data if vector, amount of input data if scalar
y : data if vector, amount of input data if scalar
sec_step : offset of each data window
sec_leng : length of each data window
wtype : window type (re, tr, hm, hn, kr, ch)
wpar : optional parameters for wtype='kr', wpar>0 for wtype='ch', 0<wpar(1)<.5, wpar(2)>0
sm : power spectral estimate in the interval [0,1]
cwp : unspecified Chebyshev window parameter

DESCRIPTION:
Cross-spectral estimate between x and y if both are given and auto-spectral estimate of x otherwise.
Spectral estimate obtained using the modified periodogram method.

EXAMPLE:

rand(’normal’);rand(’seed’,0);
x=rand(1:1024-33+1);
//make low-pass filter with eqfir
nf=33;bedge=[0 .1;.125 .5];des=[1 0];wate=[1 1];
h=eqfir(nf,bedge,des,wate);
//filter white data to obtain colored data
h1=[h 0*ones(1:maxi(size(x))-1)];
x1=[x 0*ones(1:maxi(size(h))-1)];
hf=fft(h1,-1); xf=fft(x1,-1);yf=hf.*xf;y=real(fft(yf,1));
//plot magnitude of filter
remez Scilab Function

//h2=[h 0*ones(1:968)];hf2=fft(h2,-1);hf2=real(hf2.*conj(hf2));
//hsize=maxi(size(hf2));fr=(1:hsize)/hsize;plot(fr,log(hf2));
//pspect example
sm=pspect(100,200,’tr’,y);smsize=maxi(size(sm));fr=(1:smsize)/smsize;
plot(fr,log(sm));
rand(’unif’);

SEE ALSO: cspect 452

REFERENCE:
Digital Signal Processing by Oppenheim and Schafer

2.6.48 remez ___________________________________ Remez’s algorithm

CALLING SEQUENCE:

[an]=remez(nc,fg,ds,wt)

PARAMETERS:

nc : integer, number of cosine functions
fg,ds,wt : real vectors
fg : grid of frequency points in [0,.5)
ds : desired magnitude on grid fg
wt : weighting function on error on grid fg

DESCRIPTION:
minimax approximation of a frequency domain magnitude response. The approximation takes the form

\[ h = \sum \{ a(n) \cos(\omega n) \} \]

for \( n=0,1,\ldots,\text{nc} \). An FIR, linear-phase filter can be obtained from the output of remez by using the following commands:

\[ h_{\text{nc}-1} = a(\text{nc}-12)/2; \]
\[ h_{\text{nc}} = a(1); \]
\[ h_{\text{nc+12*nc}-1} = a(2\text{nc})/2; \]

where \( a(n) \) = cosine filter coefficients

SEE ALSO: remezb 473

2.6.49 remezb ______________________ Minimax approximation of magnitude response

CALLING SEQUENCE:

[an]=remezb(nc,fg,ds,wt)

PARAMETERS:

nc : Number of cosine functions
fg : Grid of frequency points in [0,.5)
ds : Desired magnitude on grid fg
wt : Weighting function on error on grid fg
an : Cosine filter coefficients

Scilab Group April 1993 473
DESCRIPTION:
Minimax approximation of a frequency domain magnitude response. The approximation takes the form
\[ h = \sum [a(n) \cos(wn)] \] for \( n=0,1,...,nc \). An FIR, linear-phase filter can be obtained from the output of the function by using the following commands

\[
\begin{align*}
\text{hn}(1:nc-1) &= \text{an}(nc:-1:2)/2; \\
\text{hn}(nc) &= \text{an}(1); \\
\text{hn}(nc+1:2*nc-1) &= \text{an}(2:nc)/2;
\end{align*}
\]

EXAMPLE:

```matlab
// Choose the number of cosine functions and create a dense grid in [0,.24) and [.26,.5]
nc=21; ngrid=nc*16;
fg=.24*(0:ngrid/2-1)/(ngrid/2-1);
fg(ngrid/2+1:ngrid)=fg(1:ngrid/2)+.26*ones(1:ngrid/2);
// Specify a low pass filter magnitude for the desired response
ds(1:ngrid/2)=ones(1:ngrid/2);
ds(ngrid/2+1:ngrid)=zeros(1:ngrid/2);
// Specify a uniform weighting function
wt=ones(fg);
// Run remezb
an=remezb(nc,fg,ds,wt);
// Make a linear phase FIR filter
hn(1:nc-1)=an(nc:-1:2)/2; 
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;
// Plot the filter's magnitude response
plot(.5*(0:255)/256,frmag(hn,256));
```

// Choose the number of cosine functions and create a dense grid in [0,.5)
nc=21; ngrid=nc*16;
fg=.5*(0:(ngrid-1))/ngrid;
// Specify a triangular shaped magnitude for the desired response
ds(1:ngrid/2)=(0:ngrid/2-1)/(ngrid/2-1);
ds(ngrid/2+1:ngrid)=ds(ngrid/2:-1:1);
// Specify a uniform weighting function
wt=ones(fg);
// Run remezb
an=remezb(nc,fg,ds,wt);
// Make a linear phase FIR filter
hn(1:nc-1)=an(nc:-1:2)/2; 
hn(nc)=an(1);
hn(nc+1:2*nc-1)=an(2:nc)/2;
// Plot the filter’s magnitude response
plot(.5*(0:255)/256,frmag(hn,256));
```

AUTHOR: C. B.

SEE ALSO: eqfir 455

2.6.50 rplem ........................................................................ RPEM estimation

CALLING SEQUENCE:
sinc Scilab Function

\[ \text{[w1, [v]] = rpem}(w0, u0, y0, [\lambda, [k, [c]]]) \]

PARAMETERS:

\[ a, b, c : a=[a(1), \ldots, a(n)], \quad b=[b(1), \ldots, b(n)], \quad c=[c(1), \ldots, c(n)] \]

\[ w0 : \text{list}(\theta, p, \phi, \psi, l) \text{ where:} \]

\[ \theta : [a,b,c] \] is a real vector of order \(3*n\)

\[ p : (3*n \times 3*n) \text{ real matrix.} \]

\[ \phi, \psi, l : \text{real vector of dimension } 3*n \]

During the first call on can take:

\[ \theta=\phi=\psi=l=0*\text{ones}(1,3*n). \quad p=\text{eye}(3*n,3*n) \]

\[ u0 \] : real vector of inputs (arbitrary size) (if no input take \(u0=[ \text{ ]}\)).

\[ y0 \] : vector of outputs (same dimension as \(u0\) if \(u0\) is not empty). (\(y0(1)\) is not used by rpem).

If the time domain is \((t0, t0+k-1)\) the \(u0\) vector contains the inputs

\[ u(t0), u(t0+1), \ldots, u(t0+k-1) \]

and \(y0\) the outputs

\[ y(t0), y(t0+1), \ldots, y(t0+k-1) \]

DESCRIPTION:

Recursive estimation of parameters in an ARMAX model. Uses Ljung-Soderstrom recursive prediction error method. Model considered is the following:

\[
y(t)+a(1)*y(t-1)+\ldots+a(n)*y(t-n)=b(1)*u(t-1)+\ldots+b(n)*u(t-n)+e(t)+c(1)*e(t-1)+\ldots+c(n)*e(t-n)
\]

The effect of this command is to update the estimation of unknown parameter \(\theta=[a, b, c]\) with \(a=[a(1), \ldots, a(n)], \quad b=[b(1), \ldots, b(n)], \quad c=[c(1), \ldots, c(n)]\).

OPTIONAL PARAMETERS:

\(\lambda\) : optional parameter (forgetting constant) chosen close to 1 as convergence occurs:

\(\lambda=[\lambda_{0}, \alpha, \beta]\) evolves according to:

\[ \lambda(t) = \alpha*\lambda(t-1) + \beta \]

with \(\lambda(0) = \lambda_{0}\)

\(k\) : contraction factor to be chosen close to 1 as convergence occurs.

\(k=[k_{0}, \mu, \nu]\) evolves according to:

\[ k(t) = \mu*k(t-1) + \nu \]

with \(k(0) = k_{0}\).

\(c\) : large parameter (\(c=1000\) is the default value).

OUTPUT PARAMETERS:

\(w1\) : update for \(w0\).

\(v\) : sum of squared prediction errors on \(u0, y0\). (optional).

In particular \(w1(1)\) is the new estimate of \(\theta\). If a new sample \(u1, y1\) is available the update is obtained by:

\[ \text{[w2, [v]] = rpem}(w1, u1, y1, [\lambda, [k, [c]]]) \]

Arbitrary large series can thus be treated.
2.6.51 sinc ................................. samples of sinc function

CALLING SEQUENCE:

[x]=sinc(n,fl)

PARAMETERS:

n : number of samples
fl : cut-off frequency of the associated low-pass filter in Hertz.
x : samples of the sinc function

DESCRIPTION:
Calculate n samples of the function \( \frac{\sin(2\pi fl t)}{\pi t} \) for \( t=-(n-1)/2:(n-1)/2 \) (i.e., centred around the origin).

EXAMPLE:

plot(sinc(100,0.1))

SEE ALSO: sincd 476

AUTHOR: C. B.

2.6.52 sincd ................................. digital sinc function or Direchlet kernel

CALLING SEQUENCE:

[s]=sincd(n,flag)

PARAMETERS:

n : integer
flag : if flag = 1 the function is centred around the origin; if flag = 2 the function is delayed by \( \pi/(2n) \)
s : vector of values of the function on a dense grid of frequencies

DESCRIPTION:
function which calculates the function \( \frac{\sin(Nx)}{N\sin(x)} \)

EXAMPLE:

plot(sincd(10,1))

AUTHOR: G. Le V.

2.6.53 srfaur ................................ square-root algorithm

CALLING SEQUENCE:

[p,s,t,l,rt,tt]=srfaur(h,f,g,r0,n,p,s,t,l)

PARAMETERS:

h, f, g : convenient matrices of the state-space model.
r0 : E(yk*yk').
n : number of iterations.
p : estimate of the solution after n iterations.
s, t, l: intermediate matrices for successive iterations;
rt, tt: gain matrices of the filter model after n iterations.
p, s, t, l : may be given as input if more than one recursion is desired (evaluation of intermediate values of p).

DESCRIPTION:
square-root algorithm for the algebraic Riccati equation.

EXAMPLE:

//GENERATE SIGNAL
x=%pi/10:%pi/10:102.4*%pi;
rand('seed',0);rand('normal');
y=[1;1]*sin(x)+[sin(2*x);sin(1.9*x)]+rand(2,1024);

//COMPUTE CORRELATIONS
//FINDING H,F,G with 6 states
hk=hank(20,20,c);
[H,F,G]=phc(hk,2,6);

//SOLVING RICCATI EQN
r0=c(1:2,1:2);
[P,s,t,l,Rt,Tt]=srfaur(H,F,G,r0,200);

//Make covariance matrix exactly symmetric
Rt=(Rt+Rt')/2

SEE ALSO: phc 471, faurre 456, lindquist 469

2.6.54 srkf ____________________________ square root Kalman filter

CALLING SEQUENCE:

[x1,p1]=srkf(y,x0,p0,f,h,q,r)

PARAMETERS:

f, h: current system matrices
q, r: covariance matrices of dynamics and observation noise
x0, p0: state estimate and error variance at t=0 based on data up to t=-1
y: current observation Output from the function is
x1, p1: updated estimate and error covariance at t=1 based on data up to t=0

DESCRIPTION:
square root Kalman filter algorithm

2.6.55 sskf ____________________________ steady-state Kalman filter

CALLING SEQUENCE:

[xe,pe]=sskf(y,f,h,q,r,x0)

PARAMETERS:

Scilab Group
April 1993
**Scilab Function**

**yir**

y : data in form \([y_0, y_1, ..., y_n]\), \(y\) a column vector

f : system matrix \(\dim(N \times N)\)

h : observations matrix \(\dim(M \times N)\)

q : dynamics noise matrix \(\dim(N \times N)\)

r : observations noise matrix \(\dim(M \times M)\)

\(x_0\) : initial state estimate

\(x_e\) : estimated state

\(p_e\) : steady-state error covariance

**DESCRIPTION:**
steady-state Kalman filter

**AUTHOR:** C. B.

### 2.6.56 system

**CALLING SEQUENCE:**

\([x_1, y] = \text{system}(x_0, f, g, h, q, r)\)

**PARAMETERS:**

- \(x_0\) : input state vector
- \(f\) : system matrix
- \(g\) : input matrix
- \(h\) : Output matrix
- \(q\) : input noise covariance matrix
- \(r\) : output noise covariance matrix
- \(x_1\) : output state vector
- \(y\) : output observation

**DESCRIPTION:**

define system function which generates the next observation given the old state. System recursively calculated

\[
x_1 = f \cdot x_0 + g \cdot u
\]

\[
y = h \cdot x_0 + v
\]

where \(u\) is distributed \(N(0, q)\) and \(v\) is distributed \(N(0, r)\).

**AUTHOR:** C. B.

### 2.6.57 trans

**CALLING SEQUENCE:**

\(hzt = \text{trans}(pd, zd, gd, tr_type, frq)\)

**PARAMETERS:**

- \(hzt\) : output polynomial
- \(tr_type\) : type of transformation
- \(frq\) : frequency values

**DESCRIPTION:**

function for transforming standardized low-pass filter into one of the following filters: low-pass, high-pass, band-pass, stop-band.

**AUTHOR:** C. Bunks

---

Scilab Group

April 1993

478
2.6.58  wfir  linear-phase FIR filters

CALLING SEQUENCE:

[wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)

PARAMETERS:

ftype : string : 'lp','hp','bp','sb' (filter type)
forder : Filter order (pos integer)(odd for ftype='hp' or 'sb')
 cfreq : 2-vector of cutoff frequencies (0<cfreq(1),cfreq(2)<.5) only cfreq(1) is used when
         ftype='lp' or 'hp'
wtype : Window type ('re','tr','hm','hn','kr','ch')
fpar : 2-vector of window parameters. Kaiser window fpar(1)>0 fpar(2)=0. Chebyshev window
         fpar(1)>0, fpar(2)<0 or fpar(1)<0, 0<fpar(2)<.5
wft : time domain filter coefficients
wfm : frequency domain filter response on the grid fr
fr : Frequency grid

DESCRIPTION:

Function which makes linear-phase, FIR low-pass, band-pass, high-pass, and stop-band filters using the
windowing technique. Works interactively if called with no arguments.

AUTHOR : C. Bunks

2.6.59  wiener  Wiener estimate

CALLING SEQUENCE:

[xs,ps,xf,pf]=wiener(y,x0,p0,f,g,h,q,r)

PARAMETERS:

f, g, h : system matrices in the interval [t0,tf]
f = [f0,f1,...,ff], and fk is a n x n matrix
g = [g0,g1,...,gf], and gk is a n x n matrix
h = [h0,h1,...,hf], and hk is a m x n matrix
q, r : covariance matrices of dynamics and observation noise
q = [q0,q1,...,qf], and qk is a n x n matrix
r = [r0,r1,...,rf]. and rk is a m x m matrix
x0, p0 : initial state estimate and error variance
y : observations in the interval [t0,tf]. y=[y0,y1,...,yf], and yk is a column m-vector
xs : Smoother state estimate xs= [xs0,xs1,...,xsf], and xsk is a column n-vector
ps : Error covariance of smoothed estimate ps=[p0,p1,...,pf], and pk is a n x n matrix
xf : Filtered state estimate xf= [xf0,xf1,...,xff], and xfk is a column n-vector
pf : Error covariance of filtered estimate pf=[p0,p1,...,pf], and pk is a n x n matrix

DESCRIPTION:

function which gives the Wiener estimate using the forward-backward Kalman filter formulation

AUTHOR : C. B.
2.6.60  wigner  

Calling Sequence:

\[ \text{[tab]} = \text{wigner(}x, h, \text{deltat, zp)} \]

Parameters:

- \text{tab} : wigner spectrum (lines correspond to the time variable)
- \text{x} : analyzed signal
- \text{h} : data window
- \text{deltat} : analysis time increment (in samples)
- \text{zp} : length of FFT's. \( \pi / zp \) gives the frequency increment.

Description:
function which computes the 'time-frequency' wigner spectrum of a signal.

2.6.61  window  

Calling Sequence:

\[ \text{[win}_l, \text{cwp]} = \text{window(}wtype, n, \text{par)} \]

Parameters:

- \text{wtype} : window type (re, tr, hn, hm, kr, ch)
- \text{n} : window length
- \text{par} : parameter 2-vector (kaiser window: \( \text{par(1)} = \beta > 0 \)) (Chebychev window \( \text{par} = [ \text{dp}, \text{df}] \)),
  - \( \text{dp} \) =main lobe width \( (0 < \text{dp} < .5) \), \( \text{df} \) =side lobe height \( (\text{df} > 0) \)
- \text{win} : window
- \text{cwp} : unspecified Chebyshev window parameter

Description:
function which calculates symmetric window

Author: C. B.

2.6.62  yulewalk  

Calling Sequence:

\( \text{Hz} = \text{yulewalk(}N, \text{frq, mag)} \)

Parameters:

- \text{N} : integer (order of desired filter)
- \text{frq} : real row vector (non-decreasing order), frequencies.
- \text{mag} : non negative real row vector (same size as frq), desired magnitudes.
- \text{Hz} : filter \( B(z) / A(z) \)

Description:
\( \text{Hz} = \text{yulewalk(}N, \text{frq, mag)} \) finds the N-th order iir filter
\[
\begin{align*}
\text{H}(z) = \frac{\text{B}(z)}{\text{A}(z)} &= \frac{b(1)z^{n-1} + b(2)z^{n-2} + \ldots + b(n)}{z^{n-1} + a(2)z^{n-2} + \ldots + a(n)}
\end{align*}
\]

which matches the magnitude frequency response given by vectors \( \text{frq} \) and \( \text{mag} \). Vectors \( \text{frq} \) and \( \text{mag} \) specify the frequency and magnitude of the desired frequency response. The frequencies in \( \text{frq} \) must be between 0.0 and 1.0, with 1.0 corresponding to half the sample rate. They must be in increasing order and start with 0.0 and end with 1.0.

**EXAMPLE:**

\[
\begin{align*}
\text{f} &= [0, 0.4, 0.4, 0.6, 0.6, 1]; \\
\text{H} &= [0, 0, 1, 1, 0, 0]; \\
\text{Hz} &= \text{yulewalk}(8, \text{f}, \text{H}); \\
\text{fs} &= 1000; \\
\text{fhz} &= \text{f} \times \text{fs}/2; \\
\text{xbasc}(0); \\
\text{xset('window', 0); plot2d(\text{fhz}', \text{H}');} \\
\text{xtitle('Desired Frequency Response (Magnitude)');} \\
[\text{frq}, \text{repf}] &= \text{repfreq} \left( \text{Hz}, 0:0.001:0.5 \right); \\
\text{xbasc}(1); \\
\text{xset('window', 1); plot2d(\text{fs} \times \text{frq}', \text{abs}(\text{repf}'));} \\
\text{xtitle('Obtained Frequency Response (Magnitude)');}
\end{align*}
\]

### 2.6.63 zpbutt

**Butterworth analog filter**

**CALLING SEQUENCE:**

\[ [\text{poles}, \text{gain}] = \text{zpbutt}(n, \text{omegac}) \]

**PARAMETERS:**

- \( n \) : integer (filter order)
- \( \text{omegac} \) : real (cut-off frequency in Hertz)
- \( \text{poles} \) : resulting filter poles
- \( \text{gain} \) : resulting gain of filter

**DESCRIPTION:**

computes the poles of a Butterworth analog filter of order \( n \) and cutoff frequency \( \text{omegac} \). The transfer function is given by:

\[ H(s) = \frac{\text{gain}}{\text{real}(\text{poly}(\text{poles}, 's'))} \]

**AUTHOR:** F.D.

### 2.6.64 zpch1

**Chebyshev analog filter**

**CALLING SEQUENCE:**

\[ [\text{poles}, \text{gain}] = \text{zpch1}(n, \epsilon, \text{omegac}) \]

**PARAMETERS:**

- \( n \) : integer (filter order)
- \( \epsilon \) : real : ripple in the pass band \( 0 < \epsilon < 1 \)
- \( \text{omegac} \) : real : cut-off frequency in Hertz
- \( \text{poles} \) : resulting filter poles
- \( \text{gain} \) : resulting filter gain

**DESCRIPTION:**

Poles of a Type 1 Chebyshev analog filter. The transfer function is given by:

\[ H(s) = \frac{\text{gain}}{\text{poly}(\text{poles}, 's')} \]

**AUTHOR:** F.D.
2.6.65  zpch2  _________________  Chebyshev analog filter

CALLING SEQUENCE:

\[ [\text{zeros}, \text{poles}, \text{gain}] = \text{zpch2}(n, A, \text{omegar}) \]

PARAMETERS:

- \text{n} : integer : filter order
- \text{A} : real : attenuation in stop band (\(A>1\))
- \text{omegar} : real : cut-off frequency in Hertz
- \text{zeros} : resulting filter zeros
- \text{poles} : resulting filter poles
- \text{gain} : Resulting filter gain

DESCRIPTION:

Poles and zeros of a type 2 Chebyshev analog filter. \(H(s)=\text{gain}\times\text{poly(}\text{zeros,}'s'\text{)}/\text{poly(}\text{poles,}'s'\text{)}\)

AUTHOR: F.D.

2.6.66  zpell  _____________________________  lowpass elliptic filter

CALLING SEQUENCE:

\[ [\text{zeros}, \text{poles}, \text{gain}] = \text{zpell}(\epsilon, A, \text{omegac}, \text{omegar}) \]

PARAMETERS:

- \text{epsilon} : real : ripple of filter in pass band (\(0<\epsilon<1\))
- \text{A} : real : attenuation of filter in stop band (\(A>1\))
- \text{omegac} : real : pass band cut-off frequency in Hertz
- \text{omegar} : real : stop band cut-off frequency in Hertz
- \text{zeros} : resulting zeros of filter
- \text{poles} : resulting poles of filter
- \text{gain} : resulting gain of filter

DESCRIPTION:

Poles and zeros of prototype lowpass elliptic filter. \(\text{gain}\) is the gain of the filter

SEE ALSO: ell1mag 455, eqiir 456

AUTHOR: F.D.
2.7 Polynomial calculations
2.7.1 bezout __________________________ Bezout equation for polynomials

CALLING SEQUENCE:

\[ [\text{thegcd}, U] = \text{bezout}(\text{p1}, \text{p2}) \]

PARAMETERS:

\( \text{p1}, \text{p2} \) : two real polynomials

DESCRIPTION:

\[ [\text{thegcd}, U] = \text{bezout}(\text{p1}, \text{p2}) \]
computes \( \text{GCD} \) \( \text{thegcd} \) of \( \text{p1} \) and \( \text{p2} \) and in addition a (2x2) unimodular matrix \( U \) such that:
\[ [\text{p1}, \text{p2}] \cdot U = [\text{thegcd}, 0] \]
The lcm of \( \text{p1} \) and \( \text{p2} \) is given by:
\( \text{p1} \cdot U(1, 2) \) (or \( -\text{p2} \cdot U(2, 2) \))

EXAMPLE:

\[
\begin{align*}
\text{x} &= \text{poly}(0, 'x'); \\
\text{p1} &= (\text{x}+1) \cdot (\text{x}-3)^5; \text{p2} = (\text{x}-2) \cdot (\text{x}-3)^3; \\
[\text{thegcd}, U] &= \text{bezout}(\text{p1}, \text{p2}) \\
\text{det}(U) &= \\
\text{thelcm} &= \text{p1} \cdot U(1, 2) \\
\text{lcm} &= \text{[p1, p2]} \\
\end{align*}
\]

SEE ALSO: poly 65, roots 496, simp 497, clean 484, lcm 492

2.7.2 clean ___________________________ cleans matrices (round to zero small entries)

CALLING SEQUENCE:

\[ [B] = \text{clean}(A [, \text{epsa} [, \text{epsr}]]) \]

PARAMETERS:

\( A \) : a numerical matrix (scalar, polynomial, sparse...)
\( \text{epsa}, \text{epsr} \) : real numbers (default values resp. 1.d-10 and 1.d-10)

DESCRIPTION:

This function eliminates (i.e. set to zero) all the coefficients with absolute value < \( \text{epsa} \) and relative value < \( \text{epsr} \) (relative means relative w.r.t. 1-norm of coefficients) in a polynomial (possibly matrix polynomial or rational matrix).
Default values are \( \text{epsa}=1.d-10 \) and \( \text{epsr}=1.d-10 \);
For a constant (non polynomial) matrix \( \text{clean}(A, \text{epsa}) \) sets to zero all entries of \( A \) smaller than \( \text{epsa} \).

EXAMPLE:

\[
\begin{align*}
\text{x} &= \text{poly}(0, 'x'); \\
\text{w} &= [\text{x}, 1, 2+x; 3+x, 2-x, x^2; 1, 2, 3+x]/3; \\
\text{w} \cdot \text{inv}(\text{w}) &= \\
\text{clean}(\text{w} \cdot \text{inv}(\text{w})) &= \\
\end{align*}
\]
**2.7.3 cmndred**

**Calling Sequence:**

\[ [n,d]=\text{cmndred}(\text{num,den}) \]

**Parameters:**

num, den : two polynomial matrices of same dimensions

**Description:**

\[ [n,d]=\text{cmndred}(\text{num,den}) \]
computes a polynomial matrix \( n \) and a common denominator polynomial \( d \) such that:

\[ \frac{n}{d} = \frac{\text{num}}{\text{den}} \]

The rational matrix defined by \( \frac{\text{num}}{\text{den}} \) is \( \frac{n}{d} \)

**See Also:** simp 497, clean 484

**2.7.4 coeff**

**Calling Sequence:**

\[ [C]=\text{coeff}(\text{Mp} [,v]) \]

**Parameters:**

Mp : polynomial matrix  
\( v \) : integer (row or column) vector of selected degrees  
\( C \) : big matrix of the coefficients

**Description:**

\[ C=\text{coeff}(\text{Mp}) \]
returns in a big matrix \( C \) the coefficients of the polynomial matrix \( Mp \). \( C \) is partitioned as \( C=\{C0,C1,\ldots,Ck\} \) where the \( Ci \) are arranged in increasing order \( k = \maxi(\text{degree}(Mp)) \)

\[ C=\text{coeff}(\text{Mp},v) \]
returns the matrix of coefficients with degree in \( v \). (\( v \) is a row or column vector).

**See Also:** poly 65, degree 486, inv_coeff 54

**2.7.5 coffg**

**Calling Sequence:**

\[ [\text{Ns},d]=\text{coffg}(\text{Fs}) \]

**Parameters:**

Fs : square polynomial matrix

**Description:**

\( \text{coffg} \) computes \( \text{Fs}^{-1} \) where \( \text{Fs} \) is a polynomial matrix by co-factors method.  
\( \text{Fs} \) inverse = \( \text{Ns}/d \)  
\( d \) = common denominator; \( \text{Ns} \) = numerator (a polynomial matrix)  
(For large matrices, be patient...results are generally reliable)

**Example:**

\[ s=\text{poly}(0,'s') \]
\[ a=[ s, s^2+1; s \ s^2-1]; \]
\[ [a1,d]=\text{coffg}(a); \]
\[ (a1/d)-\text{inv}(a) \]

**See Also:** determ 487, detr 487, invr 491, penlaur 524, glever 512

**Author:** F. D.

April 1993
### 2.7.6 colcompr  

column compression of polynomial matrix

**CALLING SEQUENCE:**

\[ Y, r, a \] = colcompr(A);

**PARAMETERS:**

- **A** : polynomial matrix
- **Y** : square polynomial matrix (right unimodular basis)
- **rk** : normal rank of A
- **Ac** : \( Ac = A*Y \), polynomial matrix

**DESCRIPTION:**

column compression of polynomial matrix A (compression to the left)

**EXAMPLE:**

```plaintext
s=poly(0,'s');
p=[s;s*(s+1)^2;2*s^2+s^3];
[Y,rk,ac]=colcompr(p*p');
p*p'*Y
```

**SEE ALSO:** rowcompr 496

### 2.7.7 degree  
degree of polynomial matrix

**CALLING SEQUENCE:**

\[ D = \text{degree}(M) \]

**PARAMETERS:**

- **M** : polynomial matrix
- **D** : integer matrix

**DESCRIPTION:**

returns the matrix of highest degrees of M.

**SEE ALSO:** poly 65, coeff 485, clean 484

### 2.7.8 denom  
denominator

**CALLING SEQUENCE:**

den=denom(r)

**PARAMETERS:**

- **r** : rational or polynomial or constant matrix
- **den** : polynomial matrix

**DESCRIPTION:**

den=denom(r) returns the denominator of a rational matrix.

Since rationals are internally represented as \( r = \text{list}(['r','num','den','dt'], \text{num}, \text{den}, []) \), denom(r) is the same as \( r(3) \) or \( r('den') \).

**SEE ALSO:** numer 493
2.7.9 derivat ______________________________ rational matrix derivative

CALLING SEQUENCE:
\[ pd = \text{derivat}(p) \]

PARAMETERS:
\( p \): polynomial or rational matrix

DESCRIPTION:
computes the derivative of the polynomial or rational function matrix w.r.t the dummy variable.

EXAMPLE:
\[
\begin{align*}
\text{s} &= \text{poly}(0, 's'); \\
\text{derivat}(1/\text{s}) & \quad // -1/s^2;
\end{align*}
\]

2.7.10 determ ____________________________ determinant of polynomial matrix

CALLING SEQUENCE:
\[ \text{res} = \text{determ}(W [,k]) \]

PARAMETERS:
\( W \): real square polynomial matrix
\( k \): integer (upper bound for the degree of the determinant of \( W \))

DESCRIPTION:
\[
\text{res} = \text{determ}(W [,k])
\]
returns the determinant of a real polynomial matrix (computation made by FFT).
\( k \) is an integer larger than the actual degree of the determinant of \( W \).
The default value of \( k \) is the smallest power of 2 which is larger than \( n \times \maxi(degree(W)) \).
Method: evaluate the determinant of \( W \) for the Fourier frequencies and apply inverse FFT to the coefficients of the determinant.

EXAMPLE:
\[
\begin{align*}
\text{s} &= \text{poly}(0, 's'); \\
\text{w} &= \text{s} \times \text{rand}(10, 10); \\
\text{determ(w)} \\
\text{det(coff(w,1))} &= s^{10}
\end{align*}
\]

SEE ALSO: det 507, detr 487, coffg 485

AUTHOR: F.D.

2.7.11 detr ______________________________ polynomial determinant

CALLING SEQUENCE:
\[ d = \text{detr}(h) \]

PARAMETERS:
\( h \): polynomial or rational square matrix

DESCRIPTION:
\[ d = \text{detr}(h) \] returns the determinant \( d \) of the polynomial or rational function matrix \( h \). Based on Leverrier’s algorithm.

SEE ALSO: det 507, determ 487
**2.7.12  diophant  ________________________________________________________________________  diophantine (Bezout) equation**

**CALLING SEQUENCE :**

\[
[x, \text{err}] = \text{diophant}(\text{p1p2}, b)
\]

**PARAMETERS :**

- \text{p1p2} : polynomial vector \text{p1p2} = [p1 \ p2]
- \text{b} : polynomial
- \text{x} : polynomial vector \{x1; x2\}

**DESCRIPTION :**

\text{diophant}  solves the bezout equation:

\[ p1 \times x1 + p2 \times x2 = b \]

with \text{p1p2} a polynomial vector. If the equation is not solvable \text{err} = \|p1x1 + p2x2 - b\| / \|b\| else \text{err} = 0

**EXAMPLE :**

\begin{verbatim}
s=poly(0,'s');p1=(s+3)^2;p2=(1+s);
x1=s;x2=(2+s);
[x,err]=diophant([p1,p2],p1*x1+p2*x2);
p1*x1+p2*x2-p1*x(1)-p2*x(2)
\end{verbatim}

**2.7.13  factors  ________________________________________________________________________  numeric real factorization**

**CALLING SEQUENCE :**

\[
[\text{lnum}, \text{g}] = \text{factors}(\text{pol} [, \text{flag}])
\]

\[
[\text{lnum}, \text{lden}, \text{g}] = \text{factors}(\text{rat} [, \text{flag}])
\]

\[
\text{rat}=\text{factors}(\text{rat}, \text{flag})
\]

**PARAMETERS :**

- \text{pol} : real polynomial
- \text{rat} : real rational polynomial (rat=pol1/pol2)
- \text{lnum} : list of polynomials (of degrees 1 or 2)
- \text{lden} : list of polynomials (of degrees 1 or 2)
- \text{g} : real number
- \text{flag} : character string 'c' or 'd'

**DESCRIPTION :**

returns the factors of polynomial \text{pol} in the list \text{lnum} and the "gain" \text{g}.

One has \text{pol} = \text{g} times product of entries of the list \text{lnum} (if \text{flag} is not given). If \text{flag} = 'c' is given, then one has \(|\text{pol}(i \ \omega)| = |\text{g} \times \text{prod(\text{lnum},j(i \ \omega))}|.\) If \text{flag} = 'd' is given, then one has \(|\text{pol}(\exp(i \ \omega))| = |\text{g} \times \text{prod(\text{lnum},i(\exp(i \ \omega)))}|.\) If argument of factors is a 1x1 rational \text{rat}=pol1/pol2, the factors of the numerator \text{pol1} and the denominator \text{pol2} are returned in the lists \text{lnum} and \text{lden} respectively.

The "gain" is returned as \text{g} i.e. one has: \text{rat} = \text{g} times (product entries in \text{lnum}) / (product entries in \text{lden}).

If \text{flag} is 'c' (resp. 'd'), the roots of \text{pol} are reflected wrt the imaginary axis (resp. the unit circle), i.e. the factors in \text{lnum} are stable polynomials.

Same thing if \text{factors} is invoked with a rational arguments: the entries in \text{lnum} and \text{lden} are stable polynomials if \text{flag} is given. \text{R2}=\text{factors}(\text{R1}, 'c') or \text{R2}=\text{factors}(\text{R1}, 'd') with \text{R1} a rational function or SISO \text{syslin} list then the output \text{R2} is a transfer with stable numerator and denominator and with same magnitude as \text{R1} along the imaginary axis ("c") or unit circle ("d").

**EXAMPLE :**

\begin{verbatim}
\end{verbatim}
n=poly([0.2,2,5],'z');
d=poly([0.1,0.3,7],'z');
R=syslin('d',n,d);
R1=factors(R,'d')
roots(R1('num'))
roots(R1('den'))
w=exp(2*%i*%pi*[0:0.1:1]);
norm(abs(horner(R1,w))-abs(horner(R,w)))

See Also: simp 497

2.7.14 gcd ............................................. gcd calculation

Calling Sequence:

[pgcd,U]=gcd(p)

Parameters:

p : polynomial row vector p=[p1,...,pn]

Description:

[pgcd,u]=gcd(p) computes the gcd of components of p and a unimodular matrix (with polynomial inverse) U, with minimal degree such that

p*U=[0 ... 0 pgcd]

Example:

s=poly(0,'s');
p=[s,s*(s+1)^2,2*s^2+s^3];
[pgcd,u]=gcd(p);
p*u

See Also: bezout 484, lcm 492, hermit 489

2.7.15 hermit ........................................... Hermite form

Calling Sequence:

[Ar,U]=hermit(A)

Parameters:

A : polynomial matrix
Ar : triangular polynomial matrix
U : unimodular polynomial matrix

Description:

Hermite form: U is an unimodular such that A*U is in Hermite triangular form:
The output variable is Ar=A*U.
Warning: Experimental version

Example:

s=poly(0,'s');
p=[s, s*(s+1)^2, 2*s^2+s^3];
[Ar,U]=hermit(p'*p);
clean(p'*p*U), det(U)

See Also: hrmt 490, htrianr 491
2.7.16  horner __________________________ polynomial/rational evaluation

CALLING SEQUENCE :

\texttt{horner(P,x)}

PARAMETERS :

\(P\) : polynomial or rational matrix
\(x\) : real number or polynomial or rational

DESCRIPTION :

evaluates the polynomial or rational matrix \(P = P(s)\) when the variable \(s\) of the polynomial is replaced by \(x\):
\[
\text{horner}(P,x) = P(x)
\]
Example (Bilinear transform): Assume \(P = P(s)\) is a rational matrix then the rational matrix \(P((1+s)/(1-s))\) is obtained by \(\text{horner}(P,(1+s)/(1-s))\).
To evaluate a rational matrix at given frequencies use preferably the \texttt{freq} primitive.

EXAMPLES :

\[
s = \text{poly}(0,'s'); M = [s,1/s];
\]
\[
\text{horner}(M,1) \quad \text{horner}(M,\text{i}) \quad \text{horner}(M,1/s)
\]

SEE ALSO:  \texttt{freq 339},  \texttt{repfreq 355},  \texttt{evstr 35}

2.7.17  hrmt ________________________________ gcd of polynomials

CALLING SEQUENCE :

\[
[\text{pg},U] = \text{hrmt}(v)
\]

PARAMETERS :

\(v\) : row of polynomials i.e. \(1xk\) polynomial matrix
\(pg\) : polynomial
\(U\) : unimodular matrix polynomial

DESCRIPTION :

\([\text{pg},U] = \text{hrmt}(v)\) returns a unimodular matrix \(U\) and \(pg = \text{gcd}\) of row of polynomials \(v\) such that \(v*U = [pg,0]\).

EXAMPLE :

\[
x = \text{poly}(0,'x');
v = [x*(x+1), x^2*(x+1), (x-2)*(x+1), (3*x^2+2)*(x+1)];
[pg,U] = \text{hrmt}(v); U = \text{clean}(U)
det(U)
\]

SEE ALSO:  \texttt{gcd 489},  \texttt{htrianr 491}
2.7.18  htrianr  ________________ triangularization of polynomial matrix

CALLING SEQUENCE:

[Ar,U,rk]=htrianr(A)

PARAMETERS:

A : polynomial matrix
Ar : polynomial matrix
U : unimodular polynomial matrix
rk : integer, normal rank of A

DESCRIPTION:
triangularization of polynomial matrix A.
A is [m,n], m <= n.
Ar=A*U
Warning: there is an elimination of "small" terms (see function code).

EXAMPLE:

x=poly(0,'x');
M=[x;x^2;2+x^3]*[1,x-2,x^4];
[Mu,U,rk]=htrianr(M)
det(U)
M*U(:,1:2)

SEE ALSO:  hrmr 490,  colcompr 486

2.7.19  invr  __________________________ inversion of (rational) matrix

CALLING SEQUENCE:

F = invr(H)

PARAMETERS:

H : polynomial or rational matrix
F : polynomial or rational matrix

DESCRIPTION:
If H is a polynomial or rational function matrix, invr computes H^(-1) using Leverrier's algorithm (see function code)

EXAMPLE:

s=poly(0,'s')
H=[s,s*s+2;1-s,1+s]; invr(H)
[Num,den]=coffg(H);Num/den
H=[1/s,(s+1);1/(s+2), (s+3)/s];invr(H)

SEE ALSO:  glever 512,  coffg 485,  inv 516

Scilab Group  April 1993  491
2.7.20 lcm

CALLING SEQUENCE:

\[ [\text{pp}, \text{fact}] = \text{lcm}(p) \]

PARAMETERS:

\( p \):
- fact: polynomial vector
\( \text{pp} \):
- polynomial

DESCRIPTION:

\( \text{pp} = \text{lcm}(p) \) computes the lcm \( \text{pp} \) of polynomial vector \( p \).

\( [\text{pp}, \text{fact}] = \text{lcm}(p) \) computes in addition the vector \( \text{fact} \) such that:

\( p.*\text{fact} = \text{pp}.*\text{ones}(p) \)

EXAMPLE:

\[
\begin{align*}
s &= \text{poly}(0, 's'); \\
p &= \left[ s, s*(s+1)^2, s^2*(s+2) \right]; \\
[\text{pp}, \text{fact}] &= \text{lcm}(p); \\
p.*\text{fact}, \text{pp}
\end{align*}
\]

SEE ALSO: gcd 489, bezout 484

2.7.21 lcmdiag

CALLING SEQUENCE:

\[ [\text{N}, \text{D}] = \text{lcmdiag}(H) \]
\[ [\text{N}, \text{D}] = \text{lcmdiag}(H, \text{flag}) \]

PARAMETERS:

\( H \):
- rational matrix
\( N \):
- polynomial matrix
\( D \):
- diagonal polynomial matrix
\text{flag}:
- character string: 'row' or 'col' (default)

DESCRIPTION:

\[ [\text{N}, \text{D}] = \text{lcmdiag}(H, \text{'row'}) \] computes a factorization \( D*H=N \), i.e. \( H=D*(-1)*N \) where \( D \) is a diagonal matrix with \( D(k,k) = \text{lcm} \) of kth row of \( H('\text{den}) \).

\[ [\text{N}, \text{D}] = \text{lcmdiag}(H) \] or \[ [\text{N}, \text{D}] = \text{lcmdiag}(H, \text{'col}) \] returns \( H=N*D*(-1) \) with diagonal \( D \) and \( D(k,k) = \text{lcm} \) of kth col of \( H('\text{den}) \)

EXAMPLE:

\[
\begin{align*}
s &= \text{poly}(0, 's'); \\
H &= \left[ 1/s, (s+2)/s/(s+1)^2; 1/(s^2*(s+2)), 2/(s+2) \right]; \\
[\text{N}, \text{D}] &= \text{lcmdiag}(H); \\
N/D-H
\end{align*}
\]

SEE ALSO: lcm 492, gcd 489, bezout 484

Scilab Group
April 1993
492
2.7.22  ldiv  polynomial matrix long division

CALLING SEQUENCE :

[x]=ldiv(n,d,k)

PARAMETERS :

n, d : two real polynomial matrices
k : integer

DESCRIPTION :

x=ldiv(n,d,k) gives the k first coefficients of the long division of n by d i.e. the Taylor expansion of the rational matrix \[ \frac{n_{ij}(z)}{d_{ij}(z)} \] near infinity. Coefficients of expansion of \( \frac{n_{ij}}{d_{ij}} \) are stored in \( x((i-1)*n+k,j) \) \( k=1:n \)

EXAMPLE :

wss=ssrand(1,1,3);[a,b,c,d]=abcd(wss);
wtf=ss2tf(wss);
x1=ldiv(numer(wtf),denom(wtf),5)
x2=[c*b;c*a*b;c*a^2*b;c*a^3*b;c*a^4*b]
wssbis=markp2ss(x1',5,1,1);
wtfbis=clean(ss2tf(wssbis))
x3=ldiv(numer(wtfbis),denom(wtfbis),5)

SEE ALSO:  arl2 321,  markp2ss 347,  pdiv 493

2.7.23  numer  numerator

CALLING SEQUENCE :

NUM=numer(R)

PARAMETERS :

R : rational matrix

DESCRIPTION :

Utility function. NUM=numer(R) returns the numerator NUM of a rational function matrix R (R may be also a constant or polynomial matrix). numer(R) is equivalent to R(2) or R('num')

SEE ALSO:  denom 486

2.7.24  pdiv  polynomial division

CALLING SEQUENCE :

[R,Q]=pdiv(P1,P2)

PARAMETERS :

P1 : polynomial matrix
P2 : polynomial or polynomial matrix
\textbf{R, Q : two polynomial matrices}

\textbf{DESCRIPTION :}
Element-wise euclidan division of the polynomial matrix \( P_1 \) by the polynomial \( P_2 \) or by the polynomial matrix \( P_2 \). \( R_{ij} \) is the matrix of remainders, \( Q_{ij} \) is the matrix of quotients and \( P_{1ij} = Q_{ij}P_2 + Q_{ij} \) or \( P_{1ij} = Q_{ij}P_{2ij} + Q_{ij} \).

\textbf{EXAMPLE :}
\begin{verbatim}
x=poly(0,'x'); p1=(1+x^2)*(1-x);p2=1-x; [r,q]=pdiv(p1,p2) p2*q-r-p1
\end{verbatim}
\begin{verbatim}
%2 =
\end{verbatim}

\begin{verbatim}
\end{verbatim}

\textbf{SEE ALSO :} ldiv 493, gcd 489

\section*{2.7.25 \texttt{pol2des} \hfill polynomial matrix to descriptor form}

\textbf{CALLING SEQUENCE :}
\[ [N,B,C]=\text{pol2des}(D_s) \]

\textbf{PARAMETERS :}
\begin{itemize}
  \item \texttt{D_s} : polynomial matrix
  \item \texttt{N, B, C} : three real matrices
\end{itemize}

\textbf{DESCRIPTION :}
Given the polynomial matrix \( D_s=D_0+D_1s+D_2s^2+\ldots+D_k s^k \), \texttt{pol2des} returns three matrices \( N, B, C \), with \( N \) nilpotent such that:
\[ D_s = C (sN-\text{eye()} )^{-1} B \]

\textbf{EXAMPLE :}
\begin{verbatim}
s=poly(0,'s'); G=[1,s;1+s^2,3*s^3];[N,B,C]=pol2des(G); G1=clean(C*inv(s*N-eye())*B),G2=numer(G1)
\end{verbatim}
\begin{verbatim}
%2 =
\end{verbatim}

\begin{verbatim}
%3 =
\end{verbatim}

\textbf{SEE ALSO :} ss2des 361, tf2des 390

\section*{AUTHOR : F.D.}

\section*{2.7.26 \texttt{pol2str} \hfill polynomial to string conversion}

\textbf{CALLING SEQUENCE :}
\[ [\text{str}]=\text{pol2str}(p) \]

\textbf{PARAMETERS :}
\begin{itemize}
  \item \texttt{p} : real polynomial
  \item \texttt{str} : character string
\end{itemize}

\textbf{DESCRIPTION :}
converts polynomial to character string (utility function).

\textbf{SEE ALSO :} string 284, pol2tex 664
2.7.27   polfact ________________________________ minimal factors

CALLING SEQUENCE :

[f]=polfact(p)

PARAMETERS :

p : polynomial
f : vector [f0 f1 ... fn] such that p=prod(f)

DESCRIPTION :

f=polfact(p) returns the minimal factors of p i.e. f=[f0 f1 ... fn] such that p=prod(f)

SEE ALSO :  lcm 492, cmndred 485, factors 488

2.7.28   residu ________________________________ residue

CALLING SEQUENCE :

[V]=residu(P,Q1,Q2)

PARAMETERS :

P, Q1, Q2 : polynomials or matrix polynomials with real or complex coefficients.

DESCRIPTION :

V=residu(P,Q1,Q2) returns the matrix V such that V(i,j) is the sum of the residues of the rational fraction P(i,j)/(Q1(i,j)*Q2(i,j)) calculated at the zeros of Q1(i,j). Q1(i,j) and Q2(i,j) must not have any common root.

EXAMPLE :

s=poly(0,'s');
H=[s/(s+1)^2,1/(s+2)];N=numer(H);D=denom(H);
w=residu(N.*horner(N,-s),D,horner(D,-s)); //N(s) N(-s) / D(s) D(-s)
sqrt(sum(w)) //This is H2 norm
h2norm(tf2ss(H))

//
p=(s-1)*(s+1)*(s+2)*(s+10);a=(s-5)*(s-1)*(s^2)*)((s+1/2)**2);
b=(s-3)*(s+2/5)*(s+3);
residu(p,a,b)+531863/4410 //Exact
z=poly(0,'z');a=z^3+0.7*z^2+0.5*z-0.3;b=z^3+0.3*z^2+0.2*z+0.1;
atild=gtild(a,'d');btild=gtild(b,'d');
residu(b*btild,z*a,atild)-2.9488038 //Exact
a=a+0*%i;b=b+0*%i;
real(residu(b*btild,z*a,atild)-2.9488038) //Complex case

SEE ALSO :  pfss 353, bdiag 502, roots 496, poly 65, gtild 380

AUTHOR : F.D.

Scilab Group          April 1993

495
2.7.29 roots ................................. roots of polynomials

CALLING SEQUENCE :

[x]=roots(p)

PARAMETERS :

p : polynomial with real or complex coefficients

DESCRIPTION :

x=roots(p) returns in the complex vector x the roots of the polynomial p. Degree of p must be <=100.

EXAMPLE :

p=poly([0,10,1+%i,1-%i],’x’);
roots(p)
A=rand(3,3);roots(poly(A,’x’)) // Evals by characteristic polynomial spec(A)

SEE ALSO: poly 65

2.7.30 routh_t ................................ Routh’s table

CALLING SEQUENCE :

r=routh_t(h [,k]).

PARAMETERS :

h : square rational matrix

DESCRIPTION :

r=routh_t(h,k) computes Routh’s table of denominator of the system described by transfer matrix SISO h with the feedback by the gain k.
If k=poly(0,’k’) we will have a polynomial matrix with dummy variable k, formal expression of the Routh table.

2.7.31 rowcompr ............................. row compression of polynomial matrix

CALLING SEQUENCE :

[X,rk,Ac]=rowcompr(A)

PARAMETERS :

A : polynomial matrix
Y : square polynomial matrix (left unimodular basis)
rk : normal rank of A
Ac : Ac=X*A, polynomial matrix

DESCRIPTION :

row compression of polynomial matrix A .
X is a left polynomial unimodular basis which row compressed thee rows of A. rk is the normal rank of A.
Warning: elimination of ”small” terms (use with care!).

SEE ALSO: colcompr 486
2.7.32 sfact  ___________________________ discrete time spectral factorization

**CALLING SEQUENCE:**

\[ F = \text{sfact}(P) \]

**PARAMETERS:**

- \( P \): real polynomial matrix

**DESCRIPTION:**

Finds \( F \), a spectral factor of \( P \). \( P \) is a polynomial matrix such that each root of \( P \) has a mirror image with respect to the unit circle. Problem is singular if a root is on the unit circle.

\( \text{sfact}(P) \) returns a polynomial matrix \( F(z) \) which is antistable and such that \( P = F(z) \times F(1/z) \times z^n \)

For scalar polynomials a specific algorithm is implemented. Algorithms are adapted from Kucera’s book.

**EXAMPLE:**

```
//Simple polynomial example
z=poly(0,'z');
p=(z-1/2)*(2-z)
w=sfact(p);
w*numer(horner(w,1/z))
//matrix example
F1=[z-1/2,z+1/2,z^2+2;z1,z,-z;z^3+2*z,z,1/2-z];
P=F1*gtild(F1,'d'); //P is symmetric
F=sfact(P)
roots(det(P))
roots(det(gtild(F,'d'))) //The stable roots
roots(det(F)) //The antistable roots
clean(P-F*gtild(F,'d'))
//Example of continuous time use
s=poly(0,'s');
p=-3*(s+(1+%i))*(s+(1-%i))*(s+0.5)*(s-0.5)*(s-(1+%i))*(s-(1-%i));p=real(p);
//p(s) = polynomial in s^2 , looks for stable f such that p=f(s)*f(-s)
w=horner(p,(1-s)/(1+s)); // bilinear transform w=p((1-s)/(1+s))
wn=numer(w); //take the numerator
fn=sfact(wn);=numer(horner(fn,(1-s)/(s+1))); //Factor and back transform
f=f/sqrt(horner(f*gtild(f,'c'),0));f=f*sqrt(horner(p,0)); //normalization
roots(f) //f is stable
```

**SEE ALSO:** gtilde 380, fspecg 377

2.7.33 simp  _____________________________ rational simplification

**CALLING SEQUENCE:**

\[ [N1,D1] = \text{simp}(N,D) \]

\[ H1 = \text{simp}(H) \]

**PARAMETERS:**

- \( N, D \): real polynomials or real matrix polynomials
**H**: rational matrix (i.e matrix with entries \( n/d \), \( n \) and \( d \) real polynomials)

**DESCRIPTION:**
\([n_1,d_1]=\text{simp}(n,d)\) calculates two polynomials \( n_1 \) and \( d_1 \) such that \( n_1/d_1 = n/d \).

If \( N \) and \( D \) are polynomial matrices the calculation is performed element-wise.

\( H1=\text{simp}(H) \) is also valid (each entry of \( H \) is simplified in \( H1 \)).

Caution:
- no threshold is given i.e. \( \text{simp} \) cannot forces a simplification.
- For linear dynamic systems which include integrator(s) simplification changes the static gain. (\( H(0) \) for continuous systems or \( H(1) \) for discrete systems)
- For complex data, \( \text{simp} \) returns its input(s).
- Rational simplification is called after nearly each operations on rationals. It is possible to toggle simplification on or off using \( \text{simp} \_\text{mode} \) function.

**EXAMPLES:**

\[
s=poly(0,'s');
[n,d]=\text{simp}((s+1)*(s+2),(s+1)*(s-2))
\]

\[
\text{simp\_mode}\%(\text{F}); hns=s/s
\text{simp\_mode}\%(\text{T}); hns=s/s
\]

**SEE ALSO:** roots 496, trfmod 228, poly 65, clean 484, simp_mode 498

**2.7.34 simp_mode ____________________________ toggle rational simplification**

**CALLING SEQUENCE:**

\[
\text{mod}=\text{simp\_mode}()
\text{simp\_mode}(\text{mod})
\]

**PARAMETERS:**

\( \text{mod} \) : a boolean

**DESCRIPTION:**
Rational simplification is called after nearly each operations on rationals. It is possible to toggle simplification on or off using \( \text{simp\_mode} \) function.

\[
\text{simp\_mode}\%(\text{T}) \quad \text{set rational simplification mode on}
\text{simp\_mode}\%(\text{F}) \quad \text{set rational simplification mode off}
\]

\( \text{mod}=\text{simp\_mode}() \) returns in \( \text{mod} \) the current rational simplification mode

**EXAMPLES:**

\[
s=poly(0,'s');
\text{mod}=\text{simp\_mode}()
\text{simp\_mode}\%(\text{F}); hns=s/s
\text{simp\_mode}\%(\text{T}); hns=s/s
\text{simp\_mode}(\text{mod});
\]

**SEE ALSO:** simp 497
2.7.35  sylm  Sylvester matrix

CALLING SEQUENCE :

\[
[S]=\text{sylm}(a,b)
\]

PARAMETERS :

\[
a,b : \text{two polynomials}
\]
\[
S : \text{matrix}
\]

DESCRIPTION :

\[\text{sylm}(a,b)\] gives the Sylvester matrix associated to polynomials \(a\) and \(b\), i.e. the matrix \(S\) such that:
\[
\text{coeff}(a\times x + b\times y)' = S \times [\text{coeff}(x)';\text{coeff}(y)'].
\]
Dimension of \(S\) is equal to \(\text{degree}(a)+\text{degree}(b)\).
If \(a\) and \(b\) are coprime polynomials then
\[
\text{rank}(\text{sylm}(a,b)) = \text{degree}(a)+\text{degree}(b))
\]
and the instructions
\[
\begin{align*}
\text{u} &= \text{sylm}(a,b) \backslash \text{eye}(\text{na+nb},1) \\
\text{x} &= \text{poly}(\text{u}(1:nb),'z','\text{coeff}') \\
\text{y} &= \text{poly}(\text{u}(\text{nb+1:na+nb}),'z','\text{coeff}')
\end{align*}
\]
compute Bezout factors \(x\) and \(y\) of minimal degree such that \(a\times x + b\times y = 1\).

2.7.36  systmat  system matrix

CALLING SEQUENCE :

\[
[Sm]=\text{systmat}(Sl);
\]

PARAMETERS :

\[
Sl : \text{linear system (\text{syslin} list) or descriptor system}
\]
\[
Sm : \text{matrix pencil}
\]

DESCRIPTION :

System matrix of the linear system \(Sl\) (\text{syslin} list) in state-space form (utility function).
\[
Sm = [-sI + A \quad B; \\
[ \quad C \quad D]
\]
For a descriptor system \((Sl=\text{list}('\text{des}',A,B,C,D,E))\), \text{systmat} returns:
\[
Sm = [-sE + A \quad B; \\
[ \quad C \quad D]
\]

SEE ALSO :  \text{ss2des 361, sm2des 360, sm2ss 360}
2.8 Linear Algebra
2.8.1 \texttt{aff2ab} \hspace{1cm} \textbf{linear (affine) function to A,b conversion}

\textbf{CALLING SEQUENCE :}

\[ [A,b]=\texttt{aff2ab}(\texttt{afuction}, \texttt{dimX}, \texttt{D} [,\texttt{flag}]) \]

\textbf{PARAMETERS :}

\texttt{afuction} : a scilab function \( Y=fct(X,D) \) where \( X, \ D, \ Y \) are list of matrices

\texttt{dimX} : a \( p \times 2 \) integer matrix (\( p \) is the number of matrices in \( X \))

\( D : \) a list of real matrices (or any other valid Scilab object).

\texttt{flag} : optional parameter (\texttt{flag='f'} or \texttt{flag='sp'})

\( A : \) a real matrix

\( b : \) a real vector having same row dimension as \( A \)

\textbf{DESCRIPTION :}

\texttt{aff2ab} returns the matrix representation of an affine function (in the canonical basis).

\texttt{afuction} is a function with imposed syntax: \( Y=\texttt{afuction}(X,D) \) where \( X=\text{list}(X_1,X_2,...,X_p) \)

\( D=\text{list} \) of real matrices, and \( Y=\text{list}(Y_1,...,Y_q) \) is a list of \( q \) real real matrices which depend linearly of the \( X_i \)’s. The (optional) input \( D \) contains parameters needed to compute \( Y \) as a function of \( X \). (It is generally a list of matrices).

\( \texttt{dimX} \) is a \( p \times 2 \) matrix: \( \texttt{dimX}(i)=[nri,nci] \) is the actual number of rows and columns of matrix \( X_i \).

These dimensions determine \( na \), the column dimension of the resulting matrix \( A: na=nri*nci +...+nrp*npc \).

If the optional parameter \texttt{flag='sp'} the resulting \( A \) matrix is returned as a sparse matrix.

This function is useful to solve a system of linear equations where the unknown variables are matrices.

\textbf{EXAMPLE :}

// Lyapunov equation solver (one unknown variable, one constraint)
\texttt{deff(‘Y=lyapunov(X,D)’,’[A,Q]=D(:);Xm=X(:); Y=\text{list}(A''*Xm+Xm*A-Q)’)}
\( A=\text{rand}(3,3);Q=\text{rand}(3,3);Q=Q+Q’;D=\text{list}(A,Q);\texttt{dimX}=[3,3];\)
\texttt{[Aly,bly]=aff2ab(lyapunov,\texttt{dimX},D);}
\texttt{[Xl,kerA]=linsolve(Aly,bly); Xv=vec2list(Xl,\texttt{dimX}); lyapunov(Xv,D)}
\texttt{Xm=Xv(:); A'*Xm+Xm*A-Q}

// Lyapunov equation solver with redundant constraint \( X=X' \)
// (one variable, two constraints) \( D \) is global variable
\texttt{deff(‘Y=ly2(X,D)’,’[A,Q]=D(:);Xm=X(:); Y=\text{list}(A''*Xm+Xm*A-Q,Xm’-Xm’)’)}
\( A=\text{rand}(3,3);Q=\text{rand}(3,3);Q=Q+Q’;D=\text{list}(A,Q);\texttt{dimX}=[3,3];\)
\texttt{[Aly,bly]=aff2ab(ly2,\texttt{dimX},D);}
\texttt{[Xl,kerA]=linsolve(Aly,bly); Xv=vec2list(Xl,\texttt{dimX}); ly2(Xv,D)}

// Francis equations
// Find matrices \( X_1 \) and \( X_2 \) such that:
// \( A_1*X_1 - X_1*A_2 + B*X_2 -A_3 = 0 \)
// \( D_1*X_1 -D_2 = 0 \)
\texttt{deff(‘Y=bruce(X,D)’,’[A1,A2,A3,B,D1,D2]=D(:);...\}
\texttt{[X1,X2]=X(:);Y=\text{list}(A1*X1-X1*A2+B*X2-A3,D1*X1-D2)’)}
\texttt{A1=[-4,10;-1,2];A3=[1;2];B=[0;1];A2=1;D1=[0,1];D2=1;\}
\texttt{D=\text{list}(A1,A2,A3,B,D1,D2);\}
\texttt{[n1,m1]=\text{size}(A1);[n2,m2]=\text{size}(A2);[n3,m3]=\text{size}(B);\}
\texttt{\texttt{dimX}=[[m1,n2]};[m3,m2]);\}
\texttt{[Af,bf]=aff2ab(bruce,\texttt{dimX},D);\}
\texttt{[Xf,KerAf]=linsolve(Af,bf);Xsol=vec2list(Xf,\texttt{dimX})\}
\texttt{bruce(Xsol,D)}

\textbf{Scilab Group} \hspace{6cm} April 1993 \hspace{2cm} 501
// Find all X which commute with A
deff(’y=f(X,D)’,’y=list(D(:)*X(:)-X(:)*D(:))’)
A=rand(3,3);dimX=[3,3];[Af,bf]=aff2ab(f,dimX,list(A));
[Xf,KerAf]=linsolve(Af,bf);[p,q]=size(KerAf);
Xsol=vec2list(Xf+KerAf*rand(q,1),dimX);
C=Xsol(:); A*C-C*A

SEE ALSO:  linsolve 518

2.8.2  **balanc**  

**CALLING SEQUENCE:**

[A, X] = balanc(A)

**PARAMETERS:**

A:  a real square matrix
X:  a real square invertible matrix

**DESCRIPTION:**

Balance a square matrix to improve its condition number.

[A, X] = balanc(A) finds a similarity transformation X such that \( Ab = inv(X) \ast A \ast X \) has approximately equal row and column norms.

For matrix pencils, balancing is done for improving the generalized eigenvalue problem.

[Ep, Ap, X, Y] = balanc(E, A) returns left and right transformations X and Y such that \( Ep = X \ast E \ast Y \), \( Ap = X \ast A \ast Y \)

**REMARK:**

Balancing is made in the functions bdiag and spec.

**EXAMPLE:**

```plaintext
A=[1/2ˆ10,1/2ˆ10;2ˆ10,2ˆ10];
[Ab, X] = balanc(A);
norm(A(1,:))/norm(A(2,:))
norm(Ab(1,:))/norm(Ab(2,:))
```

SEE ALSO:  bdiag 502

2.8.3  **bdiag**  

**CALLING SEQUENCE:**

[Ab [, X [, bs]]] = bdiag(A [, rmax])

**PARAMETERS:**

A : real or complex square matrix
rmax : real number
Ab : real or complex square matrix
X : real or complex non-singular matrix

Scilab Group  April 1993  502
bs : vector of integers

**DESCRIPTION :**

\[ [\text{Ab}, \text{X}, [,bs]] = \text{bdiag}(A [,rmax]) \]

performs the block-diagonalization of matrix \( A \). \( bs \) gives the structure of the blocks (respective sizes of the blocks). \( X \) is the change of basis i.e \( A = \text{inv}(X) * A * X \) is block diagonal.

\( rmax \) controls the conditioning of \( X \); the default value is the \( L_1 \) norm of \( A \).

To get a diagonal form (if it exists) choose a large value for \( rmax \) (\( rmax = 1/\text{eps} \) for example). Generically (for real random \( A \)) the blocks are (1x1) and (2x2) and \( X \) is the matrix of eigenvectors.

**EXAMPLE :**

//Real case: 1x1 and 2x2 blocks
\( a = \text{rand}(5,5); [\text{ab}, \text{x}, \text{bs}] = \text{bdiag}(a); \text{ab} \)

//Complex case: complex 1x1 blocks
\[ [\text{ab}, \text{x}, \text{bs}] = \text{bdiag}(a + \%i*0); \text{ab} \]

**SEE ALSO :** schur 533, sylv 539, spec 537

### 2.8.4 chfact

**spcho=chfact(A)**

**PARAMETERS :**

\( A \) : square symmetric positive sparse matrix

\( \text{spcho} \) : list containing the Cholesky factors in coded form

**DESCRIPTION :**

\( \text{spcho}=\text{chfact}(A) \) computes the sparse Cholesky factors of sparse matrix \( A \), assumed symmetric positive definite. This function is based on the Ng-Peyton programs (ORNL). See the Fortran programs for a complete description of the variables in \( \text{spcho} \). This function is to be used with \text{chsolve}.

**SEE ALSO :** chsolve 504, sparse 214, lufact 520, luget 520, spchol 536

### 2.8.5 chol

**[R]=chol(X)**

**PARAMETERS :**

\( X \) : a symmetric positive definite real or complex matrix.

**DESCRIPTION :**

If \( X \) is positive definite, then \( R = \text{chol}(X) \) produces an upper triangular matrix \( R \) such that \( R' * R = X \).

\text{chol}(X) \) uses only the diagonal and upper triangle of \( X \). The lower triangular is assumed to be the (complex conjugate) transpose of the upper.

**EXAMPLE :**

\( W = \text{rand}(5,5) + \%i*\text{rand}(5,5); \)
\( X = W' * W; \)
\( R = \text{chol}(X); \)
\( \text{norm}(R' * R - X) \)

**SEE ALSO :** spchol 536, qr 528, svd 538, bdiag 502, fullrf 510
2.8.6  

**chsolve**  

**CALLING SEQUENCE:**

```
sol = chsolve(spcho, rhs)
```

**PARAMETERS:**

- spcho : list containing the Cholesky factors in coded form returned by chfact
- rhs, sol : full column vectors

**DESCRIPTION:**

```
sol = chsolve(spcho, rhs)
```

computes the solution of

```
sol = A * rhs
```

with

```
A   a symmetric sparse positive definite matrix.
```

This function is based on the Ng-Peyton programs (ORNL). See the Fortran programs for a complete description of the variables in spcho.

**EXAMPLE:**

```
A = sprand(20, 20, 0.1);
A = A * A' + eye();
spcho = chfact(A);
sol = (1:20)'; rhs = A * sol;
spcho = chfact(A);
chsolve(spcho, rhs)
```

**SEE ALSO:** chfact 503, sparse 214, lufact 520, luget 520, spchol 536

2.8.7  

**classmarkov**  

**CALLING SEQUENCE:**

```
[perm, rec, tr, indsRec, indsT] = classmarkov(M)
```

**PARAMETERS:**

- M : real N x N Markov matrix. Sum of entries in each row should add to one.
- perm : integer permutation vector.
- rec, tr : integer vector, number (number of states in each recurrent classes, number of transient states).
- indsRec, indsT : integer vectors. (Indexes of recurrent and transient states).

**DESCRIPTION:**

Returns a permutation vector perm such that

```
M(perm, perm) = [M11 0 0 0 0 0]
    [0 M22 0 0 0 0]
    [0 0 M33 0 0]
    [ ...   ]
    [0 0 Msr 0 0]
    [* * * * Q]
```

Each Mi is a Markov matrix of dimension rec(i), i = 1, ..., r, Q is sub-Markov matrix of dimension tr. States 1 to sum(rec) are recurrent and states from r+1 to n are transient. One has perm = [indsRec, indsT] where indsRec is a vector of size sum(rec) and indsT is a vector of size tr.

**EXAMPLE:**

```
// P has two recurrent classes (with 2 and 1 states) 2 transient states
P = genmarkov([2, 1], 2, 'perm')
[perm, rec, tr, indsRec, indsT] = classmarkov(P);
P(perm, perm)
```

**SEE ALSO:** genmarkov 511, eigenmarkov 507

Scilab Group  

April 1993  

504
2.8.8 coff

Calling Sequence:

\[ [N,d]=\text{coff}(M [,\text{var}]) \]

Parameters:

- \( M \): square real matrix
- \( \text{var} \): character string
- \( N \): polynomial matrix (same size as \( M \))
- \( d \): polynomial (characteristic polynomial \( \text{poly}(A,'s') \))

Description:

\( \text{coff} \) computes \( R=(s*\text{eye}()-M)^{-1} \) for \( M \) a real matrix. \( R \) is given by \( N/d \).

\( N \) = numerator polynomial matrix.
\( d \) = common denominator.
\( \text{var} \) character string ('s' if omitted)

Example:

\[
M=[1,2;0,3];
[N,d]=\text{coff}(M)
N/d
\text{inv}(s*\text{eye}()-M)
\]

See Also: coffg 485, ss2tf 363, nlev 522, poly 65

2.8.9 colcomp

Column compression, kernel, nullspace

Calling Sequence:

\[ [W,rk]=\text{colcomp}(A [,\text{flag}] [,\text{tol}]) \]

Parameters:

- \( A \): real or complex matrix
- \( \text{flag} \): character string
- \( \text{tol} \): real number

\( W \): square non-singular matrix (change of basis)
\( rk \): integer (rank of \( A \))

Description:

Column compression of \( A: Ac=A*W \) is column compressed i.e
\( Ac=[0,Af] \) with \( Af \) full column rank, \( \text{rank}(Af)=\text{rank}(A)=rk \).

\( \text{flag} \) and \( \text{tol} \) are optional parameters: \( \text{flag} = 'qr' \) or \( 'svd' \) (default is \( 'svd' \)).
\( \text{tol} \) = tolerance parameter (of order \%eps as default value).

The \( ma-rk \) first columns of \( W \) span the kernel of \( A \) when \( \text{size}(A)=(na,ma) \)

Example:

\[
A=\text{rand}(5,2)*\text{rand}(2,5);
[X,r]=\text{colcomp}(A);
\text{norm}(A*X(:,1:r-1),1)
\]

See Also: rowcomp 531, fullrf 510, fullrfk 510, kernel 516
2.8.10 companion

CALLING SEQUENCE :

A=companion(p)

PARAMETERS :

p : polynomial or vector of polynomials
A : square matrix

DESCRIPTION :

Returns a matrix A with characteristic polynomial equal to p if p is monic. If p is not monic the characteristic polynomial of A is equal to p/c where c is the coefficient of largest degree in p.

If p is a vector of monic polynomials, A is block diagonal, and the characteristic polynomial of the ith block is p(i).

EXAMPLE :

s=poly(0,'s');
p=poly([1,2,3,4,1],'s','c')
det(s*eye()-companion(p))
roots(p)
spec(companion(p))

SEE ALSO: spec 537, poly 65, randpencil 529

AUTHOR: F.D.

2.8.11 cond

CALLING SEQUENCE :

cond(X)

PARAMETERS :

X : real or complex square matrix

DESCRIPTION :

Condition number in 2-norm. \( \text{cond}(X) \) is the ratio of the largest singular value of X to the smallest.

EXAMPLE :

A=testmatrix('hilb',6);
cond(A)

SEE ALSO: rcond 531, svd 538
### 2.8.12 det

**Calling Sequence:**

\[ \text{det}(X) \]

\[ [e, m] = \text{det}(X) \]

**Parameters:**

- \( X \): real or complex square matrix, polynomial or rational matrix.
- \( m \): real or complex number, the determinant base 10 mantissae.
- \( e \): integer, the determinant base 10 exponent.

**Description:**

\( \text{det}(X) \cdot 10^e \) is the determinant of the square matrix \( X \).

For polynomial matrix \( \text{det}(X) \) is equivalent to \( \text{determ}(X) \).

For rational matrices \( \text{det}(X) \) is equivalent to \( \text{detr}(X) \).

**Example:**

```scilab
x = poly(0, 'x');
det([x, 1 + x; 2 - x, x^2]);
w = ssrand(2, 2, 4); roots(det(systmat(w))), trzeros(w) // zeros of linear system
A = rand(3, 3);
det(A), prod(spec(A))
```

**See Also:** detr 487, determ 487

### 2.8.13 eigenmarkov

**Calling Sequence:**

\[ [M, Q] = \text{eigenmarkov}(P) \]

**Parameters:**

- \( P \): real \( N \times N \) Markov matrix. Sum of entries in each row should add to one.
- \( M \): real matrix with \( N \) columns.
- \( Q \): real matrix with \( N \) rows.

**Description:**

Returns normalized left and right eigenvectors associated with the eigenvalue 1 of the Markov transition matrix \( P \). If the multiplicity of this eigenvalue is \( m \) and \( P \) is \( N \times N \), \( M \) is \( m \times N \) matrix and \( Q \) a \( N \times m \) matrix. \( M(k,:) \) is the probability distribution vector associated with the \( k \)-th ergodic set (recurrent class). \( M(k,x) \) is zero if \( x \) is not in the \( k \)-th recurrent class. \( Q(x,k) \) is the probability to end in the \( k \)-th recurrent class starting from \( x \). If \( P^k \) converges for large \( k \) (no eigenvalues on the unit circle except 1), then the limit is \( Q^*M \) (eigenprojection).

**Example:**

```scilab
// P has two recurrent classes (with 2 and 1 states) 2 transient states
P = genmarkov([2, 1], 2);
[M, Q] = eigenmarkov(P);
P*Q - Q
Q*M - P^2
```

**See Also:** genmarkov 511, classmarkov 504
2.8.14  ereduc  computes matrix column echelon form by qz transformations

CALLING SEQUENCE :

[E, Q, Z [,stair [,rk]]] = ereduc(X, tol)

PARAMETERS :

X : m x n matrix with real entries.
tol : real positive scalar.
E : column echelon form matrix
Q : m x m unitary matrix
Z : n x n unitary matrix
stair : vector of indexes,
* ISTAIR(i) = + j if the boundary element E(i, j) is a corner point.
* ISTAIR(i) = - j if the boundary element E(i, j) is not a corner point.
(i=1,...,M)
rk : integer, estimated rank of the matrix

DESCRIPTION :

Given an m x n matrix X (not necessarily regular) the function ereduc computes a unitary transformed matrix E=Q*X*Z which is in column echelon form (trapezoidal form). Furthermore the rank of matrix X is determined.

EXAMPLE :

X=[1 2 3; 4 5 6]
[E, Q, Z, stair, rk] = ereduc(X, 1.d-15)

SEE ALSO:  fstair 509

AUTHOR: Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

2.8.15  exp  element-wise exponential

CALLING SEQUENCE :

exp(X)

PARAMETERS :

X : scalar, vector or matrix with real or complex entries.

DESCRIPTION :

exp(X) is the (element-wise) exponential of the entries of X.

EXAMPLE :

x=[1, 2, 3+%i];
log(exp(x)) //element-wise
2^x
exp(x*log(2))

SEE ALSO:  coff 505,  log 195,  expm 509
### 2.8.16 expm .............................. square matrix exponential

**CALLING SEQUENCE:**

`expm(X)`

**PARAMETERS:**

- `X` : square matrix with real or complex entries.

**DESCRIPTION:**

`X` is a square matrix and `expm(X)` is the matrix

\[
\exp(X) = I + X + X^2/2 + ...
\]

The computation is performed by first block-diagonalizing `X` and then applying a Pade approximation on each block.

**EXAMPLE:**

```plaintext
X=[1 2;3 4]
expm(X)
logm(expm(X))
```

**SEE ALSO:** `logm`, `bdiag`, `coff`, `log`, `exp`

### 2.8.17 fstair .............................. computes pencil column echelon form by qz transformations

**CALLING SEQUENCE:**

```plaintext
[AE,EE,QE,ZE,blcks,muk,nuk,muk0,nuk0,mnei]=fstair(A,E,Q,Z,stair,rk,tol)
```

**PARAMETERS:**

- `A` : m x n matrix with real entries.
- `tol` : real positive scalar.
- `E` : column echelon form matrix
- `Q` : m x m unitary matrix
- `Z` : n x n unitary matrix
- `stair` : vector of indexes (see ereduc)
- `rk` : integer, estimated rank of the matrix
- `AE` : m x n matrix with real entries.
- `EE` : column echelon form matrix
- `QE` : m x m unitary matrix
- `ZE` : n x n unitary matrix
- `nblcks` : the number of submatrices having full row rank >= 0 detected in matrix A.
- `muk` : integer array of dimension (n). Contains the column dimensions mu(k) (k=1,...,nblcks) of the submatrices having full column rank in the pencil sE(eps)-A(eps)
- `nuk` : integer array of dimension (m+1). Contains the row dimensions nu(k) (k=1,...,nblcks) of the submatrices having full row rank in the pencil sE(eps)-A(eps)
- `muk0` : integer array of dimension (n). Contains the column dimensions mu(k) (k=1,...,nblcks) of the submatrices having full column rank in the pencil sE(eps,inf)-A(eps,inf)
- `nuk0` : integer array of dimension (m+1). Contains the row dimensions nu(k) (k=1,...,nblcks) of the submatrices having full row rank in the pencil sE(eps,inf)-A(eps,inf)
**DESCRIPTION:**

Given a pencil $sE - A$ where matrix $E$ is in column echelon form the function `fstair` computes according to the wishes of the user a unitary transformed pencil $QE(sEE-AE)ZE$ which is more or less similar to the generalized Schur form of the pencil $sE - A$. The function yields also part of the Kronecker structure of the given pencil.

$Q, Z$ are the unitary matrices used to compute the pencil where $E$ is in column echelon form (see `ereduc`)

**AUTHOR:** Th.G.J. Beelen (Philips Glass Eindhoven). SLICOT

**SEE ALSO:** `quaskro` 528, `ereduc` 508

---

**2.8.18 fullrf .............................. full rank factorization**

**CALLING SEQUENCE:**

```plaintext
[Q, M, rk]=fullrf(A, [tol])
```

**PARAMETERS:**

- `A` : real or complex matrix
- `tol` : real number (threshold for rank determination)
- `Q, M` : real or complex matrix
- `rk` : integer (rank of `A`)

**DESCRIPTION:**

Full rank factorization: `fullrf` returns $Q$ and $M$ such that $A = Q*M$ with $\text{range}(Q) = \text{range}(A)$ and $\ker(M) = \ker(A), Q$ full column rank, $M$ full row rank, $rk = \text{rank}(A) = \#\text{columns}(Q) = \#\text{rows}(M)$.

`tol` is an optional real parameter (default value is $\sqrt{\%\text{eps}}$). The rank $rk$ of $A$ is defined as the number of singular values larger than $\|A\|*\text{tol}$.

If $A$ is symmetric, `fullrf` returns $M=Q'$.

**EXAMPLE:**

```plaintext
A=rand(5,2)*rand(2,5);
[Q, M]=fullrf(A);
norm(Q*M-A,1);
[X, d]=rowcomp(A); Y=X';
svd([A, Y(:,1:d), Q]); //span(Q) = span(A) = span(Y(:,1:2))
```

**SEE ALSO:** `svd` 538, `qr` 528, `fullrfk` 510, `rowcomp` 531, `colcomp` 505

**AUTHOR:** F.D.

---

**2.8.19 fullrfk ......................... full rank factorization of $A^k$**

**CALLING SEQUENCE:**

```plaintext
[Bk, Ck]=fullrfk(A, k)
```

**PARAMETERS:**

- `A` : real or complex matrix
- `k` : integer

**AUTHOR:** F.D.
**DESCRIPTION:**
This function computes the full rank factorization of $A^k$ i.e. $Bk \times Ck = A^k$ where $Bk$ is full column rank and $Ck$ full row rank. One has $\text{range}(Bk) = \text{range}(A^k)$ and $\text{ker}(Ck) = \text{ker}(A^k)$.
For $k=1$, `fullrfk` is equivalent to `fullrf`.

**EXAMPLE:**
```scilab
A=rand(5,2)*rand(2,5);[Bk,Ck]=fullrfk(A,3);
norm(Bk*Ck-A^3,1)
```

**SEE ALSO:** `fullrf` 510, `range` 530

**AUTHOR:** F.D (1990)

---

### 2.8.20 genmarkov

Generates random Markov matrix with recurrent and transient classes.

**CALLING SEQUENCE:**
```
M=genmarkov(rec,tr)
M=genmarkov(rec,tr,flag)
```

**PARAMETERS:**
- `rec`: integer row vector (its dimension is the number of recurrent classes).
- `tr`: integer (number of transient states)
- `M`: real Markov matrix. Sum of entries in each row should add to one.
- `flag`: string ‘perm’. If given, a random permutation of the states is done.

**DESCRIPTION:**
Returns in $M$ a random Markov transition probability matrix with $\text{size}(rec,1)$ recurrent classes with $rec(1),...rec($) entries respectively and $tr$ transient states.

**EXAMPLE:**
```scilab
//P has two recurrent classes (with 2 and 1 states) 2 transient states
P=genmarkov([2,1],2,’perm’)
[perm,rec,tr,indsRec,indsT]=classmarkov(P);
P(perm,perm)
```

**SEE ALSO:** `classmarkov` 504, `eigenmarkov` 507

---

### 2.8.21 givens

Givens transformation.

**CALLING SEQUENCE:**
```
U=givens(xy)
U=givens(x,y)
[U,c]=givens(xy)
[U,c]=givens(x,y)
```

**PARAMETERS:**
- $x, y$: two real or complex numbers
- $xy$: real or complex size 2 column vector
gschur Scilab Function

U : 2x2 unitary matrix
c : real or complex size 2 column vector

DESCRIPTION:
U = givens(x, y) or U = givens(xy) with xy = [x;y] returns a 2x2 unitary matrix U such that:
U*xy=[r;0]=c.
Note that givens(x,y) and givens([x;y]) are equivalent.

EXAMPLE:
A=[3,4;5,6];
U=givens(A(:,1));
U*A

SEE ALSO: qr 528

2.8.22 glever inverse of matrix pencil

CALLING SEQUENCE:
[Bfs,Bis,chis]=glever(E,A [,s])

PARAMETERS:
E, A : two real square matrices of same dimensions
s : character string (default value 's')
Bfs, Bis : two polynomial matrices
chis : polynomial

DESCRIPTION:
Computation of \((sE-A)^{-1}\) by generalized Leverrier's algorithm for a matrix pencil.
\((sE-A)^{-1} = (Bfs/chis) - Bis.\)

chis = characteristic polynomial (up to a multiplicative constant).
Bfs = numerator polynomial matrix.
Bis = polynomial matrix (- expansion of \((sE-A)^{-1}\) at infinity).
Note the - sign before Bis.

CAUTION:
This function uses cleanp to simplify Bfs, Bis and chis.

EXAMPLE:
s=%s;F=[-1,s,0,0;0,-1,0,0;0,0,s-2,0;0,0,0,s-1];
[Bfs,Bis,chis]=glever(F)
inv(F)-((Bfs/chis) - Bis)

AUTHOR: F. D. (1988)

SEE ALSO: rowshuff 532, det 507, invr 491, coffg 485, pencan 524, penlaur 524

Scilab Group April 1993 512
gschur ________________ generalized Schur form (matrix pencils).

**CALLING SEQUENCE:**

\[
\begin{align*}
[\text{As}, \text{Es}] &= \text{gschur}(\text{A, E}) \\
[\text{As}, \text{Es}, Q, Z] &= \text{gschur}(\text{A, E}) \\
[\text{As}, \text{Es}, Z, \text{dim}] &= \text{gschur}(\text{A, E, flag}) \\
[\text{As}, \text{Es}, Z, \text{dim}] &= \text{gschur}(\text{A, E, extern})
\end{align*}
\]

**PARAMETERS:**

- \(A, E\): two real square matrices
- \(\text{flag}\): character string ('c' or 'd')
- \(\text{extern}\): Scilab "external" function (usual case). Could be also a list or a character string
- \(\text{As, Es}\): two real square matrices
- \(Q, Z\): two non-singular real matrices
- \(\text{dim}\): integer (dimension of subspace)

**DESCRIPTION:**

Schur form of matrix pencils (QZ algorithm):

\[
[\text{As}, \text{Es}] = \text{gschur}(\text{A, E})
\]

produces a quasi triangular \(\text{As}\) matrix and a triangular \(\text{Es}\) matrix which are the generalized Schur form of the pair \(A, E\).

\[
[\text{As}, \text{Es}, Q, Z] = \text{gschur}(\text{A, E})
\]

returns in addition two unitary matrices \(Q\) and \(Z\) such that \(\text{As}=Q*\text{A}*Z\) and \(\text{Es}=Q*\text{E}*Z\).

Ordered stable form:

\[
[\text{As}, \text{Es}, Z, \text{dim}] = \text{gschur}(\text{A, E, 'c'})
\]

returns the real generalized Schur form of the pencil \(s*E-A\). In addition, the \(\text{dim}\) first columns of \(Z\) span a basis of the right eigenspace associated with eigenvalues with negative real parts (stable "continuous time" generalized eigenspace).

\[
[\text{As}, \text{Es}, Z, \text{dim}] = \text{gschur}(\text{A, E, 'd'})
\]

returns the real generalized Schur form of the pencil \(s*E-A\). In addition, the \(\text{dim}\) first columns of \(Z\) make a basis of the right eigenspace associated with eigenvalues with magnitude lower than 1 (stable "discrete time" generalized eigenspace).

General subspace:

\[
[\text{As}, \text{Es}, Z, \text{dim}] = \text{gschur}(\text{A, E, extern})
\]

returns the real generalized Schur form of the pencil \(s*E-A\). In addition, the \(\text{dim}\) first columns of \(Z\) make a basis of the right eigenspace associated with eigenvalues of the pencil which are selected according to a rule which is given by the scilab function \(\text{extern}\). (See \(\text{schur}\) for definition of this function).

**EXAMPLE:**

\[
s=%s; \\
F=[-1,s,0,0;0,-1,0,0;0,0,2+s,0;0,0,0,-2+s]; \\
\text{roots(det}(F)); \\
[E,A]=\text{pen2ea}(F); \\
[\text{As}, \text{Es}, Z, \text{dim}] = \text{gschur}(A, E, 'c') \\
// Other example \\
a=\text{rand}(4,4); b=\text{rand}(4,4); [\text{as}, \text{bs}, \text{qs}, \text{zs}] = \text{gschur}(a, b);
\]
norm(qs*a*zs-as)
norm(qs*b*zs-bs)
clear a;
a(8,8)=2; a(1,8)=1; a(2,[2,3,4,5])=[0.3,0.2,4,6]; a(3,[2,3])=[-0.2,.3];
a(3,7)=.5;
a(4,4)=.5; a(4,6)=2; a(5,5)=1; a(6,6)=4; a(6,7)=2.5; a(7,6)=-10; a(7,7)=4;
b=eye(8,8);b(5,5)=0;
[al,be]=gspec(a,b);
[bs,as,q,n]=gschur(b,a,'disc');n-4

SEE ALSO: external 38, gspec 514, pencan 524, penlaur 524, coffg 485, kroneck 517

2.8.24 gspec ___________________________________________ eigenvalues of matrix pencil

CALLING SEQUENCE :

[al,be]=gspec(A,E)
[al,be,Z]=gspec(A,E)

PARAMETERS :

A, E : real square matrices
al, be : real vectors
Z : real square non-singular matrix

DESCRIPTION :

[al,be] = gspec(A,E) returns the spectrum of the matrix pencil s E - A, i.e. the roots of the polynomial matrix s E - A. The eigenvalues are given by al./be and if be(i) = 0 the ith eigenvalue is at infinity. (For E = eye(A), al./be is spec(A)).

[al,be,Z] = gspec(A,E) returns in addition the matrix Z of generalized right eigenvectors of the pencil.

EXAMPLE :

A=rand(3,3);
[al,be,Z] = gspec(A,eye(A));al./be
clean(inv(Z)*A*Z) //displaying the eigenvalues (generic matrix)
A=A+%i*rand(A);E=rand(A);
roots(det(%s*E-A)) //complex case

SEE ALSO: gschur 513, balanc 502, spec 537, kroneck 517

2.8.25 hess ___________________________________________ Hessenberg form

CALLING SEQUENCE :

H = hess(A)
[U,H] = hess(A)

PARAMETERS :

A : real or complex square matrix
H : real or complex square matrix
U : orthogonal or unitary square matrix
DESCRIPTION:

\[ [U, H] = \text{hess}(A) \] produces a unitary matrix \( U \) and a Hessenberg matrix \( H \) so that \( A = U^H U' \) and \( U'^* U = \text{Identity} \). By itself, \text{hess}(A) returns \( H \).

The Hessenberg form of a matrix is zero below the first subdiagonal. If the matrix is symmetric or Hermitian, the form is tridiagonal.

EXAMPLE:

\[
A = \text{rand}(3,3); [U, H] = \text{hess}(A);
\]
\[
\text{and}( \text{abs}(U^H U' - A) < 1.d-10 )
\]

SEE ALSO: qr 528, contr 328, schur 533

2.8.26 householder ________________ Householder orthogonal reflexion matrix

CALLING SEQUENCE:

\[
u = \text{householder}(v [,w])
\]

PARAMETERS:

\( v \): real or complex column vector
\( w \): real or complex column vector with same size as \( v \). Default value is \( \text{eye}(v) \)
\( u \): real or complex column vector

DESCRIPTION:

given 2 column vectors \( v, w \) of same size, \( \text{householder}(v, w) \) returns a unitary column vector \( u \), such that \( (\text{eye}() - 2 * u * u'^*) * v \) is proportional to \( w \). \( (\text{eye}() - 2 * u * u'^*) \) is the orthogonal Householder reflexion matrix.

\( w \) default value is \( \text{eye}(v) \). In this case vector \( (\text{eye}() - 2 * u * u'^*) * v \) is the vector \( \text{eye}(v) * \text{norm}(v) \).

SEE ALSO: qr 528, givens 511

2.8.27 im_inv ____________________________ inverse image

CALLING SEQUENCE:

\[
[X, \text{dim}] = \text{im_inv}(A, B [,\text{tol}])
\]
\[
[X, \text{dim}, Y] = \text{im_inv}(A, B, [,\text{tol}])
\]

PARAMETERS:

\( A, B \): two real or complex matrices with equal number of columns
\( X \): orthogonal or unitary square matrix of order equal to the number of columns of \( A \)
\( \text{dim} \): integer (dimension of subspace)
\( Y \): orthogonal matrix of order equal to the number of rows of \( A \) and \( B \).

DESCRIPTION:

\[
[X, \text{dim}] = \text{im_inv}(A, B) \] computes \( A^{-1}(B) \), i.e vectors whose image through \( A \) are in range(\( B \)).
The first columns of \( X \) span \( A^{-1}(B) \).
\( \text{tol} \) is a threshold used to test if subspace inclusion; default value is \( \text{tol} = 100*\text{eps} \). If \( Y \) is returned, then \([Y*A*X, Y*B] \) is partitioned as follows:

\[
[A11, A12; 0, A22], [B1; 0]
\]
where $B_1$ has full row rank (equals rank($B$)) and $A_{22}$ has full column rank and has $\text{dim}$ columns.

**EXAMPLE:**

```plaintext
A=[rand(2,5);[zeros(3,4),rand(3,1)]]; B=[[1,1;1,1];zeros(3,2)];
W=rand(5,5); A=W*A; B=W*B;
[X,dim]=im_inv(A,B)
svd([A*X(:,1:dim),B]) //vectors A*X(:,1:dim) belong to range(B)
[X,dim,Y]=im_inv(A,B); [Y*A*X,Y*B]
```

**SEE ALSO:** rowcomp 531, spaninter 534, spanplus 535, linsolve 518

**Author:** F. D.

#### 2.8.28 inv

**matrix inverse**

**CALLING SEQUENCE:**

```plaintext
inv(X)
```

**PARAMETERS:**

$X$ : real or complex square matrix, polynomial matrix, rational matrix in transfer or state-space representation.

**DESCRIPTION:**

$\text{inv}(X)$ is the inverse of the square matrix $X$. A warning message is printed if $X$ is badly scaled or nearly singular.

For polynomial matrices or rational matrices in transfer representation, $\text{inv}(X)$ is equivalent to $\text{invr}(X)$.

For linear systems in state-space representation ($\text{syslin}$ list), $\text{invr}(X)$ is equivalent to $\text{invsysslin}(X)$.

**EXAMPLE:**

```plaintext
A=rand(3,3); inv(A)*A
//
x=poly(0,'x');
A=[x,1,x; x^2,2,1+x;1,2,3]; inv(A)*A
//
A=[1/x,2+2x,2/(1+x)]
inv(A)*A
//
A=ssrand(2,2,3);
W=inv(A)*A
clean(ss2tf(W))
```

**SEE ALSO:** slash 71, backslash 26, pinv 525, qr 528, lufact 520, lusolve 521, invr 491, coff 505, coffg 485

#### 2.8.29 kernel

**kernel, nullspace**

**CALLING SEQUENCE:**

```plaintext
W=kernel(A [,tol,[,flag])
```

**See Also:**
PARAMETERS:
A : full real or complex matrix or real sparse matrix
flag : character string ‘svd’ (default) or ‘qr’
tol : real number
W : full column rank matrix

DESCRIPTION:
W=kernel (A) returns the kernel (nullspace) of A.
flag and tol are optional parameters: flag = ‘qr’ or ‘svd’ (default is ‘svd’).
tol = tolerance parameter (of order %eps as default value).

EXAMPLE:
A=rand(3,1)*rand(1,3);
A*kernel(A)
A=sparse(A);
clean(A*kernel(A))

SEE ALSO: colcomp 505, fullrf 510, fullrfk 510, linsolve 518

2.8.30 kroneck ________________________________ Kronecker form of matrix pencil

CALLING SEQUENCE:

[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck (F)
[Q,Z,Qd,Zd,numbeps,numbeta]=kroneck (E,A)

PARAMETERS:
F : real matrix pencil F=s*E-A
E,A : two real matrices of same dimensions
Q,Z : two square orthogonal matrices
Qd,Zd : two vectors of integers
numbeps,numbeta : two vectors of integers

DESCRIPTION:
Kronecker form of matrix pencil: kroneck computes two orthogonal matrices Q, Z which put the
pencil F=s*E -A into upper-triangular form:

\[
\begin{bmatrix}
  sE(\text{eps})-A(\text{eps}) & X & X & X \\
  0 & sE(\text{inf})-A(\text{inf}) & X & X \\
  0 & 0 & sE(\text{f})-A(\text{f}) & X \\
  0 & 0 & 0 & sE(\text{eta})-A(\text{eta}) \\
\end{bmatrix}
\]

The dimensions of the four blocks are given by:
eps=Qd(1) x Zd(1), inf=Qd(2) x Zd(2), f = Qd(3) x Zd(3), eta=Qd(4)xZd(4)
The inf block contains the infinite modes of the pencil.
The f block contains the finite modes of the pencil.

AUTHOR: F.D.
The structure of epsilon and eta blocks are given by:

- \( \text{numbeps}(1) \) = \# of eps blocks of size 0 x 1
- \( \text{numbeps}(2) \) = \# of eps blocks of size 1 x 2
- \( \text{numbeps}(3) \) = \# of eps blocks of size 2 x 3 etc...
- \( \text{numbeta}(1) \) = \# of eta blocks of size 1 x 0
- \( \text{numbeta}(2) \) = \# of eta blocks of size 2 x 1
- \( \text{numbeta}(3) \) = \# of eta blocks of size 3 x 2 etc...

The code is taken from T. Beelen (Slicot-WGS group).

EXAMPLE:

```plaintext
F=randpencil([1,1,2],[2,3],[-1,3,1],[0,3]);
Q=rand(17,17);Z=rand(18,18);F=Q*F*Z;
//random pencil with eps1=1,eps2=1,eps3=1; 2 J-blocks @ infty
//with dimensions 2 and 3
//3 finite eigenvalues at -1,3,1 and eta1=0,eta2=3
[Qd,Zd,Qd,Zd,numbeps,numbeta]=kroneck(F);
[Qd(1),Zd(1)] //eps. part is sum(epsi) x (sum(epsi) + number of epsi)
[Qd(2),Zd(2)] //infinity part
[Qd(3),Zd(3)] //finite part
[Qd(4),Zd(4)] //eta part is (sum(etai) + number(eta1)) x sum(etai)
numbeps
numbeta
```

SEE ALSO: gschur 513, gspec 514, systmat 499, pencan 524, randpencil 529, trzeros 368

2.8.31 \text{linsolve} \quad \text{linear equation solver}

\text{CALLING SEQUENCE:}

\[ [x0, \text{kerA}]=\text{linsolve}(A,b [,x0]) \]

\text{PARAMETERS:}

- \( A \): an \( na \times ma \) real matrix (possibly sparse)
- \( b \): an \( na \times 1 \) vector (same row dimension as \( A \))
- \( x0 \): a real vector
- \( \text{kerA} \): an \( ma \times k \) real matrix

\text{DESCRIPTION:}

\text{linsolve} computes all the solutions to \( A*x+b=0 \).
\( x0 \) is a particular solution (if any) and \( \text{kerA} = \) nullspace of \( A \). Any \( x=x0+\text{kerA}*w \) with arbitrary \( w \) satisfies \( A*x+b=0 \).
If compatible \( x0 \) is given on entry, \( x0 \) is returned. If not a compatible \( x0 \), if any, is returned.

\text{EXAMPLE:}

```plaintext
A=rand(5,3)*rand(3,8);
b=A*ones(8,1);[x,kerA]=linsolve(A,b);A*x+b //compatible b
b=ones(5,1);[x,kerA]=linsolve(A,b);A*x+b //uncompatible b
A=rand(5,5);[x,kerA]=linsolve(A,b), -inv(A)*b //x is unique
```

SEE ALSO: inv 516, pinv 525, colcomp 505, im_inv 515

Scilab Group April 1993 518
**2.8.32  lu  ________________  LU factors of Gaussian elimination**

CALLING SEQUENCE :

\[ [L, U] = \text{lufact}(A) \]
\[ [L, U, E] = \text{lufact}(A) \]

PARAMETERS :

- \( A \) : real or complex square matrix (\( n \times n \)).
- \( L, U \) : two real or complex matrices (\( n \times n \)).
- \( E \) : a (\( n \times n \)) permutation matrix.

DESCRIPTION :

- \([L, U] = \text{lufact}(A)\) produces two matrices \( L \) and \( U \) such that \( A = L \cdot U \) with \( U \) upper triangular and \( E \cdot L \) lower triangular for a permutation matrix \( E \).
- If \( A \) has rank \( k \), rows \( k+1 \) to \( n \) of \( U \) are zero.

- \([L, U, E] = \text{lufact}(A)\) produces three matrices \( L, U \) and \( E \) such that \( E \cdot A = L \cdot U \) with \( U \) upper triangular and \( E \cdot L \) lower triangular for a permutation matrix \( E \).

REMARK :

If \( A \) is a real matrix, using the function \( \text{lufact} \) and \( \text{lusolve} \) it is possible to obtain the permutation matrices and also when \( A \) is not full rank the column compression of the matrix \( L \).

\[ [h, rk] = \text{lufact}(\text{sparse}(a)) \quad // \quad \text{lufact works with sparse real matrices} \]
\[ [P, L, U, Q] = \text{luget}(h) \]
\[ \text{ludel}(h) \]
\[ P = \text{full}(P); L = \text{full}(L); U = \text{full}(U); Q = \text{full}(Q); \]
\[ // \quad P, Q \quad \text{are permutation matrices} \quad P \cdot L \cdot U \cdot Q = A \]

SEE ALSO: \( \text{lufact} \) 520, \( \text{luget} \) 520, \( \text{lusolve} \) 521, \( \text{qr} \) 528, \( \text{svd} \) 538

**2.8.33  ludel  ________________  utility function used with lufact**

CALLING SEQUENCE :

\[ \text{ludel}(\text{hand}) \]

PARAMETERS :

- \( \text{hand} \) : handle to sparse lu factors (output of \( \text{lufact} \))

DESCRIPTION :

This function is used in conjunction with \( \text{lufact} \). It clears the internal memory space used to store the result of \( \text{lufact} \).

The sequence of commands \( [p, r] = \text{lufact}(A); x = \text{lusolve}(p, b); \text{ludel}(p); \) solves the sparse linear system \( A^*x = b \) and clears \( p \).

SEE ALSO: \( \text{sparse} \) 214, \( \text{lufact} \) 520, \( \text{luget} \) 520

Scilab Group  April 1993  519
### 2.8.34 lufact

#### Calling Sequence:

```latex
[hand, rk] = lufact(A, prec)
```

#### Parameters:

- **A**: square sparse matrix
- **hand**: handle to sparse lu factors
- **rk**: integer (rank of A)
- **prec**: a vector of size two `prec = [eps, reps]` giving the absolute and relative thresholds.

#### Description:

`[hand, rk] = lufact(A)` performs the lu factorization of sparse matrix A. `hand` (no display) is used by `lusolve` (for solving linear system) and `luget` (for retrieving the factors). `hand` should be cleared by the command: `ludel(hand);`

The A matrix needs not be full rank but must be square (since A is assumed sparse one may add zeros if necessary to squaring down A).

- **eps**: The absolute magnitude an element must have to be considered as a pivot candidate, except as a last resort. This number should be set significantly smaller than the smallest diagonal element that is expected to be placed in the matrix. The default value is `%eps`.
- **reps**: This number determines what the pivot relative threshold will be. It should be between zero and one. If it is one then the pivoting method becomes complete pivoting, which is very slow and tends to fill up the matrix. If it is set close to zero the pivoting method becomes strict Markowitz with no threshold. The pivot threshold is used to eliminate pivot candidates that would cause excessive element growth if they were used. Element growth is the cause of roundoff error. Element growth occurs even in well-conditioned matrices. Setting the reps large will reduce element growth and roundoff error, but setting it too large will cause execution time to be excessive and will result in a large number of fill-ins. If this occurs, accuracy can actually be degraded because of the large number of operations required on the matrix due to the large number of fill-ins. A good value seems to be 0.001 which is the default value. The default is chosen by giving a value larger than one or less than or equal to zero. This value should be increased and the matrix resolved if growth is found to be excessive. Changing the pivot threshold does not improve performance on matrices where growth is low, as is often the case with ill-conditioned matrices. reps was chosen for use with nearly diagonally dominant matrices such as node- and modified-node admittance matrices. For these matrices it is usually best to use diagonal pivoting. For matrices without a strong diagonal, it is usually best to use a larger threshold, such as 0.01 or 0.1.

#### Example:

```latex
a = rand(5, 5); b = rand(5, 1); A = sparse(a);
[h, rk] = lufact(A);
x = lusolve(h, b); a*x - b
ludel(h)
```

#### See Also:

- `sparse 214`, `lusolve 521`, `luget 520`

---

### 2.8.35 luget

#### Calling Sequence:

```latex
[P, L, U, Q] = luget(ptr)
```

---

Scilab Group April 1993 520
**Lyap Scilab Function**

**PARAMETERS:**
- `ptr` : pointer, output of `lufact`
- `P` : sparse permutation matrix
- `L` : sparse matrix, lower triangular if `ptr` is obtained from a non singular matrix
- `U` : square non singular upper triangular sparse matrix with ones along the main diagonal
- `Q` : sparse permutation matrix

**DESCRIPTION:**

```plaintext
[P,L,U,Q]=luget(ptr) with ptr obtained by the command [ptr,rk]=lufact(A) with A a sparse matrix returns four sparse matrices such that P*L*U*Q=A.
```

The A matrix needs not be full rank but must be square (since A is assumed sparse one may add zeros if necessary to squaring down A).

If A is singular, the L matrix is column compressed (with rk independent nonzero columns): the nonsingular sparse matrix \( Q' \cdot \text{inv}(U) \) column compresses A.

**EXAMPLE:**

```plaintext
a=rand(5,2)*rand(2,5); A=sparse(a);
[ptr,rk]=lufact(A);
[P,L,U,Q]=luget(ptr);
full(L), P*L*U*Q-A
clean(P*L*U*Q-A)
ludel(ptr)
```

**SEE ALSO:** sparse 214, lusolve 521, luget 520, clean 484

---

**2.8.36 lusolve ................................. sparse linear system solver**

**CALLING SEQUENCE:**

```plaintext
lusolve(hand,b)
lusolve(A,b)
```

**PARAMETERS:**
- `b` : full real matrix
- `A` : real square sparse invertible matrix
- `hand` : handle to a previously computed sparse lu factors (output of lufact)

**DESCRIPTION:**

```plaintext
x=lusolve(hand,b) solves the sparse linear system \( A \cdot x = b \).
```

`[hand,rk]=lufact(A)` is the output of lufact.

```plaintext
x=lusolve(A,b) solves the sparse linear system \( \backslash A \cdot x = b \).
```

**EXAMPLE:**

```plaintext
non_zeros=[1,2,3,4]; rows_cols=[1,1;2,2;3,3;4,4];
sp=sparse(rows_cols,non_zeros);
[h,rk]=lufact(sp); x=lusolve(h,[1;1;1;1]);ludel(h)
```

```plaintext
rk,sp*x
```

```plaintext
non_zeros=[1,2,3,4]; rows_cols=[1,1;2,2;3,3;4,4];
sp=sparse(rows_cols,non_zeros);
x=lusolve(sp,-ones(4,1));
```

```plaintext
sp*x
```

**SEE ALSO:** sparse 214, lufact 520, slash 71, backslash 26
2.8.37 lyap Lyapunov equation

CALLING SEQUENCE:

[X]=lyap(A,C,'c')
[X]=lyap(A,C,'d')

PARAMETERS:
A, C : real square matrices, C must be symmetric

DESCRIPTION:
X = lyap(A,C,flag) solves the continuous time or discrete time matrix Lyapunov matrix equation:
A'*X + X*A = C (flag='c')
A'*X*A - X = C (flag='d')
Note that a unique solution exist if and only if an eigenvalue of A is not an eigenvalue of -A (flag='c') or 1 over an eigenvalue of A (flag='d').

EXAMPLE:
A=rand(4,4);C=rand(A);C=C+C';
X=lyap(A,C,'c');
A'*X + X*A - C
X=lyap(A,C,'d');
A'*X*A - X - C

SEE ALSO: sylv 539, ctr_gram 330, obs_gram 349

2.8.38 nlev Leverrier’s algorithm

CALLING SEQUENCE:

[num,den]=nlev(A,z [,rmax])

PARAMETERS:
A : real square matrix
z : character string
rmax : optional parameter (see bdiag)

DESCRIPTION:
[num,den]=nlev(A,z [,rmax]) computes: (zI - A)^{-1} by block diagonalization of A followed by Leverrier’s algorithm on each block.

REMARK:
This algorithm is better than the usual leverrier algorithm but still not perfect!

EXAMPLE:
A=rand(3,3);x=poly(0,'x');
[NUM,den]=nlev(A,'x')
clean(den-poly(A,'x'))
clean(NUM/den-inv(x*eye()-A))

SEE ALSO: coff 505, coffg 485, glever 512, ss2tf 363

AUTHOR: F. D., S. S.
2.8.39 orth ................................................... orthogonal basis

CALLING SEQUENCE :

Q=orth(A)

PARAMETERS :
A : real or complex matrix
Q : real or complex matrix

DESCRIPTION :
Q=orth(A) returns Q, an orthogonal basis for the span of A. Range(Q) = Range(A) and Q’ *Q=eye. The number of columns of Q is the rank of A as determined by the QR algorithm.

EXAMPLE :
A=rand(5,3)*rand(3,4);
[X,dim]=rowcomp(A);X=X';
svd([orth(A),X(:,1:dim)])

SEE ALSO : qr 528, rowcomp 531, colcomp 505, range 530

2.8.40 pbig ................................................... eigen-projection

CALLING SEQUENCE :

[Q,M]=pbig(A,thres,flag)

PARAMETERS :
A : real square matrix
thres : real number
flag : character string ('c' or 'd')
Q, M : real matrices

DESCRIPTION :
Projection on eigen-subspace associated with eigenvalues with real part >= thres (flag='c') or with magnitude >= thres (flag='d'). The projection is defined by Q*M. Q is full column rank, M is full row rank and M*Q=eye.

If flag='c', the eigenvalues of M*A*Q = eigenvalues of A with real part >= thres.
If flag='d', the eigenvalues of M*A*Q = eigenvalues of A with magnitude >= thres.

If flag='c' and if [Q1,M1] = full rank factorization (fullrf) of eye()-Q*M then eigenvalues of M1*A*Q1 = eigenvalues of A with real part < thres.
If flag='d' and if [Q1,M1] = full rank factorization (fullrf) of eye()-Q*M then eigenvalues of M1*A*Q1 = eigenvalues of A with magnitude < thres.

EXAMPLE :
A=diag([1,2,3]);X=rand(A);A=inv(X)*A*X;
[Q,M]=pbig(A,1.5,'d');
spec(M*A*Q)
[Q1,M1]=fullrf(eye()-Q*M);
spec(M1*A*Q1)

SEE ALSO : psmall 527, projspec 526, fullrf 510

AUTHOR : F. D. (1988)
2.8.41 pencan ............................. canonical form of matrix pencil

CALLING SEQUENCE :

\[ [Q,M,i1] = \text{pencan}(Fs) \]
\[ [Q,M,i1] = \text{pencan}(E,A) \]

PARAMETERS :

Fs : a regular pencil \( sE - A \)
E, A : two real square matrices
Q, M : two non-singular real matrices
i1 : integer

DESCRIPTION :

Given the regular pencil \( Fs = sE - A \), pencan returns matrices Q and M such that \( M(sE-A)*Q \) is in "canonical" form.

\( M*E*Q \) is a block matrix

\[ [I,0; \]
\[ 0,N] \]

with \( N \) nilpotent and \( i1 = \) size of the I matrix above.

\( M*A*Q \) is a block matrix:

\[ [Ar,0; \]
\[ 0,I] \]

EXAMPLE :

\[
F = \text{randpencil}([], [1,2], [1,2,3], []); \\
F = \text{rand}(6,6)*F*\text{rand}(6,6); \\
[Q,M,i1] = \text{pencan}(F); \\
W = \text{clean}(M*F*Q) \\
\text{roots} (\det (W(1:i1,1:i1))) \\
\det (W(\$-2:\$,\$-2:\$))
\]

SEE ALSO: glever 512, penlaur 524, rowshuff 532

AUTHOR : F. D.

2.8.42 penlaur ............................ Laurent coefficients of matrix pencil

CALLING SEQUENCE :

\[ [Si, Pi, Di, order] = \text{penlaur}(Fs) \]
\[ [Si, Pi, Di, order] = \text{penlaur}(E,A) \]

PARAMETERS :

Fs : a regular pencil \( sE - A \)
E, A : two real square matrices
Si, Pi, Di : three real square matrices
order : integer
**polar**

**Scilab Function**

**DESCRIPTION :**
penlaur computes the first Laurent coefficients of \((sE-A)^{-1}\) at infinity. 
\((sE-A)^{-1} = ... + Si/s - Pi - sDi + ...\) at \(s = \infty\). 
order = order of the singularity (order=index-1). 
The matrix pencil \(Fs=sE-A\) should be invertible. 
For a index-zero pencil, \(Pi, Di,...\) are zero and \(Si=inv(E)\). 
For a index-one pencil (order=0), \(Di =0\). 
For higher-index pencils, the terms \(-s^2 Di(2), -s^3 Di(3),...\) are given by: 
\(Di(2)=Di*A*Di, Di(3)=Di*A*Di*A*Di \) (up to \(Di(order)\)).

**REMARK :**
Experimental version: troubles when bad conditioning of \(so*E-A\)

**EXAMPLE :**
\[
F=randpencil([],[],[1,2],[1,2,3],[]);
F=rand(6,6)*F*rand(6,6);[E,A]=pen2ea(F);
[Si,Pi,Di]=penlaur(F);
[Bfs,Bis,chis]=glever(F);
norm(coeff(Bis,1)-Di,1)
\]

**SEE ALSO :** glever 512, pencan 524, rowshuff 532

**AUTHOR :** F. D. (1988,1990)

**2.8.43**  
**pinv**  

**pseudoinverse**

**CALLING SEQUENCE :**

pinv(A,[tol])

**PARAMETERS :**

A : real or complex matrix

tol : real number

**DESCRIPTION :**
\(X = \text{pinv}(A)\) produces a matrix \(X\) of the same dimensions as \(A'\) such that:
\(A*X*A = A, X*A*X = X\) and both \(A*X\) and \(X*A\) are Hermitian.

The computation is based on SVD and any singular values lower than a tolerance are treated as zero: this tolerance is accessed by \(X=\text{pinv}(A,tol)\).

**EXAMPLE :**

\[
A=rand(5,2)*rand(2,4);
norm(A*pinv(A)*A-A,1)
\]

**SEE ALSO :**  rank 530,  svd 538,  qr 528

**2.8.44**  
**polar**

**polar form**

**CALLING SEQUENCE :**

[Ro,Theta]=polar(A)

**PARAMETERS :**

A : real or complex square matrix

Scilab Group  
April 1993  
525
**projspec** Scilab Function

**Ro, Theta : real matrices**

**DESCRIPTION :**

\[
\{Ro,\Theta\}=\text{polar}(A)\] returns the polar form of \(A\) i.e.:
\[
A=Ro*expm(%i*\Theta)\] 
\(Ro\) symmetric \(\neq 0\) and \(\Theta\) hermitian \(\neq 0\).

**EXAMPLE :**

\[
A=\text{rand}(5,5);
\{Ro,\Theta\}=\text{polar}(A);
norm(A-Ro*\text{expm}(%i*\Theta),1)
\]

**SEE ALSO :** expm 509, svd 538

**AUTHOR :** F. D.

---

**2.8.45 proj** projection

**CALLING SEQUENCE :**

\[
P = \text{proj}(X1,X2)
\]

**PARAMETERS :**

\(X1,X2\) : two real matrices with equal number of columns 
\(P\) : real projection matrix \((P^2=P)\)

**DESCRIPTION :**

\(P\) is the projection on \(X2\) parallel to \(X1\).

**EXAMPLE :**

\[
X1=\text{rand}(5,2);X2=\text{rand}(5,3);
P=\text{proj}(X1,X2);
norm(P^2-P,1)
\]

**SEE ALSO :** projspec 526, orth 523, fullrf 510

**AUTHOR :** F. D.

---

**2.8.46 projspec** spectral operators

**CALLING SEQUENCE :**

\[
[S,P,D,i]=\text{projspec}(A)
\]

**PARAMETERS :**

\(A\) : square matrix
\(S, P, D\) : square matrices
\(i\) : integer (index of the zero eigenvalue of \(A\)).
**DESCRIPTION:**
Spectral characteristics of $A$ at 0.
$S$ = reduced resolvent at 0 ($S = (\text{Drazin}^{-1}(A)$).
$P$ = spectral projection at 0.
$D$ = nilpotent operator at 0.
index = index of the 0 eigenvalue.
One has $(s\cdot\text{eye}()-A)^{-1} = D^{(i-1)}/s^i + \ldots + D/s^2 + P/s - S - s\cdot S^2 - \ldots$ around the singularity $s=0$.

**EXAMPLE:**

deff('j=jdrn(n)', 'j=zeros(n,n);for k=1:n-1; j(k,k+1)=1; end')
A=sysdiag(jdrn(3), jdrn(2), rand(2,2)); X=rand(7,7);
A=X*A*inv(X);
[S,P,D,index]=projspec(A);
index //size of J-block
trace(P) //sum of dimensions of J-blocks
A*S-(eye()-P)
norm(D^index,1)

**SEE ALSO:** coff 505

**AUTHOR:** F. D.

---

**2.8.47 psmall ................................. spectral projection**

**CALLING SEQUENCE:**

$[Q,M]=\text{psmall}(A,\text{thres},\text{flag})$

**PARAMETERS:**

A : real square matrix
thres : real number
flag : character string ('c' or 'd')
Q, M : real matrices

**DESCRIPTION:**
Projection on eigen-subspace associated with eigenvalues with real part $< \text{thres}$ (flag='c') or with modulus $< \text{thres}$ (flag='d').
The projection is defined by $Q*M$. $Q$ is full column rank, $M$ is full row rank and $M*Q=\text{eye}$.
If flag='c', the eigenvalues of $M*A*Q$ = eigenvalues of $A$ with real part $< \text{thres}$.
If flag='d', the eigenvalues of $M*A*Q$ = eigenvalues of $A$ with magnitude $< \text{thres}$.
If flag='c' and if $[Q1,M1]=\text{full rank factorization}(\text{fullrf})$ of $\text{eye()-Q*M}$ then eigenvalues of $M1*A*Q1$ = eigenvalues of $A$ with real part $>= \text{thres}$.
If flag='d' and if $[Q1,M1]=\text{full rank factorization}(\text{fullrf})$ of $\text{eye()-Q*M}$ then eigenvalues of $M1*A*Q1$ = eigenvalues of $A$ with magnitude $>= \text{thres}$.

**EXAMPLE:**

$A=\text{diag}([1,2,3]); X=\text{rand}(A); A=\text{inv}(X)*A*X;$
$[Q,M]=\text{psmall}(A,2.5, 'd');$
$\text{spec}(M*A*Q)$
$[Q1,M1]=\text{fullrf}(\text{eye()-Q*M});$
$\text{spec}(M1*A*Q1)$

**SEE ALSO:** pbig 523, proj 526, projspec 526

**AUTHOR:** F. D. (1988)
2.8.48 qr _______________________________________________________________________ QR decomposition

CALLING SEQUENCE :

\[ [Q,R]=\text{qr}(X) \]
\[ [Q,R,E]=\text{qr}(X) \]
\[ [Q,R,rk,E]=\text{qr}(X [,\text{tol}]) \]

PARAMETERS :

\( X \) : real or complex matrix
\( \text{tol} \) : nonnegative real number
\( Q \) : square orthogonal or unitary matrix
\( R \) : matrix with same dimensions as \( X \)
\( E \) : permutation matrix
\( \text{rk} \) : integer (QR-rank of \( X*E \))

DESCRIPTION :

\[ [Q,R] = \text{qr}(X) \] produces an upper triangular matrix \( R \) of the same dimension as \( X \) and a unitary matrix \( Q \) so that \( X = Q*R \).

\[ [Q,R,E] = \text{qr}(X) \] produces a (column) permutation matrix \( E \), an upper triangular \( R \) with decreasing diagonal elements and a unitary \( Q \) so that \( X*E = Q*R \).

\[ [Q,R,rk,E] = \text{qr}(X [,\text{tol}]) \] returns \( \text{rk} = \text{rank estimate of } X \) i.e. \( \text{rk} \) is the number of diagonal elements in \( R \) which are larger than \( \text{tol} \).

\[ [Q,R,rk,E] = \text{qr}(X) \] returns \( \text{rk} = \text{rank estimate of } X \) i.e. \( \text{rk} \) is the number of diagonal elements in \( R \) which are larger than \( R(1,1)*\text{eps} \times \text{max(size(R))} \).

EXAMPLE :

\[ A=\text{rand}(5,2)*\text{rand}(2,5); \]
\[ [Q,R,rk,E]=\text{qr}(A,1.d-10); \]
\[ \text{norm}(Q'*A-R) \]
\[ \text{svd}([A,Q(:,1:rk)]) \] //\( \text{span(A)} = \text{span(Q(:,1:rk))} \)

SEE ALSO: rank 530, svd 538, rowcomp 531, colcomp 505

2.8.49 quaskro _____________________________________________________________________ quasi-Kronecker form

CALLING SEQUENCE :

\[ [Q,Z,Qd,Zd,\text{numbeps},\text{numbeta}]=\text{quaskro}(F) \]
\[ [Q,Z,Qd,Zd,\text{numbeps},\text{numbeta}]=\text{quaskro}(E,A) \]
\[ [Q,Z,Qd,Zd,\text{numbeps},\text{numbeta}]=\text{quaskro}(F,\text{tol}) \]
\[ [Q,Z,Qd,Zd,\text{numbeps},\text{numbeta}]=\text{quaskro}(E,A,\text{tol}) \]

PARAMETERS :

\( F \) : real matrix pencil \( F=sE-A \) (\( s=\text{poly}(0,\text{`}s\text{'}) \))
\( E,A \) : two real matrices of same dimensions
\( \text{tol} \) : a real number (tolerance, default value=1.d-10)
\( Q,Z \) : two square orthogonal matrices
\( Qd,Zd \) : two vectors of integers
\( \text{numbeps} \) : vector of integers
**DESCRIPTION:**

Quasi-Kronecker form of matrix pencil: `quaskro` computes two orthogonal matrices \( Q, Z \) which put the pencil \( F=sE - A \) into upper-triangular form:

\[
Q(sE-A)Z = \begin{bmatrix}
sE(\text{eps})-A(\text{eps}) & X & X \\
0 & sE(\text{inf})-A(\text{inf}) & X \\
\vdots & \vdots & \vdots \\
0 & sE(r)-A(r) & X
\end{bmatrix}
\]

The dimensions of the blocks are given by:

- \( \text{eps} = Qd(1) \times Zd(1) \)
- \( \text{inf} = Qd(2) \times Zd(2) \)
- \( r = Qd(3) \times Zd(3) \)

The \( \text{inf} \) block contains the infinite modes of the pencil.

The \( f \) block contains the finite modes of the pencil.

The structure of epsilon blocks are given by:

- \( \text{numbebeps}(1) = \) # of eps blocks of size 0 x 1
- \( \text{numbebeps}(2) = \) # of eps blocks of size 1 x 2
- \( \text{numbebeps}(3) = \) # of eps blocks of size 2 x 3 etc...

The complete (four blocks) Kronecker form is given by the function `kroneck` which calls `quaskro` on the (pertransposed) pencil \( sE(r) - A(r) \).

The code is taken from T. Beelen

**SEE ALSO:** `kroneck 517`, `gschur 513`, `gspec 514`

---

**2.8.50 randpencil**

**random pencil**

**CALLING SEQUENCE:**

\[ F = \text{randpencil}(\text{eps}, \text{infi}, \text{fin}, \text{eta}) \]

**PARAMETERS:**

- \( \text{eps} \): vector of integers
- \( \text{infi} \): vector of integers
- \( \text{fin} \): real vector, or monic polynomial, or vector of monic polynomial
- \( \text{eta} \): vector of integers
- \( F \): real matrix pencil \( F = sE - A \) \( (s = \text{poly}(0, 's')) \)

**DESCRIPTION:**

Utility function. \( F = \text{randpencil}(\text{eps}, \text{infi}, \text{fin}, \text{eta}) \) returns a random pencil \( F \) with given Kronecker structure. The structure is given by: \( \text{eps} = [\text{eps}1, \ldots, \text{eps}k] \): structure of epsilon blocks (size \( \text{eps}1 \times (\text{eps}1+1), \ldots \)) \( \text{fin} = [\text{infi1}, \ldots, \text{infin}] \) set of finite eigenvalues (assumed real) (possibly []) \( \text{infi} = [\text{infi1}, \ldots, \text{infi}k] \) size of J-blocks at infinity \( ki_i >= 1 \) (\( \text{inf} = [\text{etai1}, \ldots, \text{etap}] \): structure of \( \text{etai} \) blocks (size \( \text{etai1+1} \times \text{etai1} \)).

- \( \text{eps}i \)'s should be \( >=0 \), \( \text{etai} \)'s should be \( >=0 \), \( \text{infi} \)'s should be \( >=1 \).

If \( \text{fin} \) is a (monic) polynomial, the finite block admits the roots of \( \text{fin} \) as eigenvalues.

If \( \text{fin} \) is a vector of polynomial, they are the finite elementary divisors of \( F \); i.e. the roots of \( p(i) \) are finite eigenvalues of \( F \).

**EXAMPLE:**

Scilab Group

April 1993

529
F=randpencil([0,1],[2],[-1,0,1],[3]);
[Q,Z,Qd,Zd,numbepsi,numbetapi]=kroneck(F);
Qd, Zd
s=poly(0,'s');
F=randpencil([],[1,2],s^3-2,[]); //regular pencil
det(F)

SEE ALSO: kroneck 517, pencan 524, penlaur 524

2.8.51 range  

CALLING SEQUENCE:

[X,dim]=range(A,k)

PARAMETERS:

A : real square matrix
k : integer
X : non-singular real matrix
dim : integer (dimension of subspace)

DESCRIPTION:

Computation of Range $A^k$ ; the first dim columns of $X$ span the range of $A^k$.

SEE ALSO: fullrfk 510, rowcomp 531

AUTHOR: F. D.

2.8.52 rank  

CALLING SEQUENCE:

[i]=rank(X)
[i]=rank(X,tol)

PARAMETERS:

X : real or complex matrix
tol : nonnegative real number

DESCRIPTION:

rank(X) is the numerical rank of X i.e. the number of singular values of X that are larger than norm(size(X),'inf') * norm(X) * %eps.

rank(X,tol) is the number of singular values of X that are larger than tol.

REMARK:

Note that the default value of tol is proportional to norm(X). As a consequence rank([[1.d-80,0;0,1.d-80]]) is 2!.

EXAMPLE:

rank([[1.d-80,0;0,1.d-80]])
rank([[1,0;0,1.d-80]])

SEE ALSO: svd 538, qr 528, rowcomp 531, colcomp 505, lu 519
2.8.53 \texttt{rcond} \quad \texttt{inverse condition number}

\textbf{CALLING SEQUENCE :}

\texttt{rcond(X)}

\textbf{PARAMETERS :}

\(X\) : real or complex square matrix

\textbf{DESCRIPTION :}

\(rcond(X)\) is an estimate for the reciprocal of the condition of \(X\) in the 1-norm.

If \(X\) is well conditioned, \(rcond(X)\) is close to 1. If not, \(rcond(X)\) is close to 0.

\([r,z]=rcond(X)\) sets \(r\) to \(rcond(X)\) and returns \(z\) such that

\[\text{norm}(X*z,1) = r*\text{norm}(X,1)*\text{norm}(z,1)\]

Thus, if \(rcond\) is small, \(z\) is a vector in the kernel.

\textbf{EXAMPLE :}

\begin{verbatim}
A=diag([1:10]);
rcond(A)
A(1,1)=0.000001;
rcond(A)
\end{verbatim}

\textbf{SEE ALSO :} \texttt{svd}, \texttt{cond}, \texttt{inv}

\[538,506,516\]

2.8.54 \texttt{rowcomp} \quad \texttt{row compression, range}

\textbf{CALLING SEQUENCE :}

\[\texttt{[W,rk]=rowcomp(A [,flag] [,tol])}\]

\textbf{PARAMETERS :}

\(A\) : real or complex matrix

\texttt{flag} : character string

\texttt{tol} : real number

\(W\) : square non-singular matrix (change of basis)

\(rk\) : integer (rank of \(A\))

\textbf{DESCRIPTION :}

Row compression of \(A\). \(A_c = W*A\) is a row compressed matrix: i.e. \(A_c=[A_f;0]\) with \(A_f\) full row rank.

\texttt{flag} and \texttt{tol} are optional parameters: \texttt{flag='qr'} or \texttt{'svd'} (default \texttt{'svd'}).

\texttt{tol} is a tolerance parameter (of order \texttt{sqrt(%eps)} as default value).

The \texttt{rk} first columns of \(W'\) span the range of \(A\).

The \texttt{rk} first (top) rows of \(W\) span the row range of \(A\).

\textbf{REMARK :}

A non zero vector \(x\) belongs to \texttt{range}(\(A\)) iff \(W*x\) is row compressed in accordance with \(A_c\) i.e the norm of its last components is small w.r.t its first components.

\textbf{EXAMPLE :}

\begin{verbatim}
\end{verbatim}

\textbf{Scilab Group} \quad \textbf{April 1993} \quad \textbf{531}
A = rand(5, 2) * rand(2, 4); // 4 col. vectors, 2 independent.
[X, dim] = rowcomp(A); x = X';
svd([Xp(:, 1:dim), A]) // span(A) = span(Xp(:, 1:dim))
x = A * rand(4, 1); // x belongs to span(A)
y = x * x
norm(y(dim+1:$))/norm(y(1:dim)) // small

SEE ALSO: colcomp 505, fullrf 510, fullrfk 510

AUTHOR: F. D.

2.8.55 rowshuff shuffle algorithm

CALLING SEQUENCE:

[W, F1] = rowshuff(Fs, [alfa])

PARAMETERS:

Fs: square real pencil Fs = s*E-A
Ws: polynomial matrix
F1: square real pencil F1s = s*E1-A1 with E1 non-singular
alfa: real number (alfa = 0 is the default value)

DESCRIPTION:
Shuffle algorithm: Given the pencil Fs = s*E-A , returns Ws=W(s) (square polynomial matrix) such that:
F1s = s*E1-A1 = W(s)*(s*E-A)
is a pencil with non singular E1 matrix.
This is possible iff the pencil Fs = s*E-A is regular (i.e. invertible). The degree of Ws is equal to the
index of the pencil.
The poles at infinity of Fs are put to alfa and the zeros of Ws are at alfa.
Note that (s*E-A)^-1 = (s*E1-A1)^-1 * W(s) = (W(s)*(s*E-A))^1 * W(s)

EXAMPLE:
F = randpencil([], [2], [1, 2, 3], []);
F = rand(5, 5) * F * rand(5, 5); // 5 x 5 regular pencil with 3 evats at 1, 2, 3
[W, F1] = rowshuff(F, -1);
[E1, A1] = pen2ea(F1);
svd(E1) // E1 non singular
roots(det(Ws))
clean(inv(F) - inv(F1) * Ws, 1.d-7)

SEE ALSO: pencan 524, glever 512, penlaur 524

AUTHOR: F. D.

2.8.56 rref computes matrix row echelon form by lu transformations

CALLING SEQUENCE:

R = rref(A)

PARAMETERS:

A : m x n matrix with scalar entries
R : m x n matrix, row echelon form of a
DESCRIPTION:
  rref computes the row echelon form of the given matrix by left lu decomposition. If ones need the
  transformation used just call X=rref([A,eye(m,m)]) the row echelon form \( R \) is \( X(:,1:n) \) and
  the left transformation \( L \) is given by \( X(:,n+1:n+m) \) such as \( L*A=R \)

EXAMPLE:
  
  \[ A=[1 \ 2;3 \ 4;5 \ 6]; \]
  \[ X=rref([A,eye(3,3)]); \]
  \[ R=X(:,1:2); \]
  \[ L=X(:,3:5);L*A \]

SEE ALSO: lu 519, qr 528

2.8.57 \texttt{schur} \hspace{1cm} [ordered] Schur decomposition

CALLING SEQUENCE:
  
  \[ [U,T] = \texttt{schur}(A) \]
  \[ [U,dim]=\texttt{schur}(A,\texttt{flag}) \]
  \[ [U,dim]=\texttt{schur}(A,\texttt{myfunction}) \]

PARAMETERS:
  
  \( A \) : real or complex matrix. For ordered forms \( A \) is assumed real.
  
  \( \texttt{flag} \) : character string (’c’ or ’d’)
  
  \( \texttt{myfunction} \) : an “external” function (this parameter can also be a list or character string)
  
  \( U \) : orthogonal or unitary square matrix
  
  \( T \) : matrix
  
  \( \texttt{dim} \) : integer

DESCRIPTION:
  Schur forms, ordered Schur forms

USUAL SCHUR FORM:
  
  \[ [U,T] = \texttt{schur}(A) \] produces a Schur matrix \( T \) and a unitary matrix \( U \) so that \( A = U*T*U' \) and \( U'*U = \texttt{eye}(U) \). By itself, \texttt{schur}(A) returns \( T \). If \( A \) is complex, the Complex Schur Form is returned in
  matrix \( T \). The Complex Schur Form is upper triangular with the eigenvalues of \( A \) on the diagonal. If \( A \) is
  real, the Real Schur Form is returned. The Real Schur Form has the real eigenvalues on the diagonal and
  the complex eigenvalues in 2-by-2 blocks on the diagonal.

ORDERED STABLE FORM:
  
  \[ [U,dim]=\texttt{schur}(A,'c') \] returns an unitary matrix \( U \) which transforms \( A \) into schur form. In
  addition, the \( \texttt{dim} \) first columns of \( U \) make a basis of the eigenspace of \( A \) associated with eigenvalues with
  negative real parts (stable ”continuous time” eigenspace).
  
  \[ [U,dim]=\texttt{schur}(A,'d') \] returns an unitary matrix \( U \) which transforms \( A \) into schur form. In addition, the \( \texttt{dim} \)
  first columns of \( U \) span a basis of the eigenspace of \( A \) associated with eigenvalues with
  magnitude lower than 1 (stable ”discrete time” eigenspace).

GENERAL EIGENSPACE:
  
  \[ [U,dim]=\texttt{schur}(A,a\_function) \] returns an unitary matrix \( U \) which transforms \( A \) into schur form. In addition, the \( \texttt{dim} \) first columns of \( U \) span a basis of the eigenspace of \( A \) associated with the
  eigenvalues which are selected by the function \( a\_function \).
  
  This function must be of the following type (here \( a\_function \) is ”rule”):

  \[
  \text{function } [\text{flag}]=\text{rule}(x) \\
  \text{flag}=... 
  \]

Scilab Group  
April 1993  
533
$x$ is a vector with three components which characterizes either a real eigenvalue or a pair of complex conjugate eigenvalues.
If $x(1)=1$, a real eigenvalue is considered and this eigenvalue is $x(2)/x(3)$.
If $x(1)=2$, a pair of complex conjugate eigenvalues is considered. The sum of these two eigenvalues (twice the real part) is $x(2)$ and the product (squared magnitude) is $x(3)$.
On return, flag should be 1 if the real eigenvalue is selected or the pair of eigenvalues is selected and 0 otherwise.

**EXAMPLE OF FUNCTION:**

```scilab
def function [flag]=disc(x)
  ls =x(1);flag=0;
  select ls
    case 1 then if abs(x(2)) < ro*abs(x(3)) then flag=1;end
    case 2 then if x(3) < ro*ro then flag=1;end
  end
end
```

The function `disc` selects the eigenvalues with magnitude lower than a given scalar $ro$. And for $ro=1$ the calling sequence `[U,dim]=schur(A,'d')` and `[U,dim]=schur(A,disc)` are equivalent.

Another useful example is `%choose` (see function code in SCIDIR/macros/percent)

**EXAMPLE:**

```scilab
A=diag([-0.9,-2,2,0.9]);X=rand(A);A=inv(X)*A*X;
[U,d]=schur(A,’c’);
A1=U'*A*U;
spec(A1(1:d,1:d)) //stable cont. eigenvalues
[U,d]=schur(A,’c’);
A1=U'*A*U;
spec(A1(1:d,1:d)) //stable disc. eigenvalues
```

**SEE ALSO:** gschur 513, ricc 356, pbig 523, psmall 527

### 2.8.58 spaninter  
subspace intersection

**CALLING SEQUENCE:**

```scilab
[X,dim]=spaninter(A,B [,tol])
```

**PARAMETERS:**

- $A$, $B$ : two real or complex matrices with equal number of rows
- $X$ : orthogonal or unitary square matrix
- $dim$ : integer, dimension of subspace range($A$) inter range($B$)

**DESCRIPTION:**

- $[X,dim]=spaninter(A,B)$ computes the intersection of range($A$) and range($B$).
The first $dim$ columns of $X$ span this intersection i.e. $X(:,1:dim)$ is an orthogonal basis for $\mathcal{R}(A) \cap \mathcal{R}(B)$.
In the $X$ basis $A$ and $B$ are respectively represented by:
- $X'*A$ and $X'*B$.
- $tol$ is a threshold ($\sqrt{\%eps}$) is the default value.

**EXAMPLE:**

```scilab
A=rand(5,3)*rand(3,4); // A is 5 x 4, rank=3
B=[A(:,2),rand(5,1)]*rand(2,2);
[X,dim]=spaninter(A,B);
X1=X(:,1:dim); //The intersection
svd(A),svd([X1,A]) // X1 in span(A)
svd(B),svd([B,X1]) // X1 in span(B)
```

Scilab Group April 1993 534
2.8.59 **spanplus** .................................................. sum of subspaces

**CALLING SEQUENCE:**

\[ \begin{align*} & [X, \text{dim}, \text{dima}] = \text{spanplus}(A, B[, \text{tol}]) \end{align*} \]

**PARAMETERS:**

- \( A, B \): two real or complex matrices with equal number of rows
- \( X \): orthogonal or unitary square matrix
- \( \text{dim}, \text{dima} \): integers, dimension of subspaces
- \( \text{tol} \): nonnegative real number

**DESCRIPTION:**

\[ \begin{align*} & [X, \text{dim}, \text{dima}] = \text{spanplus}(A, B) \text{ computes a basis } X \text{ such that:} \\
& \text{the first } \text{dima} \text{ columns of } X \text{ span } \text{Range}(A) \text{ and the following } (\text{dim} - \text{dima}) \text{ columns make a basis of } A + B \\
& \text{relative to } A. \\
& \text{The dim first columns of } X \text{ make a basis for } A + B. \\
& \text{One has the following canonical form for } [A, B]: \\
& \begin{bmatrix} *, * \end{bmatrix} \text{ (dima rows)} \\
& X'\left[ A, B \right] = \begin{bmatrix} 0, * \end{bmatrix} \text{ (dim-dima rows)} \\
& \begin{bmatrix} 0, 0 \end{bmatrix} \\
& \text{tol} \text{ is an optional argument (see function code).} \end{align*} \]

**EXAMPLE:**

\[ \begin{align*} & A = \text{rand}(6, 2) * \text{rand}(2, 5); \quad // \text{rank}(A) = 2 \\
& B = [A(:, 1), \text{rand}(6, 2)] * \text{rand}(3, 3); \quad // \text{two additional independent vectors} \\
& [X, \text{dim}, \text{dimA}] = \text{spanplus}(A, B); \\
& \text{dimA} \\
& \text{dim} \end{align*} \]

**See Also:** **spaninter 534, im_inv 515, spantwo 535**

**Author:** F. D.

2.8.60 **spantwo** .................................................. sum and intersection of subspaces

**CALLING SEQUENCE:**

\[ \begin{align*} & [Xp, \text{dima}, \text{dimb}, \text{dim}] = \text{spantwo}(A, B, [\text{tol}]) \end{align*} \]

**PARAMETERS:**

- \( A, B \): two real or complex matrices with equal number of rows
- \( Xp \): square non-singular matrix
- \( \text{dima}, \text{dimb}, \text{dim} \): integers, dimension of subspaces
- \( \text{tol} \): nonnegative real number

**DESCRIPTION:**

Given two matrices \( A \) and \( B \) with same number of rows, returns a square matrix \( Xp \) (non singular but not necessarily orthogonal) such that:

\[ \begin{align*} & \text{Scilab Group} \\
& \text{April 1993} \\
& \text{535} \]
The first \( \text{dima} \) columns of \( \text{inv}(X_p) \) span range(A).
Columns \( \text{dim-dimb+1} \) to \( \text{dima} \) of \( \text{inv}(X_p) \) span the intersection of range(A) and range(B).
The \( \text{dim} \) first columns of \( \text{inv}(X_p) \) span range(A)+range(B).
Columns \( \text{dim-dimb+1} \) to \( \text{dim} \) of \( \text{inv}(X_p) \) span range(B).
Matrix \([A_1;A_2]\) has full row rank (=rank(A)). Matrix \([B_2;B_3]\) has full row rank (=rank(B)). Matrix \([A_2,B_2]\) has full row rank (=rank(A inter B)). Matrix \([A_1,0;A_2,B_2;0,B_3]\) has full row rank (=rank(A+B)).

**EXAMPLE:**

A=[1,0,0,4; 5,6,7,8; 0,0,11,12; 0,0,0,16];
B=[1,2,0,0]'; C=[4,0,0,1];
Sl=ss2ss(syslin('c',A,B,C),rand(A));
[no,X]=contr(Sl('A'),Sl('B')); CO=X(:,1:no); //Controllable part
[uo,Y]=unobs(Sl('A'),Sl('C')); UO=Y(:,1:uo); //Unobservable part
[Xp,dimc, dimu, dim]=spantwo(CO,UO); //Kalman decomposition
SICan=ss2ss(Sl,inv(Xp));

**SEE ALSO:** spanplus 535, spaninter 534

**AUTHOR:** F. D.

### 2.8.61 spchol ———— sparse cholesky factorization

**CALLING SEQUENCE:**

\[ [R, P] = \text{spchol}(X) \]

**PARAMETERS:**

X : symmetric positive definite real sparse matrix
P : permutation matrix
R : cholesky factor

**DESCRIPTION:**

\([R, P] = \text{spchol}(X) \) produces a lower triangular matrix \( \text{R} \) such that \( P*R*R'*P' = X \).

**EXAMPLE:**

X=[
3., 0., 0., 2., 0., 0., 2., 0., 0., 0.;
0., 5., 4., 0., 0., 0., 0., 0., 0., 0.;
0., 4., 5., 0., 0., 0., 0., 0., 0., 0.;
2., 0., 0., 3., 0., 0., 2., 0., 2., 0.;
0., 0., 0., 0., 5., 0., 0., 0., 0., 0., 4.;
0., 0., 0., 0., 0., 4., 0., 3., 0., 3., 0.;
2., 0., 0., 0., 2., 0., 3., 0., 2., 0., 0.;
0., 0., 0., 0., 0., 3., 0., 4., 0., 3., 0.;
2., 0., 0., 0., 2., 0., 3., 0., 3., 0., 0.;
0., 0., 0., 0., 0., 3., 0., 3., 0., 4., 0.;
];
0., 0., 0., 4., 0., 0., 0., 0., 5.];
X=sparse(X);[R,P] = spchol(X);
max(P*R*R'*P'-X)

SEE ALSO: sparse 214, lusolve 521, luget 520, chol 503

2.8.62 spec ___________________________ eigenvalues

CALLING SEQUENCE:

evals=spec(A)

PARAMETERS:

A : real or complex square matrix
evals : real or complex vector

DESCRIPTION:

evals=spec(A) returns in vector evals the eigenvalues of A.
Eigenvalues are obtained by bdiag.

EXAMPLE:

A=diag([1,2,3]);X=rand(3,3);A=inv(X)*A*X;
spec(A)
//
x=poly(0,’x’);
pol=det(x*eye()-A)
roots(pol)
//
[Ab,X,bs]=bdiag(A);
Ab
clean(inv(X)*A*X)

SEE ALSO: poly 65, det 507, gspec 514, schur 533, bdiag 502, colcomp 505

2.8.63 sqroot ___________________________ W*W’ hermitian factorization

CALLING SEQUENCE:

sqroot(X)

PARAMETERS:

X : symmetric non negative definite real or complex matrix

DESCRIPTION:

W=sqroot(X) returns W such that X=W*W’ (uses SVD).

EXAMPLE:

X=rand(5,2)*rand(2,5);X=X*X’;
W=sqroot(X)
norm(W*W’-X,1)
//
X=rand(5,2)+%i*rand(5,2);X=X*X’;
W=sqroot(X)
norm(W*W’-X,1)

SEE ALSO: chol 503, svd 538
**2.8.64 sva**

**CALLING SEQUENCE:**

\[ [U,s,V] = \text{sva}(A,k) \]
\[ [U,s,V] = \text{sva}(A,\text{tol}) \]

**PARAMETERS:**

- \( A \): real or complex matrix
- \( k \): integer
- \( \text{tol} \): nonnegative real number

**DESCRIPTION:**

Singular value approximation.

- \([U,S,V] = \text{sva}(A,k)\) with \(k\) an integer \(\geq 1\), returns \(U,S\) and \(V\) such that \(B = U*S*V'\) is the best \(L_2\) approximation of \(A\) with \(\text{rank}(B) = k\).
- \([U,S,V] = \text{sva}(A,\text{tol})\) with \(\text{tol}\) a real number, returns \(U,S\) and \(V\) such that \(B = U*S*V'\) such that \(L_2\)-norm of \(A - B\) is at most \(\text{tol}\).

**EXAMPLE:**

\[ A = \text{rand}(5,4) * \text{rand}(4,5); \]
\[ [U,s,V] = \text{sva}(A,2); \]
\[ B = U*S*V'; \]
\[ \text{svd}(A) \]
\[ \text{svd}(B) \]
\[ \text{clean}(\text{svd}(A-B)) \]

**SEE ALSO:** \(\text{svd} 538\)

**2.8.65 svd**

**CALLING SEQUENCE:**

\[ s = \text{svd}(X) \]
\[ [U,S,V] = \text{svd}(X) \]
\[ [U,S,V] = \text{svd}(X,0) \]
\[ [U,S,V,\text{rk}] = \text{svd}(X [,\text{tol}]) \]

**PARAMETERS:**

- \( X \): a real or complex matrix
- \( s \): real vector (singular values)
- \( S \): real diagonal matrix (singular values)
- \( U, V \): orthogonal or unitary square matrices (singular vectors).
- \( \text{tol} \): real number

**DESCRIPTION:**

- \([U,S,V] = \text{svd}(X)\) produces a diagonal matrix \(S\), of the same dimension as \(X\) and with non-negative diagonal elements in decreasing order, and unitary matrices \(U\) and \(V\) so that \(X = U*S*V'\).
- \([U,S,V] = \text{svd}(X,0)\) produces the "economy size" decomposition. If \(X\) is \(m\)-by-\(n\) with \(m > n\), then only the first \(n\) columns of \(U\) are computed and \(S\) is \(n\)-by-\(n\).
- \(s = \text{svd}(X)\) by itself, returns a vector \(s\) containing the singular values.
- \([U,S,V,\text{rk}] = \text{svd}(X,\text{tol})\) gives in addition \(\text{rk}\), the numerical rank of \(X\) i.e. the number of singular values larger than \(\text{tol}\).

The default value of \(\text{tol}\) is the same as in \text{rank}.

**EXAMPLE:**

Scilab Group April 1993
trace Scilab Function

X=rand(4,2)*rand(2,4)
svd(X)
sqrt(spec(X*X'))

SEE ALSO: rank 530, qr 528, colcomp 505, rowcomp 531, sva 538, spec 537

2.8.66 sylv .......................................... Sylvester equation.

CALLING SEQUENCE:
sylv(A,B,C,flag)

PARAMETERS:
A,B,C : three real matrices of appropriate dimensions.
flag : character string ('c' or 'd')

DESCRIPTION:
X = sylv(A,B,C,'c') computes X, solution of the "continuous time" Sylvester equation
A*X+X*B=C
X=sylv(A,B,C,'d') computes X, solution of the "discrete time" Sylvester equation
A*X*B-X=C

EXAMPLE:
A=rand(4,4);C=rand(4,3);B=rand(3,3);
X = sylv(A,B,C,'c');
norm(A*X+X*B-C)
X=sylv(A,B,C,'d')
norm(A*X*B-X-C)

SEE ALSO: lyap 522

2.8.67 trace .......................................... trace

CALLING SEQUENCE:
trace(X)

PARAMETERS:
X : real or complex square matrix, polynomial or rational matrix.

DESCRIPTION:
trace(X) is the trace of the matrix X.
Same as sum(diag(X)).

EXAMPLE:
A=rand(3,3);
trace(A)-sum(spec(A))

SEE ALSO: det 507
2.9 Metanet
2.9.1  add_edge  ________________  adds an edge or an arc between two nodes

CALLING SEQUENCE :

g1 = add_edge(i, j, g)

PARAMETERS :

i : integer, number of start node
j : integer, number of end node
g : graph list
g1 : graph list of the new graph with the added edge

DESCRIPTION :
add_edge returns the graph g1 with a new edge from node number i to node number j. If the graph is directed, the edge is an arc. The number of edges plus 1 is taken as the name of the new edge.

EXAMPLE :

ta=[1 1 2 2 3 4 5 7 8 9 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);
g=add_edge(1,7,g);
g('edge_color')=[ones(ta) 11];
show_graph(g);

SEE ALSO:  add_node 541,  delete_arcs 551,  delete_nodes 552

2.9.2  add_node  _____________________  adds a disconnected node to a graph

CALLING SEQUENCE :

g1 = add_node(g, [xy, name])

PARAMETERS :

g : graph list
xy : optional row vector of coordinates of new node
name : optional name of the added node
g1 : graph list of the new graph with the added node

DESCRIPTION :
add_node adds a disconnected node to graph g and returns the new graph g1.

The coordinates of the new node can be given as a row vector of coordinates in xy. If the nodes of graph g have no coordinates (elements node_x and node_y are []), to give xy has no effect. If the nodes of graph g have coordinates and xy is not given, the new node has (0,0) as coordinates.

If name is given, it is the name of the new node, otherwise the number of nodes plus 1 is taken as the name of the new node.

EXAMPLE :
arc_graph Scilab function

ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 14 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757
642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151
301];
show_graph(g);
n=g('node_number');
g1=add_node(g,[270 140]);
g1('node_color')=[ones(1,n) 11];
show_graph(g1);

SEE ALSO:  add_edge 541,  delete_arcs 551,  delete_nodes 552

2.9.3  adj_lists  computes adjacency lists

CALLING SEQUENCE:

[lp,la,ls] = adj_lists(g)
[lp,la,ls] = adj_lists(directed,n,tail,head)

PARAMETERS:

g : graph list
directed : integer, 0 (undirected graph) or 1 (directed graph)
n : integer, the number of nodes of the graph
tail : the row vector of the numbers of the tail nodes of the graph (its size is the
number of edges of the graph)
head : the row vector of the numbers of the head nodes of the graph (its size is the
number of edges of the graph)
lp : row vector, pointer array of the adjacency lists description of the graph (its size is the
number of nodes of the graph + 1)
la : row vector, arc array of the adjacency lists description of the graph (its size is the number of
edges of the graph)
ls : row vector, node array of the adjacency lists description of the graph (its size is the number of
edges of the graph)

DESCRIPTION:

adj_lists computes the row vectors of the adjacency lists description of the graph g. It is also possible
to give adj_lists the description of the graph given by the number of nodes n and the row vectors
tail and head.

EXAMPLE:

ta=[2 3 3 5 3 4 4 5 8];
he=[1 2 4 2 6 6 7 7 4];
g=make_graph('foo',1,8,ta,he);
g('node_x')=[129 200 283 281 128 366 122 236];
g('node_y')=[61 125 129 189 173 135 236 249];
show_graph(g);
[lp,la,ls]=adj_lists(g)
[lp,la,ls]=adj_lists(1,g('node_number'),ta,he)

SEE ALSO:  chain_struct 546,  graph_2_mat 558
2.9.4  arc_graph  ____________________  graph with nodes corresponding to arcs

CALLING SEQUENCE :

\[ g1 = \text{arc\_graph}(g) \]

PARAMETERS :

- \( g \) : graph list of the old graph
- \( g1 \) : graph list of the new graph

DESCRIPTION :

\text{arc\_graph}  returns the directed graph \( g1 \)  with the nodes corresponding to the arcs of the directed graph \( g \). \( g1 \) is defined in the following way:

- its nodes correspond to the arcs of \( g \)
- 2 nodes of the new graph are adjacent if and only if the corresponding arcs of the graph \( g \) are consecutive.

The coordinates of the nodes of \( g1 \) are given by the middle points of the corresponding edges of \( g \).

If such an arc graph does not exist, an empty vector is returned.

EXAMPLE :

```plaintext
ta=[1 1 2 4 4 5 6 7 2 3 5 1];
he=[2 3 6 7 8 9 4 7 3 5]
\text{g} = \text{make\_graph('foo',1,8,ta,he)};
g('node\_x')=[281 284 360 185 405 182 118 45];
g('node\_y')=[262 179 130 154 368 248 64 309];
\text{show\_graph(g)};
g1=\text{arc\_graph(g)};
\text{show\_graph(g1,'new')};
```

SEE ALSO:  line_graph 564

2.9.5  arc_number  _______________________  number of arcs of a graph

CALLING SEQUENCE :

\[ ma = \text{arc\_number}(g) \]

PARAMETERS :

- \( g \) : graph list
- \( ma \) : integer, number of arcs

DESCRIPTION :

\text{arc\_number}  returns the number \( ma \) of arcs of the graph. If the graph is directed, it is the number of edges. If the graph is undirected, it is twice the number of edges.

SEE ALSO:  edge_number 553,  node_number 579
2.9.6  **articul**                           finds one or more articulation points

**CALLING SEQUENCE:**

\[ \text{nart} = \text{articul}([i], g) \]

**PARAMETERS:**

- \( g \): graph list
- \( i \): integer
- \( \text{nart} \): integer row vector

**DESCRIPTION:**

**articul** finds one or more articulation points (if they exist) of the graph \( g \). \( \text{nart} \) is the row vector of numbers of articulation nodes: deleting one of these nodes increases the number of connected components of the graph. \( i \) is the optional node number from which the algorithm starts. The default is 1. Note that the result depends strongly on this starting node.

**EXAMPLE:**

```plaintext
ta=[2 1 3 2 2 4 4 5 6 7 8 9 10 10 10 11 12 13 14 15 16 17 17];
he=[1 10 2 5 7 3 2 4 5 8 6 9 7 7 11 13 15 12 13 14 11 16 17 14 15];
g=make_graph('foo',i,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
nart = articul(g);
show_nodes(nart);
```

2.9.7  **bandwr**                          bandwidth reduction for a sparse matrix

**CALLING SEQUENCE:**

```plaintext
[iperm,mrepi,prof,ierr] = bandwr(sp,[iopt])
[iperm,mrepi,prof,ierr] = bandwr(lp,ls,n,[iopt])
```

**PARAMETERS:**

- \( sp \): sparse matrix
- \( lp \): integer row vector
- \( ls \): integer row vector
- \( n \): integer
- \( iopt \): integer
- \( iperm \): integer row vector
- \( mrepi \): integer row vector
- \( prof \): integer row vector
- \( ierr \): integer

**DESCRIPTION:**

**bandwr** solves the problem of bandwidth reduction for a sparse matrix: the matrix is supposed to be upper triangular with a full diagonal (it is easy to complete a non symmetric matrix, and then discards the added terms).
In the first calling sequence, \texttt{sp} denotes a sparse matrix; the optional argument \texttt{iopt} is 0 or 1: 1 if reducing the profile of the matrix is more important than reducing the bandwidth and 0 if bandwidth reduction is most important.

The second calling sequence corresponds to the description of a graph: \texttt{lp} is a row vector, pointer array of the adjacency lists description of a graph (its size is the number of nodes of the graph + 1); \texttt{ls} is a row vector, node array of the adjacency lists description (its size is the number of edges of the graph i.e. the number of non-zero terms of the corresponding sparse matrix). \texttt{n} is the number of nodes (dimension of \texttt{sp}).

\texttt{iperm} is the permutation vector for reordering the rows and columns which reduces the bandwidth and/or profile (new numbering of the nodes of the graph); \texttt{mrepi} is the inverse permutation (\texttt{mrepi(iperm)} is the identity). \texttt{prof} is the array giving the profile of the sparse matrix after the bandwidth reduction if \texttt{iopt} is 1. If \texttt{iopt} is 0 this array is zero except for the first term giving the bandwidth. The simple command 

\begin{verbatim}
max(prof(2:$)-prof(1:($-1)))
\end{verbatim}

returns the bandwidth of the matrix. \texttt{ierr} is an integer indicating an error if its value is not zero.

**EXAMPLE**:

\begin{verbatim}
ta=[2 1 3 2 2 4 4 5 6 7 8 8 9 10 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[1 10 2 5 7 3 2 4 5 6 9 7 7 11 13 15 12 13 9 14 11 16 1 17 14 15];
g=make_graph('foo',0,17,ta,he);
g('node_x')=[283 163 63 146 164 273 271 339 141 209 319 221 324 432 141 209 319 428 443 187 151 301];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
// THE GRAPH
show_graph(g);

a=graph_2_mat(g,'node-node');
ww=tril(a)+eye();
ww1=full(ww);
xset('window',0)
hist3d((ww1+tril(ww1,-1)+tril(ww1,-1)'),52,85);

// BANDWIDTH REDUCTION FOR THE MATRIX
[iperm,mrepi,prof,ierr]=bandwr(ww);
max(prof(2:$)-prof(1:($-1)))

// GRAPH WITH THE NEW NUMBERING

// NEW MATRIX
n=g('node_number');

yy=ww1(mrepi,mrepi);
xset('window',1)
hist3d((yy+tril(yy,-1)+tril(yy,-1)'),52,85);

// STARTING WITH THE SAME MATRIX

[ij,v,mn]=spget(ww);
gl=make_graph('foo',0,n,ij(:,1)',ij(:,2)');
gl('node_x')=g('node_x');gl('node_y')=g('node_y');

// GRAPH

//show_graph(gl,'rep');

[ipperm,mrepi,prof,ierr]=bandwr(lp,ls,n,0);
g2=g;g2('node_name')=string(ipperm);
show_graph(g2,'new');
\end{verbatim}
CALLING SEQUENCE:

```matlab
[card,match] = best_match(g)
```

PARAMETERS:

- `g`: graph list
- `card`: integer
- `match`: integer row vector

DESCRIPTION:

`best_match` finds an optimal matching for the graph `g`. The output are `card` and the vector `match`. `card` is the cardinality of an optimal matching. `match(i)` is the node adjacent to node `i` in the optimal matching or 0 if `i` is unmatched.

EXAMPLE:

```matlab
ta = [27 27 3 12 11 12 27 26 25 24 23 23 21 22 21 20 18 18 18];
ta = [ta 16 15 14 12 9 10 6 9 17 8 17 10 20 11 23 23 12 18 28];
he = [1 2 2 4 5 11 13 1 25 22 24 22 22 19 13 13 14 16 16 9 16];
he = [he 10 10 10 12 2 6 5 5 7 8 7 9 6 11 4 18 13 3 28 17];
n = 28;
g = make_graph('foo', 0, n, ta, he);
xx = [46 120 207 286 366 453 543 544 473 387 300 206 136 250 346 408];
g('node_x') = [xx 527 443 306 326 196 139 264 55 58 46 118 513];
yy = [36 34 37 40 38 40 35 102 102 98 93 96 167 172 101 179];
g('node_y') = [yy 198 252 183 148 172 256 259 258 167 109 104 253];
show_graph(g);
```

```matlab
[card,match] = best_match(g);
sp = sparse([ta' he'], [1:size(ta,2)], [n,n]);
sp1 = sparse([1:n', match'], ones(1, size(match,2)), [n,n]);
[ij, v, mn] = spget(sp * sp1);
show_arcs(v');
```

SEE ALSO: `perfect_match` 581

2.9.9 `chain_struct` chained structure from adjacency lists of a graph

CALLING SEQUENCE:

```matlab
[fe, che, fn, chn] = chain_struct(g)
[fe, che, fn, chn] = chain_struct(lp, la, ls)
```

PARAMETERS:

- `g`: graph list
- `lp`, `la`, `ls`: adjacency lists

Scilab Group

September 1996

546
lp : row vector, pointer array of the adjacency lists description of the graph (its size is the number of nodes of the graph + 1)
la : row vector, arc array of the adjacency lists description of the graph (its size is the number of edges of the graph)
ls : row vector, node array of the adjacency lists description of the graph (its size is the number of edges of the graph)
fe : row vector of the numbers of the first edges starting from nodes (its size is the number of nodes of the graph)
che : row vector of the numbers of the chained edges (its size is the number of edges of the graph)
fn : row vector of the numbers of the first nodes reached by the edges of fe (its size is the number of nodes of the graph)
chn : row vector of the nodes reached by the edges of che

DESCRIPTION:
chain_struct computes the row vectors of the edge chained structure description of the graph g. It is also possible to give directly chain_struct the adjacency lists of the graph. This is more efficient if the adjacency lists are already available since chain_struct uses them to make computations.

The vectors fe, che, fn and chn describe the chained structure in the following way:
fe(i) is the number of the first edge starting from node i
che(fe(i)) is the number of the second edge starting from node i, che(che(fe(i))) is the number of the third edge starting from node i and so on until the value is 0
fn(i) is the number of the first node reached from node i
ch(i) is the number of the node reached by edge che(i).

EXAMPLE:

```
ta=[1 1 2 3 5 4 6 7 7 3 3 8 8 5];
he=[2 3 5 4 6 7 4 3 2 8 1 7 4];
g=make_graph(‘foo’,1,8,ta,he);
g(‘node_x’)=[116 231 192 323 354 454 305 155];
g(‘node_y’)=[118 116 212 219 117 185 334 316];
show_graph(g);
[fe,che,fn,chn] = chain_struct(g)
```

SEE ALSO: adj_lists 542, graph_2_mat 558

2.9.10 check_graph ____________________________ checks a Scilab graph list

CALLING SEQUENCE:
check_graph(g)

PARAMETERS:
g : graph list to check

DESCRIPTION:
check_graph checks its argument g to see if it is a graph list. The checking is not only syntactic (number of elements of the list, compatible sizes of the vectors), but also semantic in the sense that check_graph checks that node, number, tail and head elements of the list can really represent a graph.

Moreover, the names of the node must be different. In fact, this do not give errors in Scilab, but strange behaviour can appear when using the Metanet window. So, this is not checked by check_graph because it is time consuming. It is only checked when loading, saving or showing a graph.

SEE ALSO: graph-list 555
2.9.11  circuit __________ finds a circuit or the rank function in a directed graph

CALLING SEQUENCE:

[p, r] = circuit(g)

PARAMETERS:

g : graph list
p : row vector of integer numbers of the arcs of the circuit if it exists
r : row vector of rank function if there is no circuit

DESCRIPTION:

circuit tries to find a circuit for the directed graph g. It returns the circuit p as a row vector of the corresponding arc numbers if it exists and it returns the empty vector [] otherwise. If the graph has no circuit, the rank function is returned in r, otherwise its value is the empty vector [].

EXAMPLE:

// graph with circuit
ta=[1 1 2 3 5 4 6 7 7 3 3 8 8 5];
he=[2 3 5 4 6 6 7 4 3 2 8 1 7 4];
g=make_graph('foo',1,8,ta,he);
g('node_x')=[116 231 192 323 354 454 305 155];
g('node_y')=[118 116 212 219 117 185 334 316];
show_graph(g);
p=circuit(g)
show_arcs(p)

// graph without circuit
[g,'node_x']=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];

2.9.12  con_nodes ___________________ set of nodes of a connected component

CALLING SEQUENCE:

ns = con_nodes(i, g)

PARAMETERS:

i : integer, number of the connected component
g : graph list
ns : row vector, node numbers of the connected component

DESCRIPTION:

con_nodes returns the row vector ns of the numbers of the nodes which belong to the connected component number i. If i is not the number of a connected component, the empty vector [] is returned.

EXAMPLE:

ta=[1 1 2 2 2 3 4 4 5 7 7 9 10 12 12 13 14 15];
he=[2 6 3 4 5 1 3 5 1 8 9 8 11 10 11 15 13 14];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);
con_nodes(2,g)
x_message('Displaying the nodes of component #2');
n=g('node_number');
nodcolor=0*ones(1,n);
nodcolor(1,con_nodes(2,g))=11*ones(con_nodes(2,g));
g('node_color')=nodcolor;
nodediam=20.*ones(1,n);
nodediam(1,con_nodes(2,g))=30*ones(con_nodes(2,g));
g('node_diam')=nodediam;
show_graph(g);

SEE ALSO: connex 549, is_connex 563, strong_connex 590, strong_con_nodes 589

2.9.13 connex ___________________________ connected components

CALLING SEQUENCE:

[nc,ncomp] = connex(g)

PARAMETERS:

g : graph list
nc : integer, number of connected components
ncomp : row vector of connected components

DESCRIPTION:

connex returns the number nc of connected components of graph g and a row vector ncomp giving
the number of the connected component for each node. For instance, if i is a node number, ncomp[i]
is the number of the connected component to which node number i belongs.

EXAMPLE:

ta=[1 1 2 2 3 4 4 5 6 7 7 7 8 9 10 12 12 13 13 14 15];
he=[2 6 3 4 5 1 3 5 1 7 5 8 9 5 8 11 10 11 11 15 13 14];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);
[nc,ncomp]=connex(g)
g('node_color')=10+ncomp;
g('node_diam')=10+10*ncomp;
x_message('Displaying the connected components of this graph');
show_graph(g);

SEE ALSO: con_nodes 548, is_connex 563, strong_connex 590, strong_con_nodes 589

2.9.14 contract_edge ____________________ contracts edges between two nodes

CALLING SEQUENCE:

g1 = contract_edge(i,j,g)
PARAMETERS:

i : integer, number of start or end node of edge
j : integer, number of end or start node of edge
g : graph list
g1 : graph list of the new graph

DESCRIPTION:
contract_edge returns the graph g1, the edges between the nodes number i and j being deleted, the nodes being reduced to one node with the same name as node i and located at the middle point between the 2 previous nodes.

EXAMPLE:

ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 10 11 12 13 13 13 14 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);
g1=contract_edge(10,13,g);
show_graph(g1,'new');

SEE ALSO: add_edge 541, add_node 541, delete_arcs 551, delete_nodes 552

2.9.15 convex_hull ______________________ convex hull of a set of points in the plane

CALLING SEQUENCE:

[nhull,ind] = convex_hull(xy)

PARAMETERS:

xy : 2 x n real matrix
nhull : integer
ind : integer row vector

DESCRIPTION:
convex_hull finds the convex hull of a given set of n points in the plane. xy is the 2 x n matrix of the (x,y) coordinates of the given points. convex_hull returns in nhull the number of the points of the boundary of the convex hull and in ind the row vector (of size nhull) giving the indices in xy of the points of the boundary. The order in ind corresponds to consecutive points on the boundary.

EXAMPLE:

ta=[27 27 3 12 11 12 27 26 26 25 25 24 23 23 21 22 21 20 19 18 18];
ta=[ta 16 15 14 12 9 10 6 9 17 8 17 10 20 11 23 23 12 18 28];
he=[1 2 2 4 5 11 13 1 25 22 24 22 22 19 13 13 14 16 16 9 16];
he=[he 10 10 11 12 2 6 5 5 7 8 7 9 6 11 4 18 13 3 28 17];
g=make_graph('foo',0,128,ta,he);
x=[46 120 207 286 366 453 543 544 473 387 300 206 136 250 346 408];
g('node_x')=[x 527 443 306 326 196 139 264 55 58 46 118 513];
xx=[36 34 37 40 38 40 35 102 102 98 93 96 167 172 101 179];
g('node_y')=[yy 198 252 183 148 172 256 259 258 167 109 104 253];
show_graph(g);
xy=[g('node_x');g('node_y')];
[nhull,ind] = convex_hull(xy)
show_nodes(ind);

2.9.16  cycle_basis ____________ basis of cycle of a simple undirected graph

CALLING SEQUENCE :

spc = cycle_basis(g)

PARAMETERS :

g : graph list
spc : sparse matrix

DESCRIPTION :

First a spanning tree is found by using min_weight_tree and then used to find all fundamental cycles with respect to this tree. They are returned as a set of cycles, each cycle being represented by a set of edges. These cycles are returned in a sparse matrix spc: each line of this matrix corresponds to a cycle.

The graph g is supposed to be a simple undirected and connected graph (cycle_basis does not check that the graph is simple, use graph_simp before calling it if necessary).

EXAMPLE :

ta=[1 1 2 2 3 4 5 7 8 8 9 10 10 10 11 12 13 13 14 15 16 16 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13 9 10 14 11 16 1 17 14 15];
gt=make_graph('foo',1,17,ta,he);
gt('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
gt('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
gt('edge_color')=modulo([1:(edge_number(gt))],15)+1;
gt('node_diam')=[1:(gt('node_number'))]+20;
show_graph(gt);
g=graph_simp(gt);
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=gt('node_diam');
g('default_edge_hi_width')=12;
show_graph(g);
spc=cycle_basis(g);
for kk=1:(size(spc,1)),
   aaa=spc(kk,:);aaa=full(aaa);aaa(aaa==0)=[];
   show_arcs(aaa);
end;

SEE ALSO:  min_weight_tree 577,  graph_simp 561

2.9.17  delete_arcs __________ deletes all the arcs or edges between a set of nodes

CALLING SEQUENCE :
g1 = delete_arcs(ij, g)

PARAMETERS:
ij : matrix of integers (number of nodes)
g : graph list
g1 : graph list of the new graph without the arcs or edges defined by ij

DESCRIPTION:
If g is a directed graph, delete_arcs returns the graph g1 with the arcs defined by matrix ij being deleted. ij must be a n x 2 matrix of node numbers: the n arcs to be deleted are defined by couples of nodes (ij(i,1),ij(i,2)).

If g is an undirected graph, the edges corresponding to matrix ij are deleted.

EXAMPLE:

```plaintext
ta=[1 1 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);
ij=[13 10; 8 6; 5 4; 2 2];
gt=delete_arcs(ij, g);
show_graph(gt,'new');
g('directed')=0;
gt=delete_arcs(ij, g);
show_graph(gt,'new');
```

SEE ALSO: add_edge 541, add_node 541, delete_nodes 552

2.9.18 delete_nodes delete_nodes deletes nodes

CALLING SEQUENCE:

```
g1 = delete_nodes(v, g)
```

PARAMETERS:

v : vector of integers, numbers of nodes to be deleted
g : graph list
g1 : graph list of the new graph with deleted nodes

DESCRIPTION:

delete_nodes returns the graph g1, with the nodes given by the vector v being deleted.

EXAMPLE:

```plaintext
ta=[1 1 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
```
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);
v=[10 13 4];
gt=delete_nodes(v,g);
show_graph(gt,'new');

SEE ALSO: add_edge 541, add_node 541, delete_arcs 551

2.9.19 edge_number ______________________________ number of edges of a graph

CALLING SEQUENCE:
ma = edge_number(g)

PARAMETERS:
g : graph list
m : integer, number of edges

DESCRIPTION:
edge_number returns the number m of edges of the graph. If the graph is directed, it is the number of arcs. If the graph is undirected, it is half the number of edges. It is always equal to the dimension of g('tail') and g('head').

SEE ALSO: arc_number 543, node_number 579

2.9.20 find_path ______________________________ finds a path between two nodes

CALLING SEQUENCE:
p = find_path(i,j,g)

PARAMETERS:
i : integer, number of start node
j : integer, number of end node
g : graph list
p : row vector of integer numbers of the arcs of the path if it exists

DESCRIPTION:
find_path returns a path p from node number i to node number j if one exists, and the empty vector [] otherwise.

EXAMPLE:
ta=[1 1 2 2 3 4 5 5 7 8 8 9 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',i,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);
p=find_path(1,14,g);
edgcolor=1*ones(ta); edgcolor(p)=11*ones(p); g('edge_color')=edgcolor;
show_graph(g); show_arcs(p);
2.9.21 \hspace{1cm} \texttt{gen\_net} \hspace{1cm} \text{generation of a network}

**CALLING SEQUENCE:**

\[
g = \text{gen\_net} (\text{name}, \text{directed}, \text{v})
g = \text{gen\_net} ()
\]

**PARAMETERS:**

- **name**: string, the name of the graph
- **directed**: integer, 0 (undirected graph) or 1 (directed graph)
- **v**: row vector with 12 values for defining the network
- **g**: graph list

**DESCRIPTION:**

\texttt{gen\_net} generates a network \( g \). The arguments are the name of the graph, a flag equal to 0 (undirected graph) or to 1 (directed graph) and a vector describing the network (see below).

If no argument are given, a dialog box for the definition of all the arguments is opened.

- \( v \) must be a row vector with 12 values. The meaning of the values are:
  - Seed for random: used for initialization of random generation
  - Number of nodes
  - Number of sources
  - Number of sinks
  - Minimum cost
  - Maximum cost
  - Input supply
  - Output supply
  - Minimum capacity
  - Maximum capacity
  - Percentage of edges with costs: between 0 and 100
  - Percentage of edges with capacities: between 0 and 100

The cost of edges without cost are put to minimum cost. The maximum capacity of edges without capacity are put to maximum supply.

The result is a network \( g \) built on a planar connected graph, by using a triangulation method. Moreover, computations are made in order to have a coherent network. Values of costs and maximum capacities are put on the edges. Minimum capacities are reduced to 0.

**EXAMPLE:**

\[
v=\left[1,10,2,1,0,10,100,100,0,100,50,50\right];
g=\text{gen\_net} ('foo',1,v);
\text{show\_graph} (g);
// generating using dialogs
\]

**SEE ALSO:** \texttt{mesh2d} \hspace{1cm} 569
2.9.22 girth ____________________________ girth of a directed graph

CALLING SEQUENCE:

\[ d = \text{girth}(g) \]

PARAMETERS:

- \( g \): graph list
- \( d \): integer

DESCRIPTION:

\text{girth} computes the length (number of arcs) of the shortest cycle in an unweighted directed graph \( g \).

EXAMPLE:

\[
\begin{align*}
ta &= [1 \ 6 \ 2 \ 4 \ 7 \ 5 \ 6 \ 8 \ 4 \ 3 \ 5 \ 1]; \\
he &= [2 \ 1 \ 3 \ 6 \ 4 \ 8 \ 7 \ 2 \ 7 \ 3 \ 5]; \\
g &= \text{make_graph}('foo',1,8,ta,he); \\
g('node_x') &= [285 \ 284 \ 335 \ 160 \ 405 \ 189 \ 118 \ 45]; \\
g('node_y') &= [266 \ 179 \ 83 \ 176 \ 368 \ 252 \ 64 \ 309]; \\
\text{show_graph}(g); \\
d &= \text{girth}(g)
\end{align*}
\]

2.9.23 glist ________________________________ graph list creation

CALLING SEQUENCE:

\[ g = \text{glist}(a1, \ldots, a34) \]

DESCRIPTION:

\text{glist}(a1, \ldots, a34) is a shortcut to \text{tlist}(['graph', 'name', 'directed', 'node_number', 'tail', 'head', 'node_name', 'node_type', 'node_x', 'node_y', 'node_color', ... 'node_diam', 'node_border', 'node_style', 'node_font_style', 'node_font_size', 'node_fill', ... 'default_node_width', 'default_node_height', ... 'default_font_size', 'node_label', 'edge_label', ... 'a34}). It is a low level function to create graph lists, mainly used by programmers. No checking is done. For standard creation of graph lists, use \text{make_graph}.

SEE ALSO: \text{check_graph} 547, \text{graph-list} 555, \text{make_graph} 566

2.9.24 graph-list ____________________________ description of graph list

DESCRIPTION:

A graph in Scilab is represented by a Scilab typed list. We call it a graph list.

You will find below the complete description of the list. Each element is described by one or more lines. The first line gives the name of the element and its definition. Additional informations, such as the default for elements that can have one, are given in the other lines. Indeed, only the 5 first elements must have a value in the list, all the others can be given the empty vector \([\ ]\) as a value, and then the default is used when it is needed by functions or by the Metanet window. For instance, you can define a graph list by

\[ g = \text{make_graph}('min',1,1,[1],[1]); \]
which is the simplest graph you can create in Metanet (it is directed, has one node and one loop arc on this node).

The name of the element in the list is very important because it is used to access the elements of the list. For instance, if \( g \) is a graph list, to get the name of the graph, you only have to do:

\[
g(\text{'name'})
\]

and if you want to change the name of the graph to ‘toto’, you have to do:

\[
g(\text{'name'})='toto';
\]

Moreover, you can get the number of edges and the number of arcs of the graph by using `edge_number(g)` and `arc_number(g)` (these names do not correspond to elements of the list). For compatibility, `node_number(g)` can also be used instead of `g('node_number')`.

A graph list can be syntactically correct but not represent a good graph. You can use the function `check_graph` to check it. Moreover it is a good idea to give nodes different names. In fact, this does not give errors in Scilab, but strange behaviour can appear when using the Metanet window. This is not checked by `check_graph` because it is time consuming. It is only checked when loading, saving or showing a graph.

The elements of a graph list are given below:

- **name**: the name of the graph
  - it is a string with a maximum of 80 characters
- **directed**: flag giving the type of the graph
  - it is equal to 1 (graph directed) or equal to 0 (graph undirected)
- **node_number**: number of nodes
- **tail**: row vector of the tail node numbers
- **head**: row vector of the head node numbers
- **node_name**: row vector of node names
  - the names of the nodes must be different
  - default is the node numbers as node names
- **node_type**: row vector of the node types
  - the type is an integer from 0 to 2, default is 0 (plain node):
    - 0 = plain node
    - 1 = sink node
    - 2 = source node
- **node_x**: row vector of the x coordinate of the nodes
  - default is computed
- **node_y**: row vector of the y coordinate of the nodes
  - default is computed
- **node_color**: row vector of the node colors
  - the color is an integer from 0 to 16, default is 0 (default foreground):
    - 0 = default foreground
    - 1 = navyblue
    - 2 = blue
    - 3 = skyblue
    - 4 = aquamarine
    - 5 = forestgreen
    - 6 = green
    - 7 = lightcyan
    - 8 = cyan
    - 9 = orange
    - 10 = red
    - 11 = magenta
    - 12 = violet
    - 13 = yellow
    - 14 = gold
- 15 = beige
- 16 = background

**node_diam**: - row vector of the size of the node diameters in pixels
  - a node is drawn as a circle
  - default is the value of element `default_node_diam`

**node_border**: - row vector of the size of the node borders in pixels
  - a node is drawn as a circle
  - default is the value of element `default_node_border`

**node_font_size**: - row vector of the size of the font used to draw the name of the node
  - you can choose 8, 10, 12, 14, 18 or 24
  - default is the value of element `default_font_size`

**node_demand**: - row vector of the node demands
  - default is 0

**edge_name**: - row vector of the edge names
  - it is better that the names of the edges are different, but this is not an error
  - default is the edge numbers as edge names

**edge_color**: - row vector of the edge colors
  - the color is an integer from 0 to 16 (see `node_color`)
  - default is 0 (default foreground)

**edge_width**: - row vector of the size of the edge widths in pixels
  - default is the value of element `default_edge_width`

**edge_hi_width**: - row vector of the size of the highlighted edge widths in pixels
  - default is the value of element `default_edge_hi_width`

**edge_font_size**: - row vector of the size of the fonts used to draw the name of the edge
  - you can choose 8, 10, 12, 14, 18 or 24
  - default is the value of element `default_font_size`

**edge_length**: - row vector of the edge lengths
  - default is 0

**edge_cost**: - row vector of the edge costs
  - default is 0

**edge_min_cap**: - row vector of the edge minimum capacities
  - default is 0

**edge_max_cap**: - row vector of the edge maximum capacities
  - default is 0

**edge_q_weight**: - row vector of the edge quadratic weights
  - default is 0

**edge_q_orig**: - row vector of the edge quadratic origins
  - default is 0

**edge_weight**: - row vector of the edge weights
  - default is 0

**default_node_diam**: - default size of the node diameters of the graph
  - default is 20 pixels

**default_node_border**: - default size of the node borders of the graph
  - default is 2 pixels

**default_edge_width**: - default size of the edge widths of the graph
  - default is 1 pixel

**default_edge_hi_width**: - default size of the highlighted edge widths of the graph
  - default is 3 pixels

**default_font_size**: - default size of the font used to draw the names of nodes and edges
  - default is 12

**node_label**: - row vector of node labels

**edge_label**: - row vector of edge labels

**EXAMPLE**:

```scilab
g=load_graph(SCI+'demos/metanet/mesh100');
```

Scilab Group September 1996 557
g('node_color')=int(rand(1:g('node_number'))*16);
g('edge_color')=int(rand(1:edge_number(g))*16);
show_graph(g)

SEE ALSO: arc_number 543, check_graph 547, edge_number 553, glist 555, make_graph 566, node_number 579

2.9.25  graph_2_mat  node-arc or node-node incidence matrix of a graph

CALLING SEQUENCE:

a = graph_2_mat(g,mat)

PARAMETERS:

  g : graph list
  mat : optional string, 'node-arc' or 'node-node' matrix
  a : sparse node-arc or node-node incidence matrix

DESCRIPTION:

  graph_2_mat computes the node-arc or the node-node incidence matrix corresponding to the graph g. If the optional argument mat is omitted or is the string 'node-arc', the node-arc matrix is computed. If mat is the string 'node-node', the node-node matrix is computed. If n is the number of nodes of the graph and m is the number of edges of the graph, the node-arc matrix is a Scilab sparse matrix of size (n,m).
  It is defined as follows. If the graph is directed:
  \( a(i, j) = +1 \) if node i is the tail of arc \( j \)
  \( a(i, j) = -1 \) if node i is the head of arc \( j \)
  If the graph is undirected:
  \( a(i, j) = 1 \) if node i is the tail or the head of arc \( j \)
  \( a(i, j) = 0 \) if there is no arc from node i to node j

EXAMPLE:

  g=load_graph(SCI+'/demos/metanet/colored');
a=graph_2_mat(g)
a=graph_2_mat(g,'node-node')

SEE ALSO: mat_2_graph 566

2.9.26  graph_center  center of a graph

CALLING SEQUENCE:

[no,rad] = graph_center(g)

PARAMETERS:

  g : graph list
  no : integer
  rad : integer
**DESCRIPTION:**
graph_center computes the center of the graph $g$ i.e. the node for which the largest of the shortest paths to all the other nodes is minimum. The lengths of the arcs are supposed to be integer (and the default value is 1). The output is the value $rad$ of the length of the radius and $no$ which is the node number of the center of the graph.

**EXAMPLE:**

```scilab
ta=[1 1 2 2 3 4 5 5 7 8 8 9 10 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13 9 14 11 16 1 17 14 15];
g=make_graph('foo',0,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
[no,rad] = graph_center(g)
show_nodes(no);
```

**SEE ALSO:** graph_diameter 560

### 2.9.27 graph_complement ............................ complement of a graph

**CALLING SEQUENCE:**

```scilab
g1 = graph_complement(g,[gmax])
```

**PARAMETERS:**

- $g$ : graph list
- $gmax$ : graph list
- $g1$ : graph list of the new graph

**DESCRIPTION:**

graph_complement returns the undirected graph $g1$ which is the complement of the graph $g$ with respect to the corresponding complete graph. When $gmax$ is given, the complement is made with respect to $gmax$. $g$ and $gmax$ are supposed to be simple graphs (use graph_simp before calling graph_complement if necessary) and to have the same number of nodes.

**EXAMPLE:**

```scilab
ta=[1 1 2 2 3 4 5 5 7 8 8 9 10 10 10 10 11 12 13 13 14 15 17 17 16 16];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13 9 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
g1=graph_complement(g);
show_graph(g1,'new');
g=graph_complement(g1);
show_graph(g);
```

**SEE ALSO:** graph_sum 561, graph_simp 561
2.9.28  **graph_diameter**  

**DESCRIPTION:**

`graph_diameter` computes the diameter of the graph `g` i.e. the largest shortest path between two nodes. The length of the arcs are supposed to be integer (and the default value is 1). The output is the value `d` of the length of the diameter and `p` is the corresponding path.

**EXAMPLE:**

```scilab
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13 9 14 11 16 1 17 14 15];
g=make_graph('foo',0,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
[d,p] = graph_diameter(g);
show_arcs(p);
```

**SEE ALSO:**  `graph_center` 558

2.9.29  **graph_power**  

**DESCRIPTION:**

`graph_power` computes the directed graph `g1` which is the kth power of directed 1-graph `g`. There is an arc between two nodes in `g1` if there exists a path between these nodes of length at most k in `g`. `graph_power(g,1)` is graph `g`.

If such a graph does not exist, an empty vector is returned.

**EXAMPLE:**

```scilab
ta=[1 1 2 4 4 5 6 7 2 3 5 1];
he=[2 6 3 6 7 8 8 8 4 7 3 5];
g=make_graph('foo',1,8,ta,he);
g('node_x')=[285 284 335 160 405 189 118 45];
g('node_y')=[266 179 83 176 368 252 64 309];
show_graph(g);
g1=graph_power(g,2);
show_graph(g1,'new');
```

Scilab Group  September 1996  560
2.9.30 `graph_simp` converts a graph to a simple undirected graph

**CALLING SEQUENCE:**

```plaintext
g1 = graph_simp(g)
```

**PARAMETERS:**

- `g` : graph list of the old graph
- `g1` : graph list of the new graph

**DESCRIPTION:**

`graph_simp` converts a graph to a simple undirected graph. It deletes loops in `g`, replaces directed edges with undirected edges and replaces multiple edges with single edges.

**EXAMPLE:**

```plaintext
ta=[1 1 2 2 2 3 4 4 4 5 5 5 6 7 8 8 9 9 10 10 10 10 10 10 11 12 12 13 13 13 14 14 15 16 16 17 17];
he=[1 2 10 3 5 7 4 2 9 9 4 6 6 6 6 8 2 6 9 7 4 7 11 13 13 13 15 12 11 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 98 164 162 273 235 267 384 504 493 409 573 601 627 642];
g('node_y')=[59 133 223 311 227 299 221 288 384 141 209 299 398 383 187 121 301];
show_graph(g);
g1=graph_simp(g);
show_graph(g1,'new');
```

2.9.31 `graph_sum` sum of two graphs

**CALLING SEQUENCE:**

```plaintext
g2 = graph_sum(g,g1)
```

**PARAMETERS:**

- `g` : graph list
- `g1` : graph list
- `g2` : graph list of the new graph

**DESCRIPTION:**

`graph_sum` creates a graph with an adjacency matrix equal to the sum of the adjacency matrices of the two graphs `g` and `g1`. `g` and `g1` are supposed to be simple graphs (use `graph_simp` before calling `graph_complement` if necessary) and to have the same number of nodes.

**EXAMPLE:**

```plaintext
ta=[1 1 2 2 2 3 4 4 5 5 7 8 8 9 9 10 10 10 10 10 11 12 13 13 13 14 14 15 16 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13 19 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 98 164 162 273 235 271 339 384 504 493 409 573 601 627 642];
g('node_y')=[59 133 223 311 227 299 221 288 384 141 209 299 398 383 187 121 301];
```

Scilab Group September 1996 561
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('edge_width')=ones(1,(edge_number(g)));
g('node_diam')=[1:(g('node_number'))]+20;
g('node_name')=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q'];
show_graph(g);

2.9.32 graph_union ------------------------------- union of two graphs

CALLING SEQUENCE:

g2 = graph_union(g,g1)

PARAMETERS:

- g : graph list
- g1 : graph list
- g2 : graph list of the new graph

DESCRIPTION:

graph_union creates a new graph $g_2$. The node set of $g_2$ is the union (in the usual sense) of the node sets of $g$ and $g_1$. $g_2$ has an edge for each edge of $g$ and an edge for each edge of $g_1$. The edges of $g$ and $g_1$ having the same endpoints are kept and in this case $g_2$ has multiple edges.

EXAMPLE:

ta=[1 1 2 2 2 3 4 5 5 5 7 8 8 8 9 10 10 10 10 11 11 12 13 13 13 14 14 15 16 16 17 17];
he=[2 1 0 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 12 13 9 10 14 11 16 6 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=[1:(g('node_number'))]+20;
g('node_name')=['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q'];

SEE ALSO: graph_complement 559, graph_union 562
show_graph(g);
l=netwindows(); nw=l(2);
v=[7 8 9 10 11 12 13];
show_nodes(v);
g1=subgraph(v,'nodes',g);
show_graph(g1,'new');
v=[1 2 5 6 7 8 9 10];
netwindow(nw);
show_nodes(v);
g2=subgraph(v,'nodes',g);
show_graph(g2,'new');
g=graph_union(g1,g2);
show_graph(g,'new');

SEE ALSO: supernode 592, subgraph 590

2.9.33  hamilton _______________________________ hamiltonian circuit of a graph

CALLING SEQUENCE:

cir = hamilton(g)

PARAMETERS:

g : graph list
cir : integer row vector

DESCRIPTION:
hamilton finds an hamiltonian circuit (if it exists) of the directed graph g.

EXAMPLE:

ta=[2 1 3 2 2 4 4 5 6 7 8 8 9 10 10 10 10 11 12 13 14 15 16 16 17 17];
he=[1 10 2 5 7 3 2 4 5 8 6 9 7 7 11 13 15 12 13 9 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
cir=hamilton(g);
show_arcs(cir);

2.9.34  is_connex _______________________________ connectivity test

CALLING SEQUENCE:

res = is_connex(g)

PARAMETERS:

g : graph list
res : integer, result of the test
DESCRIPTION:
\texttt{is\_connex} returns 1 if the graph \texttt{g} is connected and 0 otherwise.

EXAMPLE:
\begin{verbatim}
g=make_graph('foo',1,3,[1,2,3,1],[2,3,1,3]);
\texttt{is\_connex}(g)
g=make_graph('foo',1,4,[1,2,3,1],[2,3,1,3]);
\texttt{is\_connex}(g)
\end{verbatim}

SEE ALSO: con\_nodes 548, strong\_connex 590

2.9.35 knapsack ________________ solves a 0-1 multiple knapsack problem

CALLING SEQUENCE:
\begin{verbatim}
[earn,ind] = knapsack(profit,weight,capa,[bck])
\end{verbatim}

PARAMETERS:
\begin{itemize}
\item profit : integer row vector
\item weight : integer row vector
\item capa : integer row vector
\item bck : integer
\item earn : integer
\item ind : integer row vector
\end{itemize}

DESCRIPTION:
\texttt{knapsack} solve a 0-1 multiple knapsack problem with \( n \) (\( n \geq 2 \)) items and \( m \) knapsacks (\( m \geq 1 \)). \texttt{profit} is the vector of the (integer) profits of the \( n \) items and \texttt{weight} is the vector of the corresponding (integer) weights. \texttt{capa} is the vector of the (integer) capacities of the \( m \) knapsacks. \texttt{bck} is an optional integer: the maximum number of backtrackings to be performed, if heuristic solution is required. If the exact solution is required \texttt{bck} can be omitted or can have a negative value. \texttt{earn} is the value of the criterion for the "optimal" solution and \texttt{ind} is a vector giving the optimal location: \texttt{ind}(i) gives the number of the knapsack where item \( i \) is inserted and this value is 0 if the item \( i \) is not in the optimal solution.

We recall that the problem to be solved is the following:

\[
p(i) \text{ and } w \text{ denote respectively the profit and the weight of the item } i \text{ } i=1,...,n; \text{ } c(j) \text{ denotes the capacity of the knapsack } j \text{ } j=1,...,m; \text{ } q(j,i) \text{ denotes the quantity of item } i \text{ in knapsack } j \text{ (in fact 0 or 1).}
\]

We want to maximize the global profit \( E \):
\[
E = p(1)*\left[x(1,1)+...+x(m,1)\right]+...+p(n)*\left[x(1,n)+...+x(m,n)\right]
\]
under the constraints:
\[
[w(1)*x(j,1)+...+w(n)*x(j,m)] <= c(j) \text{ } j=1,...,m \text{ } \left[x(1,i)+...+x(m,i)\right] \leq 1 \text{ } i=1,...,n \text{ } x(j,i)= 0 \text{ or } 1 \text{ } p(),w(),c() \text{ are positive integers.}
\]

EXAMPLE:
\begin{verbatim}
weight=ones(1,15).*[1:4];
profit=ones(1,60);
capa=[15 45 30 60];
[earn,ind]=knapsack(profit,weight,capa)
\end{verbatim}

SEE ALSO: qassign 584

2.9.36 line\_graph ________________ graph with nodes corresponding to edges

CALLING SEQUENCE:
\[ g1 = \text{line\_graph}(g) \]

**PARAMETERS:**

- \( g \) : graph list of the old graph
- \( g1 \) : graph list of the new graph

**DESCRIPTION:**

\text{line\_graph} returns the graph \( g1 \) with the nodes corresponding to the edges of the graph \( g \). \( g1 \) is defined in the following way:

- Its nodes correspond to the edges of \( g \).
- Two nodes of the new graph are adjacent if and only if the corresponding edges of the graph \( g \) are adjacent.

The coordinates of the nodes of \( g1 \) are given by the middle points of the corresponding edges of \( g \).

**EXAMPLE:**

```matlab
ta=[1 1 2 4 4 5 6 7 2 3 5 1];
he=[2 6 3 6 7 8 8 4 7 3 5 1];
g=make_graph('foo',0,8,ta,he);
g('node_x')=[281 284 360 185 405 182 118 45];
g('node_y')=[262 179 130 154 368 248 64 309];
show_graph(g);
g1=line_graph(g);
show_graph(g1,'new');
```

**SEE ALSO:** arc\_graph 543

---

2.9.37 \text{load\_graph} \hfill \text{loads a graph}

**CALLING SEQUENCE:**

```matlab
g = \text{load\_graph}(\text{name})
```

**PARAMETERS:**

- name : string, the path of the graph to load
- g : graph list

**DESCRIPTION:**

name is the name of a graph file which contains the ASCII description of a graph. Such a file must have the "graph" extension. name can be the name or the pathname of the file; if the "graph" extension is missing in name, it is assumed. load\_graph returns the corresponding graph list.

**EXAMPLE:**

```matlab
g=load_graph(SCI+'/demos/metanet/mesh100.graph');
show_graph(g);
g=load_graph(SCI+'/demos/metanet/colored');
show_graph(g,'new');
```

**SEE ALSO:** save\_graph 585
2.9.38  make_graph  makes a graph list

CALLING SEQUENCE :

\[ g = \text{make\_graph}(\text{name}, \text{directed}, n, \text{tail}, \text{head}) \]

PARAMETERS :

- name : string, the name of the graph
- directed : integer, 0 (undirected graph) or 1 (directed graph)
- n : integer, the number of nodes of the graph
- tail : row vector of the numbers of the tail nodes of the graph (its size is the number of edges of the graph)
- head : row vector of the numbers of the head nodes of the graph (its size is the number of edges of the graph)
- g : graph list

DESCRIPTION :

make_graph makes a graph list according to its arguments which are respectively the name of the graph, a flag for directed or undirected, the number of nodes and the row vectors tail and head. These are the minimal data needed for a graph.

If \( n \) is a positive number, graph \( g \) has \( n \) nodes; this number must be greater than or equal to \( \max(\max(\text{tail}), \max(\text{head})) \). If it is greater than this number, graph \( g \) has isolated nodes. The nodes names are taken as the nodes numbers.

If \( n \) is equal to 0, graph \( g \) has no isolated node and the number of nodes is computed from \( \text{tail} \) and \( \text{head} \). The nodes names are taken from the numbers in \( \text{tail} \) and \( \text{head} \).

EXAMPLE :

// creating a directed graph with 3 nodes and 4 arcs.
g=make_graph('foo',1,3,[1,2,3,1],[2,3,1,3]);

// creating a directed graph with 13 nodes and 14 arcs.
ta=[1 1 2 7 8 9 10 10 10 11 12 13 13];
he=[2 10 7 8 9 7 7 11 13 13 12 13 9 10];
g=make_graph('foo',1,13,ta,he);
g('node_x')=[120 98 87 188 439 698 226 127 342 467 711 779 477];
g('node_y')=[21 184 308 426 435 428 129 360 435 55 109 320 321];
show_graph(g)

// creating same graph without isolated node and 14 arcs.
g=make_graph('foo',1,0,ta,he);
g('node_x')=[120 98 226 127 342 467 711 779 477];
g('node_y')=[21 184 308 426 435 55 109 320 321];
show_graph(g,'new')

SEE ALSO:  graph-list 555

2.9.39  mat_2_graph  graph from node-arc or node-node incidence matrix

CALLING SEQUENCE :

\[ g = \text{mat\_2\_graph}(a, \text{directed}, [\text{mat}]) \]

PARAMETERS :

- a : sparse node-arc or node-node incidence matrix
Directed : integer, 0 (undirected graph) or 1 (directed graph)
Mat : optional string, 'node-arc' or 'node-node' matrix
G : graph list

DESCRIPTION :
Mat_2_graph computes the graph g corresponding to the node-arc or the node-node incidence matrix a. Note that a checking is made to insure that a is a sparse node-arc or node-node incidence matrix of a directed (directed = 1) or undirected (directed = 0) graph. If the optional argument mat is omitted or is the string 'node-arc', a must be a node-arc matrix. If mat is the string 'node-node', a must be a node-node matrix.

EXAMPLE :

g=load_graph(SCI+’/demos/metanet/colored’);
show_graph(g);
a=graph_2_mat(g);
g1=mat_2_graph(a,1);
g1(’node_x’)=g(’node_x’); g1(’node_y’)=g(’node_y’);
show_graph(g1,’new’);
a=graph_2_mat(g,’node-node’);
g1=mat_2_graph(a,1,’node-node’);
g1(’node_x’)=g(’node_x’); g1(’node_y’)=g(’node_y’);
show_graph(g1,’new’);

SEE ALSO:  adj_lists 542, chain_struct 546, graph_2_mat 558

2.9.40  max_cap_path __________________________ maximum capacity path

CALLING SEQUENCE :

[p, cap] = max_cap_path(i, j, g)

PARAMETERS :

i, j : integers, node numbers
G : graph list
P : row vector of integer numbers of the arcs of the path if it exists
Cap : value of the capacity of the path

DESCRIPTION :
max_cap_path returns the path with maximum capacity from node i to node j for the graph g if it exists and returns the empty vector [] otherwise.

The capacities of the edges are given by the element edge_max_cap of the graph list. If its value is not given (empty vector []), max_cap_path returns the empty vector []. The capacities must be strictly positive, i.e negative capacities are considered as equal to 0 (no capacity at all).

EXAMPLE :

ta=[112223455788910101112131314141516161717];
he=[210357424686977111512139101411161171415];
g=make_graph(’foo’,i,17,ta,he);
g(’node_x’)=[2831636357164164273271339384504513439623631757642];
g(’node_y’)=[59133223318227319221324432432141209319428443187151301];
show_graph(g);
ma=edge_number(g);
max_flow Scilab function

max_flow('edge_max_cap')=int(rand(1,ma)*16)+5;
[p,cap]=max_cap_path(1,14,g);
edgcolor=1*ones(1,ma); edgcolor(p)=11*ones(p); g('edge_color')=edgcolor;
x_message(['The maximum capacity is: '+string(cap);
    'Showing the corresponding path']);
show_graph(g); show_arcs(p);

2.9.41 max_clique ____________________________ maximum clique of a graph

CALLING SEQUENCE :

[size,nodes] = max_clique(g,[ind])

PARAMETERS :

  g : graph list
  ind : integer (optional)
  size : integer
  nodes : integer row vector

DESCRIPTION :

max_clique computes the maximum clique of the graph g i.e. the complete subgraph of maximum size. ind is a parameter for the choice of the method: if ind is 0 the method is a partial enumerative algorithm and if ind is 1 the algorithm is based on quadratic zero-one programming. The default is 0. The output size is the number of the nodes of the clique found by the algorithm and nodes is the vector of the corresponding nodes.

EXAMPLE :

ta=[1 2 3 4 5 6 7 8 9 10 16 16 10 11 12 12 11 14 15 15 13 7 13 13];
he=[2 3 4 5 6 7 8 8 9 10 16 2 3 11 12 13 11 14 14 15 5 9 12 4 14 15];
g=make_graph('foo',0,16,ta,he);
g('node_x')=[106 199 369 467 470 403 399 347 308 269 184 108 199 268 345 272];
g('node_y')=[341 420 422 321 180 212 286 246 193 244 243 209 59 134 51 348];
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
[ns,no] = max_clique(g);
show_nodes(no);
g1=graph_complement(g);
[ns,no] = max_clique(g1);
show_nodes(no);

2.9.42 max_flow ____________________________ maximum flow between two nodes

CALLING SEQUENCE :

[v,phi,flag] = max_flow(i,j,g)

PARAMETERS :

  i : integer, number of start node
  j : integer, number of end node
**mesh2d**

**Scilab function**

---

**g :** graph list  
**v :** value of the maximum flow it is exists  
**phi :** row vector of the value of the flow on the arcs  
**flag :** feasible problem flag (0 or 1)

**DESCRIPTION :**

`max_flow` returns the value of maximum flow \( v \) from node number \( i \) to node number \( j \) if it exists, and the value of the flow on each arc as a row vector \( \phi \). All the computations are made with integer numbers. The graph must be directed. If the problem is not feasible, \( \text{flag} \) is equal to 0, otherwise it is equal to 1.

The bounds of the flow are given by the elements `edge_min_cap` and `edge_max_cap` of the graph list. The value of the maximum capacity must be greater than or equal to the value of the minimum capacity. If the value of `edge_min_cap` or `edge_max_cap` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

**EXAMPLE :**

```plaintext
ta=[1 1 2 2 3 3 4 4 5 5 5 5 6 6 6 7 7 15 15 15 15 15 15];
ta=[ta, 15 8 9 10 11 12 13 14];
he=[10 13 9 14 8 11 9 11 8 10 12 13 8 9 12 8 11 1 2 3 4];
he=[he, 5 6 7 16 16 16 16 16 16 16];
n=16;
g=make_graph('foo',1,n,ta,he);
g('node_x')=[42 615 231 505 145 312 403 233 506 34 400 312 142 614 260 257];
g('node_y')=[143 145 154 154 147 152 157 270 273 279 269 273 273 274 50 376];
ma=edge_number(g);
g('edge_max_cap')=ones(1,ma);
g('edge_min_cap')=zeros(1,ma);
source=15; sink=16;
nodetype=0*ones(1,n); nodetype(source)=2; nodetype(sink)=1;
g('node_type')=nodetype;
nodecolor=0*ones(1,n); nodecolor(source)=11; nodecolor(sink)=11;
g('node_color')=nodecolor;
show_graph(g);
[v,phi,ierr]=max_flow(source,sink,g);
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
g('edge_font_size')=edgefontsize;
g('edge_label')=string(phi);
show_graph(g);
```

---

**2.9.43 mesh2d  ___________________________  triangulation of n points in the plane**

**CALLING SEQUENCE :**

```plaintext
[nutr,A] = mesh2d(x,y,[front])
```

**PARAMETERS :**

- **x :** real row array  
- **y :** real row array  
- **front :** integer row array  
- **nutr :** integer matrix  
- **A :** sparse 0-1 matrix

---

Scilab Group  
September 1996  
569
DESCRIPTION:
The arrays x and y are the coordinates of n points in the plane. `mesh2d` returns a matrix `nutr(3,nbt)` of the numbers of the nodes of the nbt triangles of the triangulation of the points. It returns also a sparse matrix A representing the connections between the nodes \( A(i,j)=1 \) if \((i,j)\) is a side of one of the triangles or \(i=j\). In the case of 3 parameters `front` is the array defining the boundary: it is the array of the indices of the points located on the boundary. The boundary is defined such that the normal to the boundary is oriented towards outside. The boundary is given by its connected components: a component is the part \((i1,i2)\) such that `front(i1)=front(i2)` (the external boundary is defined in the counterclockwise way, see the examples below). The error cases are the following: err = 0 if no errors were encountered; err = 3 all nodes are collinear.

If the boundary is given, the other error cases are: err = 2 some points are identical; err = 5 wrong boundary array; err = 6 crossed boundary; err = 7 wrong orientation of the boundary; err = 10 an interior point is on the boundary; err = 8 size limitation; err = 9 crossed boundary; err = 12 some points are identical or size limitation.

EXAMPLE:

```scilab
// FIRST CASE
theta=0.025*[1:40]*2.*%pi;
x=1+cos(theta);
y=1.+sin(theta);
theta=0.05*[1:20]*2.*%pi;
x1=1.3+0.4*cos(theta);
y1=1.+0.4*sin(theta);
theta=0.1*[1:10]*2.*%pi;
x2=0.5+0.2*cos(theta);
y2=1.+0.2*sin(theta);
x=[x x1 x2];
y=[y y1 y2];
//
u=mesh2d(x,y);
nbt=size(nu,2);
jj=[nu(1,:)' nu(2,:)' nu(3,:)' nu(3,:)' nu(1,:)'];
as=sparse(jj,ones(size(jj,1),1));
ast=tril(as+abs(as'-as));
[jj,v,mn]=spget(ast);
n=size(x,2);
g=make_graph('foo',0,n,jj(:,1)',jj(:,2)');
g('node_x')=300*x;
g('node_y')=300*y;
g('default_node_diam')=10;
show_graph(g)

// SECOND CASE !!! NEEDS x,y FROM FIRST CASE
x3=2.*rand(1:200);
y3=2.*rand(1:200);
wa=((x3-1).*1+(y3-1).*1);
i1=find(wa >= .94);
x3(i1)=[];
y3(i1)=[];
wa=((x3-0.5).*1+(y3-1).*1);
i2=find(wa <= 0.055);
x3(ii)=[];
y3(ii)=[];
wa=((x3-1.3).*1+(y3-1).*1);
i3=find(wa <= 0.21);
x3(ii)=[];
y3(ii)=[];
xnew=[x x3];ynew=[y y3];
fr1=[[1:40] 1];fr2=[[41:60] 41];fr2=fr2(1:1);
```

Scilab Group September 1996
fr3=[[61:70] 61]; fr3=fr3(:,1:1);
front=[fr1 fr2 fr3];
//
nu=mesh2d(xnew,ynew,front);
nbt=size(nu,2);
jj=[nu(1,:)’ nu(2,:)’ nu(3,:)’ nu(3,:)’ nu(1,:)’];
as=sparse(jj,ones(size(jj,1),1));
ast=tril(as+abs(as’-as));
[jj,v,mn]=spget(ast);
n=size(xnew,2);
g=make_graph(’foo’,0,n,jj(:,1)’,jj(:,2)’);
g(’node_x’)=300*xnew;
g(’node_y’)=300*ynew;
g(’default_node_diam’) = 10;
show_graph(g)
// REGULAR CASE !!! NEEDS PREVIOUS CASES FOR x,y,front
xx=0.1*[1:20];
yy=xx.*ones(1,20);
zz=ones(1,20).*xx;
x3=yy;y3=zz;
wa=((x3-1.3).* (x3-1)+(y3-1).* (y3-1));
ii=find(wa >= .94);
x3(ii)=[];y3(ii)=[];
wa=((x3-0.5).* (x3-0.5)+(y3-1).* (y3-1));
ii=find(wa <= 0.055);
x3(ii)=[];y3(ii)=[];
wa=((x3-1.3).* (x3-1.3)+(y3-1).* (y3-1));
ii=find(wa <= 0.21);
x3(ii)=[];y3(ii)=[];
xnew=[x x3];ynew=[y y3];
u=mesh2d(xnew,ynew,front);
nbt=size(nu,2);
jj=[nu(1,:)’ nu(2,:)’ nu(3,:)’ nu(3,:)’ nu(1,:)’];
as=sparse(jj,ones(size(jj,1),1));
ast=tril(as+abs(as’-as));
[jj,v,mn]=spget(ast);
n=size(xnew,2);
g=make_graph(’foo’,0,n,jj(:,1)’,jj(:,2)’);
g(’node_x’)=300*xnew;
g(’node_y’)=300*ynew;
g(’default_node_diam’) = 3;
show_graph(g)

2.9.44 metanet __________________________ opens a Metanet window

CALLING SEQUENCE:

window = metanet([path,winsize])

PARAMETERS:

path : string, directory where graph files are searched
winsize : row vector defining the size of Metanet window
window : integer, window number
**DESCRIPTION:**
This function is used to open a Metanet window from Scilab.

path is an optional argument; it is the directory where graph files are searched. If this path is the pathname of a graph, this graph is displayed in the Metanet window and the directory of this pathname becomes the default directory. By default, path is the working directory.

winsize is an optional argument; it is a row vector [width height] giving the size in pixels of Metanet window. The default is [1000 1000].

Usually, show_graph is used and metanet is seldom used.

Each time metanet is executed, a new window is created and its number is incremented by 1.

**SEE ALSO:** netclose 578, netwindow 578, netwindows 579, show_graph 587

### 2.9.45 metanet_sync __________ asynchronous or synchronous mode in Metanet

**CALLING SEQUENCE:**
es = metanet_sync([flag])

**PARAMETERS:**

res : integer
flag : integer

**DESCRIPTION:**
By default Metanet windows work with Scilab in asynchronous mode, i.e., Scilab proceeds without waiting for graphics commands sent to Metanet window to terminate: these commands are show_graph, show_arcs and show_nodes. This mode is the most efficient. But when running a lot of such graphical commands, problems can arise.

metanet_sync(0) changes to asynchronous mode (default).
metanet_sync(1) changes to synchronous mode.
metanet_sync() returns the current mode (0 = asynchronous, 1 = synchronous).

### 2.9.46 min_lcost_cflow __________ minimum linear cost constrained flow

**CALLING SEQUENCE:**

[c,phi,v,flag] = min_lcost_cflow(i,j,cv,g)

**PARAMETERS:**

i : integer, source node number
j : integer, sink node number
cv : scalar, value of constrained flow
g : graph list
c : value of cost
phi : row vector of the values of flow on the arcs
v : value of flow from source to sink
flag : feasible constrained flow flag (0 or 1)

**DESCRIPTION:**
min_lcost_cflow computes the minimum cost flow in the network g, with the value of the flow from source node i to sink node j constrained to be equal to cv.

min_lcost_cflow returns the total cost of the flows on the arcs c, the row vector of the flows on the arcs phi and the value of the flow v on the virtual arc from sink to source. If v is less than cv, a message is issued, but the computation is done: in this case flag is equal to 0, otherwise it is equal to 1.
The bounds of the flows are given by the elements \texttt{edge_min_cap} and \texttt{edge_max_cap} of the graph list. The value of the minimum capacity must be equal to zero, and the value of the maximum capacity must be non-negative and must be integer numbers. If the value of \texttt{edge_min_cap} or \texttt{edge_max_cap} is not given (empty row vector \texttt{[]}), it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the element \texttt{edge_cost} of the graph list. The costs must be non-negative. If the value of \texttt{edge_cost} is not given (empty row vector \texttt{[]}), it is assumed to be equal to 0 on each edge.

The demands, element \texttt{node_demand} of the graph list, must be equal to zero.

This function uses the algorithm of Busacker and Goven.

**EXAMPLE :**

```matlab
ta=[1 1 2 2 3 4 4 5 6 6 7 7 7 8 10 10 12 12 13 13 13 14 15 14 9 11 10];
he=[2 6 3 4 5 1 3 5 1 7 10 11 11 5 8 9 5 8 11 10 11 9 11 15 13 14 4 6 9 1];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 181 276 278 276 103 174 281 177 86 175 90 290 397 399];
show_graph(g);

g1=g; ma=arc_number(g1); n=g1('node_number');
g1('edge_min_cap')=0*ones(1,ma);
rand('uniform');
g1('edge_max_cap')=round(20*rand(1,ma))+ones(1,ma);
g1('edge_cost')=10*rand(1,ma)+ones(1,ma);
source=15; sink=1; cv=5;
[c,phi,v]=min_lcost_cflow(source,sink,cv,g1);
x_message(['The cost is: '+string(c);
    'Showing the flow on the arcs']);
    nodetype=0*ones(1,n); nodetype(source)=2; nodetype(sink)=1;
g1('node_type')=nodetype;
i=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g1('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
nodecolor=0*ones(1,n); nodecolor(source)=11; nodecolor(sink)=11;
g1('node_color')=nodecolor;
g1('edge_font_size')=edgefontsize;
g1('edge_label')=string(phi);
show_graph(g1);
```

**SEE ALSO :** min_lcost_flow1 573, min_lcost_flow2 574, min_qcost_flow 576

### 2.9.47 min_lcost_flow1 573

**minimum linear cost flow**

**CALLING SEQUENCE :**

```matlab
[c,phi,flag] = min_lcost_flow1(g)
```

**PARAMETERS :**

- **g** : graph list
- **c** : value of cost
- **phi** : row vector of the value of flow on the arcs
- **flag** : feasible problem flag (0 or 1)

**DESCRIPTION :**

Scilab Group

September 1995

573
min_lcost_flow1 computes the minimum linear cost flow in the network \( g \). It returns the total cost of the flows on the arcs \( c \) and the row vector of the flows on the arcs \( \phi \). If the problem is not feasible (impossible to find a compatible flow for instance), \( \text{flag} \) is equal to 0, otherwise it is equal to 1.

The bounds of the flow are given by the elements \( \text{edge min cap} \) and \( \text{edge max cap} \) of the graph list. The value of the minimum capacity and of the maximum capacity must be non negative and must be integer numbers. The value of the maximum capacity must be greater than or equal to the value of the minimum capacity. If the value of \( \text{edge min cap} \) or \( \text{edge max cap} \) is not given (empty row vector \([\,]\)), it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the element \( \text{edge cost} \) of the graph list. The costs must be non negative. If the value of \( \text{edge cost} \) is not given (empty row vector \([\,]\)), it is assumed to be equal to 0 on each edge.

The demands, element \( \text{node demand} \) of the graph list, must be equal to zero.

This function uses the out-of-kilter algorithm.

**EXAMPLE :**

ta=[1 1 2 2 3 4 4 5 6 6 7 7 8 8 9 10 12 12 13 13 14 15 14 9 11 10 1 8];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14 4 6 9 1 12 14];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 221 316 318 316 143 214 321 217 126 215 80 330 437 439];
show_graph(g);
g1=g;ma=arc_number(g1);
rand('uniform');
while %T then
    g1('edge_min_cap')=round(20*rand(1,ma));
    g1('edge_max_cap')=round(20*rand(1,ma))+g1('edge_min_cap')+33*ones(1,ma);
    g1('edge_cost')=round(10*rand(1,ma))+ones(1,ma);
    [c,phi,flag]=min_lcost_flow1(g1);
    if flag==1 then break; end;
end;
x_message(['The cost is: '+string(c);
    'Showing the flow on the arcs ']);
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g1('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
g1('edge_font_size')=edgefontsize;
g1('edge_label')=string(phi);
show_graph(g1);

**SEE ALSO :** min_lcost_cflow 572, min_lcost_flow2 574, min_qcost_flow 576

### 2.9.48 min_lcost_flow2

**minimum linear cost flow**

**CALLING SEQUENCE :**

\[
[c, \phi, \text{flag}] = \text{min_lcost_flow2}(g)
\]

**PARAMETERS :**

- \( g \) : graph list
- \( c \) : value of cost
- \( \phi \) : row vector of the value of flow on the arcs
flag : feasible problem flag (0 or 1)

**DESCRIPTION:**

`min_lcost_flow2` computes the minimum linear cost flow in the network `g`. It returns the total cost of the flows on the arcs `c` and the row vector of the flows on the arcs `phi`. If the problem is not feasible (impossible to find a compatible flow for instance), `flag` is equal to 0, otherwise it is equal to 1.

The bounds of the flow are given by the elements `edge_min_cap` and `edge_max_cap` of the graph list. The value of the minimum capacity must be equal to zero. The values of the maximum capacity must be non negative and must be integer numbers. If the value of `edge_min_cap` or `edge_max_cap` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the element `edge_cost` of the graph list. The costs must be non negative and must be integer numbers. If the value of `edge_cost` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each edge.

The demand on the nodes are given by the element `node_demand` of the graph list. The demands must be integer numbers. Note that the sum of the demands must be equal to zero for the problem to be feasible. If the value of `node_demand` is not given (empty row vector `[]`), it is assumed to be equal to 0 on each node.

This functions uses a relaxation algorithm due to D. Bertsekas.

**EXAMPLE:**

```scilab
ta=[1 1 2 2 2 3 4 5 6 6 7 7 8 8 9 9 10 10 11 12 12 13 13 14 14 15 14 9 11 10 1 8];
he=[2 6 3 4 5 1 3 5 1 7 10 11 11 10 11 9 11 15 13 14 14 4 6 9 1 12 14];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 221 316 318 316 143 214 321 217 126 215 80 330 437 439];
show_graph(g);
g1=g; ma=arc_number(g1); n=g1('node_number');
g1('edge_min_cap')=0.*ones(1,ma);
x_message(['Random generation of data';
    'The first(s) generated problem(s) may be unfeasible']);
while %T then
    rand('uniform');
    g1('edge_max_cap')=round(20*rand(1,ma))+20*ones(1,ma);
    g1('edge_cost')=round(10*rand(1,ma)+ones(1,ma));
    rand('normal');
    dd=20.*rand(1,n)-10*ones(1,n);
    dd=round(dd-sum(dd)/n*ones(1,n));
    dd(n)=dd(n)-sum(dd);
    g1('node_demand')=dd;
    [c,phi,flag]=min_lcost_flow2(g1);
    if flag==1 then break; end;
end;
x_message(['The cost is: '+string(c);
    'Showing the flow on the arcs and the demand on the nodes']);
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g1('edge_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
g1('edge_font_size')=edgefontsize;
g1('edge_label')=string(phi);
g1('node_label')=string(g1('node_demand'));
show_graph(g1);
```

**SEE ALSO:** `min_lcost_cflow 572`, `min_lcost_flow1 573`, `min_qcost_flow 576`
2.9.49 \texttt{min\_qcost\_flow} \hspace{1cm} \text{minimum quadratic cost flow}

**CALLING SEQUENCE:**

\[
[c, \phi, \text{flag}] = \text{min\_qcost\_flow}(\epsilon, g)
\]

**PARAMETERS:**

- \(\epsilon\): scalar, precision
- \(g\): graph list
- \(c\): value of cost
- \(\phi\): row vector of the value of flow on the arcs
- \(\text{flag}\): feasible problem flag (0 or 1)

**DESCRIPTION:**

\texttt{min\_qcost\_flow} computes the minimum quadratic cost flow in the network \(g\). It returns the total cost of the flows on the arcs \(c\) and the row vector of the flows on the arcs \(\phi\). \(\epsilon\) is the precision of the iterative algorithm. If the problem is not feasible (impossible to find a compatible flow for instance), \(\text{flag}\) is equal to 0, otherwise it is equal to 1.

The bounds of the flow are given by the elements \(\text{edge\_min\_cap}\) and \(\text{edge\_max\_cap}\) of the graph list. The value of the maximum capacity must be greater than or equal to the value of the minimum capacity. If the value of \(\text{edge\_min\_cap}\) or \(\text{edge\_max\_cap}\) is not given (empty row vector []), it is assumed to be equal to 0 on each edge.

The costs on the edges are given by the elements \(\text{edge\_q\_orig}\) and \(\text{edge\_q\_weight}\) of the graph list. The cost on arc \(u\) is given by:

\[
(1/2) \times \text{edge\_q\_weight}[u] (\phi[u] - \text{edge\_q\_orig}[u])^2
\]

The costs must be non-negative. If the value of \(\text{edge\_q\_orig}\) or \(\text{edge\_q\_weight}\) is not given (empty row vector []), it is assumed to be equal to 0 on each edge.

This function uses an algorithm due to M. Minoux.

**EXAMPLE:**

```plaintext
ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 12 13 13 13 14 14 14 14 14 14 9 11 10 1 8];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14 4 6 9 1 12 14];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 221 316 318 316 143 214 321 217 126 215 80 330 437 439];
show_graph(g);
g1=g; ma=arc_number(g1);
rand('uniform');
while %T then
    g1('edge\_min\_cap')=round(5*rand(1,ma));
    g1('edge\_max\_cap')=round(20*rand(1,ma))+30*ones(1,ma);
    g1('edge\_q\_orig')=0*ones(1,ma);
    g1('edge\_q\_weight')=ones(1,ma);
    [c,phi,flag]=min\_qcost\_flow(0.001,g1);
    if flag==1 then break; end;
end;
end;
x\_message(['The cost is: '+string(c);
    'Showing the flow on the arcs']);
ii=find(phi<>0); edgecolor=phi; edgecolor(ii)=11*ones(ii);
g1('edge\_color')=edgecolor;
edgefontsize=8*ones(1,ma); edgefontsize(ii)=18*ones(ii);
g1('edge\_font\_size')=edgefontsize;
```

Scilab Group September 1995  576
neighbors Scilab function

gl('edge_label')=string(phi);
show_graph(gl);

SEE ALSO:  min_lcost_cflow 572, min_lcost_flow1 573, min_lcost_flow2 574

2.9.50  min_weight_tree  __________________________ minimum weight spanning tree

CALLING SEQUENCE :

t  = min_weight_tree([i],g)

PARAMETERS :

i : integer, node number of the root of the tree

g : graph list

t : row vector of integer numbers of the arcs of the tree if it exists

DESCRIPTION :

min_weight_tree  tries to find a minimum weight spanning tree for the graph g. The optional argument

i is the number of the root node of the tree; its default value is node number 1. This node is meaningless

for an undirected graph.

The weights are given by the element edge_weight of the graph list. If its value is not given (empty

vector []), it is assumed to be equal to 0 on each edge. Weights can be positive, equal to 0 or negative.

To compute a spanning tree without dealing with weights, give to weights a value of 0 on each edge or the

empty vector [].

min_weight_tree  returns the tree t as a row vector of the arc numbers (directed graph) or edge

numbers (undirected graph) if it exists or the empty vector [] otherwise. If the tree exists, the dimension

of t is the number of nodes less 1. If t(i) is the root of the tree: - for j < i, t(j) is the number of the

arc in the tree after node t(j) - for j > i, t(j) is the number of the arc in the tree before node t(j)

EXAMPLE :

ta=[1 1 2 2 3 4 5 5 7 8 8 9 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 11 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757

642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151

301];
show_graph(g);
t=min_weight_tree(1,g);
gl=g; ma=arc_number(gl); n=gl('node_number');
nodetype=0*ones(1,n); nodetype(1)=2; gl('node_type')=nodetype;
edgecolor=1*ones(1,ma); edgewidth=t=4*ones(t); gl('edge_color')=edgecolor;
edgewidth=4*ones(1,ma); edgewidth(t)=4*ones(t); gl('edge_width')=edgewidth;
x_message('Minimum weight tree from node 1');
show_graph(gl);

2.9.51  neighbors  __________________________ nodes connected to a node

CALLING SEQUENCE :

a  = neighbors(i,g)
PARAMETERS:

i : integer
g : graph list
a : vector of integers

DESCRIPTION:
neighbors returns the numbers of the nodes connected with node i for graph g (directed or not).

EXAMPLE:

```
ta=[1 6 2 4 7 5 6 8 4 3 5 1];
he=[2 1 3 6 4 8 8 7 2 7 3 5];
g=make_graph('foo',1,8,ta,he);
g('node_x')=[285 284 335 160 405 189 118 45];
g('node_y')=[266 179 83 176 368 252 64 309];
show_graph(g);
a=neighbors(6,g)
show_nodes(a);
```

SEE ALSO: predecessors 584, successors 591
"Scilab function"

2.9.52 netclose __________________________________________ closes a Metanet window

CALLING SEQUENCE:
netclose(window)

PARAMETERS:

window : integer, window number

DESCRIPTION:
Each Metanet window has a window number returned by the metanet and show_graph functions. This function is used to close the Metanet window with number window.

SEE ALSO: metanet 571, netwindow 578, netwindows 579, show_graph 587

2.9.53 netwindow ________________________________________ chooses a Metanet window

CALLING SEQUENCE:
netwindow(window)

PARAMETERS:

window : integer, window number

DESCRIPTION:
This function is used to change the Metanet window. Each Metanet window has a window number returned by the metanet and show_graph functions. To use the Metanet window associated to window number window, use netwindow(window). The numbers of existing windows are given by the function netwindows.

SEE ALSO: metanet 571, netclose 578, netwindows 579, show_graph 587
2.9.54 netwindows gets the numbers of Metanet windows

CALLING SEQUENCE:

l = netwindows()

PARAMETERS:

l : list

DESCRIPTION:
This function returns a list l. Its first element is the row vector of all the Metanet windows and the second element is the number of the current Metanet window. This number is equal to 0 if no current Metanet window exists.

SEE ALSO: metanet, netclose, netwindow, show_graph

2.9.55 node_number number of nodes of a graph

CALLING SEQUENCE:

n = node_number(g)

PARAMETERS:

g : graph list
n : integer, number of nodes

DESCRIPTION:
node_number returns the number n of nodes of the graph.

SEE ALSO: arc_number, edge_number

2.9.56 nodes_2_path path from a set of nodes

CALLING SEQUENCE:

p = nodes_2_path(ns,g)

PARAMETERS:

ns : row vector of integer numbers of the set of nodes
g : graph list
p : row vector of integer numbers of the arcs of the path if it exists

DESCRIPTION:
nodes_2_path returns the path p corresponding to the node sequence ns given by its node numbers if it exists; it returns the empty vector [ ] otherwise.

EXAMPLE:

Scilab Group September 1995
path_2_nodes

Se also:

2.9.57 nodes_degrees __________________________ degrees of the nodes of a graph

Calling sequence:

[outdegree, indegree] = graph_degree(g)

Parameters:

- g : graph list
- outdegree : row vector of the out degrees of the nodes
- indegree : row vector of the in degrees of the nodes

Description:

nodes_degrees returns the 2 row vectors of the out and in degrees of the nodes of the graph g.

Example:

ta=[1 1 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 13 14 15 16 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);

See also: adj_lists 542

2.9.58 path_2_nodes __________________________ set of nodes from a path

Calling sequence:

ns = path_2_nodes(p,g)
PARAMETERS:

p : row vector of integer numbers of the arcs of the path

PARAMETERS :
g : graph list

DESCRIPTION:

path_2_nodes returns the set of nodes ns corresponding to the path p given by its arc numbers; if p is not a path, the empty vector [] is returned.

EXAMPLE:

ta=[1 1 2 2 3 4 5 7 8 8 9 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 10 10 7 7 7 11 11 12 12 13 13 15 15 16 16 16 16 17 17];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);
p=[2 16 23 25 26 17 18 19 13 10 11];
g1=g; edgecolor=1*ones(ta); edgecolor(p)=11*ones(p);
g1('edge_color')=edgecolor;
show_graph(g1); show_arcs(p);
ns=path_2_nodes(p,g);
g1=g; nodecolor=1*ones(g1('node_number')); nodecolor(ns)=11*ones(ns);
g1('node_color')=nodecolor;
show_graph(g1); show_nodes(ns);
show_arcs(p,'sup');

SEE ALSO: nodes_2_path 579

2.9.59 perfect_match ____________________________ min-cost perfect matching

CALLING SEQUENCE:

[cst,nmatch] = perfect_match(g,arcost)

PARAMETERS:

g : graph list
arcost : integer row vector
cst : integer
nmatch : integer row vector

DESCRIPTION:

perfect_match finds a perfect min-cost matching for the graph g. g must be an undirected graph with an even number of nodes. arcost is the vector of the (integer) costs of the arcs (the dimension of arcost is twice the number of edges of the graph). The output is the vector nmatch of the perfect matching and the corresponding cost cst.

EXAMPLE:

ta=[27 27 3 12 11 12 27 26 27 26 25 25 24 23 23 21 22 21 20 19 18 18];
ta=[27 27 3 12 11 12 27 26 27 26 25 25 24 23 23 21 22 21 20 19 18 18];
he=[1 1 2 2 4 5 11 13 1 25 22 24 22 22 19 13 13 14 16 16 9 16];
he=[1 1 2 2 4 5 11 13 1 25 22 24 22 22 19 13 13 14 16 16 9 16];

Scilab Group September 1996 581
n=28;
g=make_graph('foo',0,n,ta,he);
xx=[46 120 207 286 366 453 543 544 473 387 300 206 136 250 346 408];
g('node_x')=[xx 527 443 306 196 139 264 55 58 46 118 513];
yy=[36 34 37 40 38 40 35 102 102 98 93 96 167 172 101 179];
g('node_y')=[yy 198 252 183 148 172 256 259 258 167 109 104 253];
show_graph(g);m2=2*size(ta,2);
arcost=round(100.*rand(1,m2));
[cst,nmatch] = perfect_match(g,arcost);
sp=sparse([ta' he'],[1:size(ta,2)]',[n,n]);
sp1=sparse([1:n' nmatch']',ones(1,size(nmatch,2))',[n,n]);
[ij,v,mn]=spget(sp.*sp1);
show_arcs(v');

SEE ALSO: best_match 545

2.9.60 pipe_network _______________ solves the pipe network problem

CALLING SEQUENCE:

[x,pi] = pipe_network(g)

PARAMETERS:

- g: graph list
- x: row vector of the value of the flow on the arcs
- pi: row vector of the value of the potential on the nodes

DESCRIPTION:

pipe_network returns the value of the flows and of the potentials for the pipe network problem: flow problem with two Kirchhoff laws. The graph must be directed. The problem must be feasible (the sum of the node demands must be equal to 0). The resistances on the arcs must be strictly positive and are given as the values of the element 'edge_weight' of the graph list.

The problem is solved by using sparse matrices LU factorization.

EXAMPLE:

ta=[1 1 2 2 3 3 4 4 5 5 5 5 6 6 6 7 7 15 15 15 15 15 15];
ta=[ta, 15 8 9 10 11 12 13 14];
he=[10 13 9 14 8 11 9 11 8 10 12 13 8 9 12 8 11 1 2 3 4];
he=[he, 5 6 7 16 16 16 16 16 16 16];
n=16;
g=make_graph('foo',1,n,ta,he);
g('node_x')=[42 615 231 505 145 312 403 233 506 34 400 312 142 614 260 257];
g('node_y')=[143 145 154 154 147 152 157 270 273 279 269 273 273 274 50 376];
show_graph(g);
g('node_demand')=[0 0 0 0 0 0 0 0 0 0 0 0 -100 100];
w = [1 3 2 6 4 7 8 1 2 2 4 7 8 9 2 3 5 7 3 2 5 8 2 5 8];
g('edge_weight')=[w, 6 4 3 5 6];
[x,pi] = pipe_network(g)

2.9.61 plot_graph ___________________________ general plot of a graph

CALLING SEQUENCE:
plot_graph(g,[rep,rep1])

PARAMETERS :

  g : graph list  
  rep : row vector of 13 values for the parameters of the plot  
  rep1 : row vector of 4 values defining the plotting rectangle

DESCRIPTION :
plot_graph plots graph g in a Scilab graphical window. The optional arguments rep and rep1 define the parameters of the plot. If there are not given, a dialog box for the definition of these parameters is opened.

  rep must be a row vector with 13 integer numbers which must be 1 or 2. The meaning of the values of rep are:
  Frame definition: 1 = Automatic 2 = Given (see below)  
  Plotting arrows: 1 = yes, 2 = no  
  Plotting sink and source nodes: 1 = yes, 2 = no  
  Plotting node names: 1 = yes, 2 = no  
  Plotting node labels: 1 = yes, 2 = no  
  Plotting arc names : 1 = yes, 2 = no  
  Plotting arc labels: 1 = yes, 2 = no  
  Plotting node demand: 1 = yes, 2 = no  
  Plotting edge length: 1 = yes, 2 = no  
  Plotting edge cost: 1 = yes, 2 = no  
  Plotting edge min cap: 1 = yes, 2 = no  
  Plotting edge max cap: 1 = yes, 2 = no  
  Plotting edge weight: 1 = yes, 2 = no

If rep(1) is 2, the frame definition must be given by rep1. Otherwise, rep1 can be omitted. rep1 must be a row vector [orx,ory,w,h] giving respectively the coordinates of the upper-left point, the width and the height of the plotting rectangle.

EXAMPLE :

// simple graph with different choices for the plot
ta=[2 2 1 1 2 4 3 3 4];  
he=[2 2 3 2 3 2 1 2 1];  
g=make_graph('foo',1,4,ta,he);  
g('node_type')=[1 1 1 2]; g('node_name')=string([1:4]);  
g('node_x')=[73 737 381 391]; g('node_y')=[283 337 458 142];  
g('node_color')=[3 3 3 11];  
g('node_diam')=[30 30 30 60];  
g('edge_color')=[10 0 2 6 11 11 0 0 11];  
rep=[2 2 1 1 2 2 2 2 2 2 2 2 2];  
rep1=[100 -400 650 300];  
xbasc(); plot_graph(g,rep,rep1);  
xbasc(); plot_graph(g,rep,rep1);  
// plotting using dialogs  
x_message('plot the graph with different parameters');  
xbasc(); plot_graph(g,rep,rep1);  
xbasc(); plot_graph(g);  
xset("thickness",4);  
xset('default');  

SEE ALSO:  show_graph 587
2.9.62 **predecessors** ............................. tail nodes of incoming arcs of a node

**CALLING SEQUENCE:**

```plaintext
a = predecessors(i,g)
```

**PARAMETERS:**

- `i` : integer
- `g` : graph list
- `a` : row vector of integers

**DESCRIPTION:**

`predecessors` returns the row vector of the numbers of the tail nodes of the incoming arcs to node `i` for a directed graph `g`.

**EXAMPLE:**

```plaintext
ta=[1 6 2 4 7 5 6 8 4 3 5 1];
he=[2 1 3 6 4 8 8 7 2 7 3 5];
g=make_graph('foo',1,8,ta,he);
g('node_x')=[285 284 335 160 405 189 118 45];
g('node_y')=[266 179 83 176 368 252 64 309];
show_graph(g);
a=predecessors(8,g)
show_nodes(a);
```

**SEE ALSO:** neighbors 577, successors 591

---

2.9.63 **qassign** ............................... solves a quadratic assignment problem

**CALLING SEQUENCE:**

```plaintext
[crit,order] = qassign(c,f,d)
```

**PARAMETERS:**

- `c` : real matrix
- `f` : real matrix
- `d` : real matrix
- `crit` : real scalar
- `order` : integer row vector

**DESCRIPTION:**

`qassign` solves the quadratic assignment problem i.e. minimize the global criterium: 

```plaintext
crit = e(1)+...+e(n)
```

where 

```plaintext
e(i) = c(i,l(i))+ fd(i)  where fd(i) = f(i,1)*d(l(i),l(1))+...+f(i,n)*d(l(i),l(n))
```

`c`, `f` and `d` are n x n real arrays; their diagonal entries are zero.

**EXAMPLE:**

```plaintext
n=15;
d=100*rand(15,15);
d=d-diag(diag(d));
c=zeros(n,n);f=c;
f(2:n,1)=ones(1:n-1)';
[crit,order]=qassign(c,f,d)
```

**SEE ALSO:** knapsack 564
2.9.64  salesman  solves the travelling salesman problem

CALLING SEQUENCE :

\[ \text{cir} = \text{salesman}(g, [\text{nstac}]) \]

PARAMETERS :

- \( g \) : graph list
- \( \text{nstac} \) : integer
- \( \text{cir} \) : integer row vector

DESCRIPTION :

salesman solves the travelling salesman problem. \( g \) is a directed graph; \( \text{nstac} \) is an optional integer which is a given bound for the allowed memory size for solving this problem. Its value is \( 100n^2 \) by default where \( n \) is the number of nodes.

EXAMPLE :

\[
\begin{align*}
\text{ta} &= [2 1 3 2 2 4 4 5 6 7 8 8 9 10 10 10 10 11 12 13 13 14 15 16 16 17 17]; \\
\text{he} &= [1 10 2 5 7 3 2 4 5 8 6 9 7 7 11 13 15 12 13 9 14 11 16 1 17 14 15]; \\
g &= \text{make_graph}('\text{foo}', 0, 17, \text{ta}, \text{he}); \\
g('\text{node}_x') &= [283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642]; \\
g('\text{node}_y') &= [59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301]; \\
g('\text{node}_diam') &= [1: (g('\text{node}_number'))] + 20; \\
\text{show_graph}(g); \\
g1 &= \text{make_graph}('\text{foo1}', 1, 17, [\text{ta he}], [\text{he ta}]); \\
m &= \text{arc_number}(g1); \\
g1('\text{edge}_length') &= 5 + \text{round}(30 * \text{rand}(1, m)); \\
cir &= \text{salesman}(g1); \\
\text{ii} &= \text{find(cir > edge_number(g))}; \\
\text{if (ii <> []) then cir(ii) = cir(ii) - edge_number(g); end; } \\
\text{show_arcs(cir)};
\end{align*}
\]

2.9.65  save_graph  saves a graph

CALLING SEQUENCE :

\[ \text{save_graph}(g, \text{path}) \]

PARAMETERS :

- \( g \) : graph list
- \( \text{name} \) : string, the path of the graph to save

DESCRIPTION :

save_graph saves the graph \( g \) in a graph file. \( \text{path} \) is the name of the graph file where the graph will be saved. \( \text{path} \) can be the name or the pathname of the file; if the "graph" extension is missing in \( \text{path} \), it is assumed. If \( \text{path} \) is the name of a directory, the name of the graph is used as the name of the file.

EXAMPLE :

```
Scilab Group  September 1995  585
```
g = load_graph(SCI+'/demos/metanet/mesh100');
show_graph(g);
unix('rm mymesh100.graph')
save_graph(g,'mymesh100.graph');
g = load_graph('mymesh100');
show_graph(g,'new');

SEE ALSO: load_graph 565

2.9.66 shortest_path

CALLING SEQUENCE :
[p,lp] = shortest_path(i,j,g,[typ])

PARAMETERS :
i : integer, number of start node
j : integer, number of end node
g : graph list
typ : string, type of shortest path
p : row vector of integer numbers of the arcs of the shortest path if it exists
lp : length of shortest path

DESCRIPTION :
shortest_path returns the shortest path p from node i to node j if it exists, and the empty vector [] otherwise. The optional argument typ is a string which defines the type of shortest path, 'arc' for the shortest path with respect to the number of arcs and 'length' for the shortest path with respect to the length of the edges edge_length.

For the shortest path with respect to the length of the edges, the lengths are given by the element edge_length of the graph list. If its value is not given (empty vector []), it is assumed to be equal to 0 on each edge. Lengths can be positive, equal to 0 or negative.

When a shortest path exists, lp is the length of this path.

EXAMPLE :
ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 13 13 13 14 14 14 9 11 10];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14 4 6 9 1];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[194 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[56 181 276 278 276 103 174 281 177 86 175 90 290 397 399];
show_graph(g);
g1=g;ma=prod(size(g1('head')));
rnd('uniform');
g1('edge_length')=int(20*rnd(1,ma));
[p,lp]=shortest_path(13,1,g1,'length');
p
x_message(['Showing the arcs of the shortest path ';
'Choose "Display arc names" in the Graph menu to see arc names']);
g1('edge_name')=string(g1('edge_length'));
edgencolor=ones(1,ma);edgencolor(p)=11*ones(p);
g1('edge_color')=edgencolor;
edgenvfsize=12*ones(1,ma);edgenvfsize(p)=18*ones(p);
g1('edge_font_size')=edgenvfsize;
show_graph(g1);

SEE ALSO: find_path 553, nodes_2_path 579
2.9.67  **show_arcs**  

Highlights a set of arcs

**CALLING SEQUENCE:**

show_arcs(p, [sup])

**PARAMETERS:**

p : row vector of arc numbers (directed graph) or edge numbers (undirected graph)
sup : string, superposition flag

**DESCRIPTION:**

show_arcs highlights the set of arcs or edges p of the displayed graph in the current Metanet window. If the optional argument sup is equal to the string 'sup', the highlighting is superposed on the previous one.

By default, this function works in asynchronous mode (see *metanet_sync*).

**EXAMPLE:**

```matlab
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 11 12 13 13 14 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);
t=min_weight_tree(1,g); g1=g; ma=edge_number(g1);
edgewidth=1*ones(1,ma); edgewidth(t)=4*ones(t); g1('edge_width')=edgewidth;
for i=8:12,
edgewidth(t)=i*ones(t); g1('edge_width')=edgewidth;
unix('sleep 2'); show_graph(g1);
show_arcs(t);
end;
```

**SEE ALSO:**  *metanet_sync* 572,  *show_nodes* 588

2.9.68  **show_graph**  

Displays a graph

**CALLING SEQUENCE:**

nw = show_graph(g, [smode, scale])
nw = show_graph(g, 'new', [scale, winsize])

**PARAMETERS:**

g : graph list
smode : string, mode value
winsize : row vector defining the size of Metanet window
scale : real vector, scale factor
nw : integer
DESCRIPTION:
show_graph displays the graph $g$ in the current Metanet window. If there is no current Metanet window, a Metanet window is created. The return value $nw$ is the number of the Metanet window where the graph is displayed.

If the optional argument $smode$ is equal to the string 'rep' or is not given and if there is already a graph displayed in the current Metanet window, the new graph is displayed instead.

If the optional argument $smode$ is equal to the string 'new', a new Metanet window is created. In this case, if the optional argument $winsize$ is given as a row vector $[width \, height]$, it is the size in pixels of Metanet window. The default is $[1000 \, 1000]$.

The optional argument $scale$ is the value of the scale factor when drawing the graph. The default value is 1.

The labels of the nodes and edges, if they exist, are displayed.

By default, this function works in asynchronous mode (see metanet_sync).

EXAMPLE:

```scilab
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 14 15 15 16 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g,2);
show_graph(g,0.5);
show_graph(g,1);
```

SEE ALSO: metanet_sync 572

2.9.69 show_nodes .......................................... highlights a set of nodes

CALLING SEQUENCE:

```scilab
show_nodes(nodes, [sup])
```

PARAMETERS:

- nodes : row vector of node numbers
- sup : string, superposition flag

DESCRIPTION:

show_nodes highlights the set of nodes $nodes$ of the displayed graph in the current Metanet window. If the optional argument $sup$ is equal to the string 'sup', the highlighting is superposed on the previous one.

By default, this function works in asynchronous mode (see metanet_sync).

EXAMPLE:

```scilab
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 14 15 15 16 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
for i=2:3:g('node_number'), show_nodes([i]); end;
for i=1:3:g('node_number'), show_nodes([i],'sup'); end;
```

SEE ALSO: metanet_sync 572, show_arcs 587
2.9.70  split_edge ___________________________  splits an edge by inserting a node

CALLING SEQUENCE:

\[ g1 = \text{split\_edge}(i,j,g,name) \]

PARAMETERS:

- \( i \) : integer, number of start node of edge
- \( j \) : integer, number of end node of edge
- \( g \) : graph list
- \( name \) : optional name of the added node
- \( g1 \) : graph list of the new graph

DESCRIPTION:

split\_edge returns the graph \( g1 \), the edge from node number \( i \) to node number \( j \) being splitted: a
new node is created and located at the middle point between the 2 previous nodes. This new node is linked
with the 2 nodes \( i \) and \( j \). If \( name \) is given, it is the name of the new node, otherwise the number of nodes
plus 1 is taken as the name of the new node.

EXAMPLE:

```plaintext
ta=[1 2 2 2 3 4 5 5 7 8 8 9 9 10 10 10 10 10 10 11 12 13 13 14 14 15 16 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
show_graph(g);
gt=split_edge(1,2,g);
show_graph(gt,'new');
```

SEE ALSO:  add\_edge 541,  add\_node 541,  delete\_arcs 551,  delete\_nodes 552

2.9.71  strong\_con\_nodes _________ set of nodes of a strong connected component

CALLING SEQUENCE:

\[ ns = \text{strong\_con\_nodes}(i,g) \]

PARAMETERS:

- \( i \) : integer, number of the strong connected component
- \( g \) : graph list
- \( ns \) : row vector, node numbers of the strong connected component

DESCRIPTION:

strong\_con\_nodes returns the row vector \( ns \) of the numbers of the nodes which belong to the strong
connected component number \( i \).

EXAMPLE:

```plaintext
Scilab Group September 1995 589
```
subgraph Scilab function

```plaintext
subgraph Scilab function

ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 12 13 13 13 13 14 15];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);
ncmp=strong_con_nodes(3,g);
n=g('node_number');
nodecolor=0*ones(1,n); nodecolor(ncmp)=11*ones(ncmp);
g('node_color')=nodecolor;
nodediam=20*ones(1,n); nodediam(ncmp)=40*ones(ncmp);
g('node_diam')=nodediam;
x_message('Set of nodes of the strong connected component #3');
show_graph(g);

See Also:  connex 549, con_nodes 548, strong_connex 590

2.9.72 strong_connex __________________________ strong connected components

CALLING SEQUENCE:

[nc,ncomp] = strong_connex(g)

PARAMETERS:

g : graph list
nc : integer, number of strong connected components
ncomp : row vector of strong connected components

DESCRIPTION:

strong_connex returns the number nc of strong connected components for the graph g and a row vector ncomp giving the number of the strong connected component for each node. For instance, if i is a node number, ncomp[i] is the number of the strong connected component to which node i belongs.

EXAMPLE:

ta=[1 1 2 2 2 3 4 4 5 6 6 6 7 7 7 8 9 10 12 12 12 13 13 13 13 14 15];
he=[2 6 3 4 5 1 3 5 1 7 10 11 5 8 9 5 8 11 10 11 9 11 15 13 14];
g=make_graph('foo',1,15,ta,he);
g('node_x')=[197 191 106 194 296 305 305 418 422 432 552 550 549 416 548];
g('node_y')=[76 181 276 278 276 83 174 281 177 86 175 90 290 397 399];
show_graph(g);
[nc,ncomp]=strong_connex(g);
g1=g; g1('node_color')=8+ncomp; g1('node_diam')=10+5*ncomp;
x_message('Connected components of the graph');
show_graph(g1);

See Also:  connex 549, con_nodes 548, strong_con_nodes 589

2.9.73 subgraph ____________________________ subgraph of a graph

CALLING SEQUENCE:

g1 = subgraph(v,ind,g)

Scilab Group  September 1996  590
```
PARAMETERS:

v : row vector, numbers of nodes or edges
ind : string, 'nodes' or 'edges'
g : graph list
g1 : graph list of the new graph

DESCRIPTION:
subgraph returns the graph g1, built with the numbers given by the row vector v. If ind is the string 'nodes', g1 is built with the node numbers given by v and the connected edges of these nodes in g. If ind is the string 'edges', g1 is built with the edge numbers given by v and the tail-head nodes of these edges in g.

All the characteristics of the old nodes and edges of g are preserved.

EXAMPLE:

```scilab
ta=[1 1 2 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 14 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
v=[2 3 4 5 17 13 10];
show_nodes(v);
g1=subgraph(v,'nodes',g);
show_graph(g1);
v=[10 13 12 16 20 19];
show_graph(g);
show_arcs(v);
g1=subgraph(v,'edges',g);
show_graph(g1);
```

SEE ALSO: add_edge 541, add_node 541, delete_arcs 551, delete_nodes 552, supernode 592

2.9.74 successors _________________ head nodes of outgoing arcs of a node

CALLING SEQUENCE:

a = successors(i, g)

PARAMETERS:

i : integer
g : graph list
a : row vector of integers

DESCRIPTION:

successors returns the row vector of the numbers of the head nodes of the outgoing arcs from node i for a directed graph g.

EXAMPLE:

```scilab
Scilab Group September 1996
```
trans_closure Scilab function

ta=[1 6 2 4 7 5 6 8 4 3 5 1];
he=[2 1 3 6 4 8 7 2 7 3 5];
g=make_graph('foo',1,8,ta,he);
g('node_x')=[285 284 335 160 405 189 118 45];
g('node_y')=[266 179 83 176 368 252 64 309];
show_graph(g);
a=successors(6,g)
show_nodes(a);

SEE ALSO: neighbors 577, predecessors 584

2.9.75 supernode _______________ replaces a group of nodes with a single node

CALLING SEQUENCE:
g1 = supernode(v, g)

PARAMETERS:
v : row vector, nodes numbers
:g : graph list
:g1 : graph list of the new graph

DESCRIPTION:
supernode returns the graph g1 with the nodes with numbers given by the vector v being contracted in a single node. The number of the supernode is the lowest number in v. The characteristics of the old nodes and edges are preserved. The supernode is located at the mean center of v. Its diameter and border are twice the previous of the replaced node.

The demand of the new node, if it exists, is the sum of the demands of the shrunken nodes.

EXAMPLE:
ta=[1 1 2 2 3 4 5 5 7 8 8 9 10 10 10 10 10 11 12 13 13 14 15 16 16 17 17];
he=[2 10 3 5 7 4 2 4 6 8 6 9 7 7 11 13 13 15 12 13 9 10 14 11 16 1 17 14 15];
g=make_graph('foo',1,17,ta,he);
g('node_x')=[283 163 63 57 164 164 273 271 339 384 504 513 439 623 631 757 642];
g('node_y')=[59 133 223 318 227 319 221 324 432 141 209 319 428 443 187 151 301];
g('edge_color')=modulo([1:(edge_number(g))],15)+1;
g('node_diam')=[1:(g('node_number'))]+20;
show_graph(g);
v=[7 10 13 9];
show_nodes(v);g1=supernode(v,g);
show_graph(g1,'new');

SEE ALSO: add_edge 541, add_node 541, delete_arcs 551, delete_nodes 552

2.9.76 trans_closure ______________________________________________________________________ transitive closure

CALLING SEQUENCE:
g1 = trans_closure(g)

PARAMETERS:

g : graph list

g1 : graph list

DESCRIPTION:

trans_closure returns as a new graph list g1 the transitive closure of the graph g. This graph must be directed and connected. If <name> if the name of graph g. trans_closure is the name of the transitive closure.

EXAMPLE:

ta=[2 3 3 5 4 4 5 8];
he=[1 2 4 2 6 7 7 4];
g=make_graph('foo',1,8,ta,he);
g('node_x')=[129 200 283 281 128 366 122 333];
g('node_y')=[61 125 129 189 173 135 236 249];
show_graph(g);
g1=trans_closure(g);
vv=1*ones(ta); aa=sparse([ta' he'],vv');
ta1=g1('tail'); he1=g1('head');
ww=1*ones(ta1); bb=sparse([ta1' he1'],ww');
dif=bb-aa; lim=size(ta1); edgecolor=0*ones(ta1);
for i=1:lim(2)
    if dif(ta1(i),he1(i))==1 then edgecolor(i)=11; end;
end;
g1('edge_color')=edgecolor;
x_message('Transitive closure of the graph');
show_graph(g1);
2.10 Scicos
2.10.1 Scicos editor

2.10.1.1 scicos ____________ Block diagram editor and GUI for the hybrid simulator scicosim

CALLING SEQUENCE:

sys=scicos()
sys=scicos(sys,[menus])
sys=scicos(file,[menus])

PARAMETERS:

sys : a Scicos data structure
file : a character string. The path of a file containing the image of a Scicos data structure. These files may have .cos or .cosf extensions.
menus : a vector of character strings. It allows to select some of the Scicos menus. If menus==[] Scicos draws the diagram and the contents of each super blocks in separate windows without menu bar. This option is useful to print diagrams.

DESCRIPTION:
Scicos is a visual editor for constructing models of hybrid dynamical systems. Invoking Scicos with no argument opens up an empty Scicos window. Models can then be assembled, loaded, saved, compiled, simulated, using GUI of Scicos. The input and output arguments are only useful for debugging purposes. Scicos serves as an interface to the various block diagram compilers and the hybrid simulator scicosim.

SEE ALSO: scicosim 626, scicos_main 620, scicos_menus 595

2.10.1.2 scicos_menus ________________________________ Scicos menus description

DESCRIPTION:
Here is a list of operations available in Scicos:

Main menus:

Edit : Opens the diagram/palette edition menu.
Simulate : Opens the compilation/execution menu.
Diagram : Opens the diagram/file management menu.
Block : Opens the block management menu.
Misc : Opens miscellaneous menu items.
Diagram/palette edition menu. : This menu allows to edit diagram and palettes
Palettes : opens up a selection dialog where user may select a desired palette among all defined palettes.
Context : opens up a dialog where user may enter and modify Scilab instructions to be executed when diagram is loaded (Edit../Load menu) or evaluated (Simulate../Eval menu) (of course instructions are also evaluated when dialog returns). These instructions may be used to define Scilab variables whose names are used in the block parameters definition expressions.
Move : To move a block in main Scicos window, select first the Move menu item, then click on the selected block, drag the mouse to the desired block position and click again to fix the position.
Copy : To copy a block in main Scicos window, select first the Copy menu item, then click left on the to-be-copied block (in Scicos window or in a palette), and finally click where you want the copy to be placed in Scicos window. This menu item remains active until user choose an other one
Copy Region: To copy a region in main Scicos window, select first the Copy menu item, then click right on a corner of the desired region (in Scicos window or in a palette), drag to select the desired region, click to fix the selected region and finally click where you want the copy to be placed in Scicos window. If source diagram is a big region, selection may take a while.
Replace : To replace a block in the active editor Scicos window select first the Replace menu item, then select the replacement block (in Scicos window or in a palette), and finally click on the to-be-replaced block. It is not possible to replace a connected block with another block with different port locations.

Align : To obtain nice diagrams, you can align ports of different blocks, vertically and horizontally. Select first the Align menu item, then on the first port and finally on the second port. The block corresponding to the second port is moved. Connected blocks cannot be aligned.

AddNew : To add a newly defined block to the current palette or diagram select first this menu item, a dialog box will pop up asking for the name of the GUI function associated with the block. If this function is not already loaded it is searched in the current directory. The user may then click at the desired position of the block in the palette or diagram.

Link : This menu item is defined only in diagram edition mode. To connect an output port to an input port, select first the Link menu item, then on the intermediate points, if necessary, and finally on the input port. Scicos tries to draw horizontal and vertical lines to form links.

To split a link, select first the Link menu item, then on the link where the split should be placed, and finally on an input port. Only one link can go from and to a port. Link color can be changed directly by clicking on the link.

This menu item remains active until user choose an other one

Delete : To delete a block or a link, select first the Delete menu item, then click left on the selected object. If you delete a block all links connected to it are deleted as well. This menu item remains active until user choose an other one.

Delete Region : To delete a region in main Scicos window select first the Delete Region menu item, then click right on a corner of the desired region (in Scicos window or in a palette), drag to select the desired region, click to fix the selected region. If source diagram is a big region, selection may take a while.

Flip : To reverse the positions of the (regular) inputs and outputs of a block placed on its sides, click on the Flip menu item first and then on the selected block. This does not affect the order, nor the position of the input and output event ports which are numbered from left to right. A connected block cannot be flipped.

Undo : Click on the Undo menu item to undo the last edit operation.

Simulation menu :

Setup : In the main Scicos window, clicking on the Setup menu item invokes a dialog box that allows you to change integration parameters: absolute and relative error tolerances for the ode solver, the time tolerance (the smallest time interval for which the ode solver is used to update continuous states), and the maximum time increase realized by a single call to the ode solver.

Compile : This menu item need never be used since compilation is performed automatically, if necessary, before the beginning of every simulation (Run menu item).

Normally, a new compilation is not needed if only system parameters and internal states are modified. In some cases however modifications are not correctly updated and a manual compilation may be needed before a Restart or a Continue. Click on this menu item to compile the block diagram. Please report if you encounter such a case.

Eval : Blocks dialogs answers can be defined using Scilab expressions. These expressions are evaluated immediately and they are also stored as character strings. Click on the Eval menu item to have them re-evaluated according to the new values of underlying Scilab variables defined by context for example.

Run : To start the simulation. If the system has already been simulated, a dialog box appears where you can choose to Continue, Restart or a Continue. You may interrupt the simulation by clicking on the "stop" button, change any of the block parameters and continue or restart the simulation with the new values.

Diagram menu :

Replot : Scicos window stores the complete history of the editing session. Click on the Replot menu item to erase the history and replot the diagram or palette. Replot diagram before printing or exporting Scicos diagrams.

New : Clicking on the New menu item creates an empty diagram in the main Scicos window. If the previous
content of the window is not saved, it will be lost.

Purge : Suppress deleted blocks out of Scicos data structure. This menu changes block indexing and implies compilation of the diagram before compilation.

Rename : Click on this menu item to change the diagram or palette’s name. A dialog window will pop up.

Make block : Click on this menu item to save the Super Block as a new Scicos block. A Scilab function is generated and saved in <window_name>.sci file in the desired directory. <window_name> is the name of the Super Block appearing on top of the window. A dialog allows choosing the directory. This block may be added to a palette using Edit/AddNew menu item.

Save : Saves the block diagram in the current binary file selected by a previous call to SaveAs or Load menu item. If no current binary file, diagram is saved in the current directory as <window_name>.cos.

Save As : Saves the block diagram in a binary file. A file selection dialog will pop up.

FSave : Save the diagram in a formatted ascii file. A dialog box allows choosing the file name which must have a .cos extension.

Formatted save is slower than regular save but has the advantage that the generated file is system independent (usefull for exchanging data on different computers.

Load : Loads an ascii or binary file containing a saved block diagram. A file selection dialog will pop up.

Save as Palette : select the Save as Palette menu item to save the block diagram as a palette in a binary file. A dialog box allows choosing the file which must have a .cos extension. The palette takes the name of the file (without the extension).

save user file is updated.

FSave as Palette : select the FSave as Palette menu item to save the block diagram as a palette in an ascii formatted file. A dialog box allows choosing the file which must have a .cosf extension. The palette takes the name of the file (without the extension).

Load as Palette :select the Load menu item to load an ascii or binary file containing a saved block diagram as a palette. A dialog box allows user choosing the file.

Exit : Click on the Exit menu item to close current diagram. If current diagram is not a Super block Exit menu item leave Scicos and return to Scilab session. Save your diagram or palette before leaving.

Object menu :

Set :To change the parameters of a regular block or link, to open a super block, select first this menu item, click next on the desired object. A dialog or edition window appear that allows you to modify object.

Resize : To change the size of a block, select first this menu item, click next on the desired block. A dialog appears that allows you to change the width and/or height of the block shape.

Icon : To change the icon of a block drawn by standard draw, select first this menu item, click next on the desired block. A dialog appears that allows you to enter Scilab instructions used to draw the icon. These instructions may refer to orig and sz variables and more generally to the block data structure named o in this context (see scicos block). If Icon description selects colors for drawing, it is necessary to get it through scs_color function to have Color menu item work properly.

Color : To change the background of a block drawn by standard draw, or color of a link select first this menu item, click next on the selected object. A color palette appears where user may select the block background color.

Label : To change or define the blocks label, select first this menu item, click next on the desired block. A dialog appears that allows you to enter the desired label. Labels may be used within blocks computational functions as an identification (see getlabel function).

Miscellaneous menu :

Window : Clicking on the Window menu item invokes a dialog box that allows you to change the editor window dimensions.

Shift :To shift the diagram to left, right, up or down, select this menu item, then click on the point you want to appear in the middle of the graphics window.

Zoom in : When you select this menu item the diagram is zoomed in by a factor of 10%

Zoom out : When you select this menu item the diagram is zoomed out by a factor of 10%

Options : Select this menu item to set display options.
Help: To get help on an object or menu item, select first Help menu item and then on the selected object or menu item.

Calc: When you click on this menu item you switch Scilab to the pause mode (see the help on pause).
In the Scilab main window and you may enter Scilab instructions to compute whatever you want. To go back to Scicos you need to enter ""return"" or ""[...]=return(...)"" Scilab instruction. If you use ""[...]=return(...)"" Scilab instruction take care not to modify Scicos variables such as ""scs_m",""scs_gc",""menus",""datam"".... If you have modified Scicos graphic window you may restore it using the Scicos ""Replot"" menu.

SEE ALSO: scicos 595

2.10.2 Blocks

2.10.2.1 ABSBLK Scicos abs block

DIALOGUE PARAMETERS:
None.
DESCRIPTION:
This block realizes element-wise vector absolute value operation. This block has a single input and a single output port. Port dimension is determined by the context.

2.10.2.2 AFFICH Scicos numerical display

DIALOGUE PARAMETERS:
font : integer, the selected font number (see xset)
fontsize : integer, the selected font size (set xset)
color : integer, the selected color for the text (see xset)
Total number of digits : an integer greater than 3, the maximum number of digits used to represent the number (sign, integer part and rational part)
rational part number of digits : an integer greater than or equal 0, the number of digits used to represent the rational part

DESCRIPTION:
This block displays the value of its unique input inside the block (in the diagram) during simulation. The block must be located in the main Scicos window.
Warning: each time the block is moved user must click on it to set its parameters. The display position is then automatically updated.
SEE ALSO: SCOPE 615

2.10.2.3 ANDLOG Scicos logical AND block

DIALOGUE PARAMETERS:
None.
DESCRIPTION:
This block, with two event inputs and a regular output, outputs +1 or -1 on its regular output depending on input events.
+1 : When events are synchronously present on both event input ports
-1 : When only one event is present.
SEE ALSO: IFTHEL 608
2.10.2.4 ANIMXY\textsubscript{f} Scicos 2D animated visualization block

**DESCRIPTION:**
This block realizes the visualization of the evolution of the two regular input signals by drawing the second input as a function of the first at instants of events on the event input port.

**DIALOGUE PARAMETERS:**

- **Curve colors:** an integer. It is the color number ($\geq 0$) or marker type ($< 0$) used to draw the evolution of the input port signal. See \texttt{xset()} for color (dash type) definitions.
- **Line or mark size:** an integer.
- **Output window number:** The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired).
- **Output window position:** a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer \texttt{[]} for default window position.
- **Output window size:** a 2 vector specifying the width and height of the graphic window. Answer \texttt{[]} for default window dimensions.
- **Xmin, Xmax:** Minimum and maximum values of the first input; used to set up the X-axis of the plot in the graphics window.
- **Ymin, Ymax:** Minimum and maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.
- **Buffer size:** an integer. In order to minimize the number of graphics outputs, data may be buffered.

**REMARKS:**
Output window number, Output window size, Output window position are only taken into account at the initialisation time of the simulation.

**SEE ALSO:** \texttt{SCOPE\_f 615}, \texttt{EVENTSCOPE\_f 605}, \texttt{SCOPXY\_f 615}

2.10.2.5 BIGSOM\textsubscript{f} Scicos addition block

**DIALOGUE PARAMETERS:**

- **Input signs:** a vector \texttt{sgn} of weights (generally +1 or -1). The number of input signs fix the number of input ports.

**DESCRIPTION:**
This block realizes weighted sum of the input vectors. The output is vector \texttt{kth component} is the sum of the \texttt{kth components} of each input ports weighted by \texttt{sgn (k)}.

**SEE ALSO:** \texttt{GAIN\_f 606}, \texttt{SOM\_f 616}

2.10.2.6 CLINDUMMY\textsubscript{f} Scicos dummy continuous system with state

**DESCRIPTION:**
This block should be placed in any block diagram that contains a zero-crossing block but no continuous system with state. The reason for that is that it is the ode solver that find zero crossing surfaces.

**SEE ALSO:** \texttt{ZCROSS\_f 619}
2.10.2.7 **CLKINV** Scicos Super Block event input port

**DESCRIPTION:**
This block must only be used inside Scicos Super Blocks to represent an event input port.
In a Super Block, the event input ports must be numbered from 1 to the number of event input ports.

**DIALOGUE PARAMETERS:**
- Port number: an integer defining the port number.

**SEE ALSO:**  IN\_f 609, OUT\_f 611, CLKOUTV\_f 600

2.10.2.8 **CLKIN** Scicos Super Block event input port

**DESCRIPTION:**
This block must only be used inside Scicos Super Blocks to represent an event input port.
In a Super Block, the event input ports must be numbered from 1 to the number of event input ports.

**DIALOGUE PARAMETERS:**
- Port number: an integer defining the port number.

**SEE ALSO:**  IN\_f 609, OUT\_f 611, CLKOUT\_f 600

2.10.2.9 **CLKOUTV** Scicos Super Block event output port

**DESCRIPTION:**
This block must only be used inside Scicos Super Blocks to represent an event output port.
In a Super Block, the event output ports must be numbered from 1 to the number of event output ports.

**DIALOGUE PARAMETERS:**
- Port number: an integer giving the port number.

**SEE ALSO:**  IN\_f 609, OUT\_f 611, CLKINV\_f 600

2.10.2.10 **CLKOUT** Scicos Super Block event output port

**DESCRIPTION:**
This block must only be used inside Scicos Super Blocks to represent an event output port.
In a Super Block, the event output ports must be numbered from 1 to the number of event output ports.

**DIALOGUE PARAMETERS:**
- Port number: an integer giving the port number.

**SEE ALSO:**  IN\_f 609, OUT\_f 611, CLKIN\_f 600

2.10.2.11 **CLKSOMV** Scicos event addition block

**DIALOGUE PARAMETERS:**
None.

**DESCRIPTION:**
This block is an event addition block with up to three inputs. The output reproduces the events on all the input ports. Strictly speaking, CLKSOMV is not a Scicos block because it is discarded at the compilation phase. The inputs and output of CLKSOMV are synchronized.

Scilab Group Janvier 1996 600
2.10.2.12  CLKSOM  Scicos event addition block

DIALOGUE PARAMETERS:
None.

DESCRIPTION:
This block is an event addition block with up to three inputs. The output reproduces the events on all the input ports. Strictly speaking, CLKSOM is not a Scicos block because it is discarded at the compilation phase. The inputs and output of CLKSOM are synchronized.

2.10.2.13  CLKSPLIT  Scicos event split block

DIALOGUE PARAMETERS:
None.

DESCRIPTION:
This block is an event split block with an input and two outputs. The outputs reproduces the event the input port on each output ports. Strictly speaking, CLKSPLIT is not a Scicos block because it is discarded at the compilation phase. This block is automatically created when creating a new link issued from a link.

The inputs and output of CLKSPLIT are synchronized.

2.10.2.14  CLOCK  Scicos periodic event generator

DESCRIPTION:
This block is a Super Block constructed by feeding back the output of an event delay block into its input event port. The unique output of this block generates a regular train of events.

DIALOGUE PARAMETERS:

- **Period**: scalar. One over the frequency of the clock. Period is the time that separates two output events.
- **Init time**: scalar. Starting date. If negative the clock never starts.

SEE ALSO:  EVTDLY

2.10.2.15  CLR  Scicos continuous-time linear system (SISO transfer function)

DIALOGUE PARAMETERS:

- **Numerator**: a polynomial in s.
- **Denominator**: a polynomial in s.

DESCRIPTION:
This block realizes a SISO linear system represented by its rational transfer function $\frac{\text{Numerator}}{\text{Denominator}}$. The rational function must be proper.

SEE ALSO:  CLSS, INTEGRAL

Scilab Group  Janvier 1996  601
2.10.2.16  CLSS.f  Scicos continuous-time linear state-space system

DESCRIPTION:
This block realizes a continuous-time linear state-space system.

\[ \dot{x} = Ax + Bu \]
\[ y = Cx + Du \]

The system is defined by the \((A,B,C,D)\) matrices and the initial state \(x_0\). The dimensions must be compatible.

DIALOGUE PARAMETERS:

\(A\) : square matrix. The A matrix
\(B\) : the B matrix, \([\]\) if system has no input
\(C\) : the C matrix, \([\]\) if system has no output
\(D\) : the D matrix, \([\]\) if system has no D term.
\(x_0\) : vector. The initial state of the system.

SEE ALSO: CLR_f 601, INTEGRAL_f 608

2.10.2.17  CONSF  Scicos constant value(s) generator

DIALOGUE PARAMETERS:

\(\text{constants}\) : a real vector. The vector size gives the size of the output port. The value \(\text{constants}(i)\) is assigned to the \(i\)th component of the output.

DESCRIPTION:
This block is a constant value(s) generator.

2.10.2.18  COSBLK.f  Scicos cosine block

DIALOGUE PARAMETERS:

None.

DESCRIPTION:
This block realizes vector cosine operation. \(y(i) = \cos(u(i))\). The port input and output port sizes are equal and determined by the context.

SEE ALSO: SINBLK_f 616, GENSIN_f 607

2.10.2.19  CURVF  Scicos block, tabulated function of time

DIALOGUE PARAMETERS:

Tabulated function is entered using a graphics curve editor (see edit_curv in Scilab documentation)

DESCRIPTION:
This block defines a tabulated function of time. Between mesh points block performs a linear interpolation. Outside tabulation block outputs last tabulated value.

User may define the tabulation of the function using a curve editor.
2.10.2.20 DELAYV.f Scicos time varying delay block

DIALOGUE PARAMETERS:

- **Number inputs**: size of the delayed vector (-1 not allowed)
- **Register initial state**: register initial state vector. Dimension must be greater than or equal to 2
- **Max delay**: Maximum delay that can be produced by this block

DESCRIPTION:
This block implements a time varying discretized delay. The value of the delay is given by the second input port. The delayed signal enters the first input port and leaves the unique output port.

The first event output port must be connected to unique input event port if auto clocking is desired. But the input event port can also be driven by outside clock. In that case, the max delay is size of initial condition times the period of the incoming clock.

The second output event port generates an event if the second input goes above the maximum delay specified. This signal can be ignored. In that case the output will be delayed by max delay.

SEE ALSO: DELAY_f 603, EVTDLY_f 605, REGISTER_f 613

2.10.2.21 DELAY.f Scicos delay block

DIALOGUE PARAMETERS:

- **Discretization time step**: positive scalar, delay discretization time step
- **Register initial state**: register initial state vector. Dimension must be greater than or equal to 2

DESCRIPTION:
This block implements as a discretized delay. It is in fact a Scicos super block formed by a shift register and a clock.

The value of the delay is given by the discretization time step multiplied by the number of states of the register minus one.

SEE ALSO: DELAYV_f 603, EVTDLY_f 605, REGISTER_f 613

2.10.2.22 DEMUX.f Scicos demultiplexer block

DIALOGUE PARAMETERS:

- **number of output ports**: positive integer less than or equal to 8.

DESCRIPTION:
Given a vector valued input this block splits inputs over vector valued outputs. So $u=[y_1; y_2; \ldots; y_n]$, where $y_i$ are numbered from top to bottom. Input and Output port sizes are determined by the context.

SEE ALSO: MUX_f 610
2.10.2.23  DLRADAPT_f  Scicos discrete-time linear adaptive system

DIALOGUE PARAMETERS:

- Vector of p mesh points: a vector which defines u2 mesh points. Numerator roots: a matrix, each line gives the roots of the numerator at the corresponding mesh point.
- Denominator roots: a matrix, each line gives the roots of the denominator at the corresponding mesh point.
- Gain: a vector, each vector entry gives the transfer gain at the corresponding mesh point.
- Past inputs: a vector of initial value of past degree (Numerator) inputs.
- Past outputs: a vector of initial value of past degree (Denominator) outputs.

DESCRIPTION:
This block realizes a SISO linear system represented by its rational transfer function whose numerator and denominator roots are tabulated functions of the second block input. The rational function must be proper. Roots are interpolated linearly between mesh points.

See Also: DLSS_f 604, DLR_f 604

2.10.2.24  DLR_f  Scicos discrete-time linear system (transfer function)

DIALOGUE PARAMETERS:

- Numerator: a polynomial in z.
- Denominator: a polynomial in z.

DESCRIPTION:
This block realizes a SISO linear system represented by its rational transfer function (in the symbolic variable z). The rational function must be proper.

See Also: DLSS_f 604, DLRADAPT_f 604

2.10.2.25  DLSS_f  Scicos discrete-time linear state-space system

DESCRIPTION:
This block realizes a discrete-time linear state-space system. The system is defined by the (A,B,C,D) matrices and the initial state x0. The dimensions must be compatible. At the arrival of an input event on the unique input event port, the state is updated.

DIALOGUE PARAMETERS:

- B: the B matrix.
- C: the C matrix.
- x0: vector. The initial state of the system.

See Also: DLR_f 604, INTEGRAL_f 608, CLSS_f 602, DLSS_f 604
2.10.2.26 EVENTSCOPE.f ____________________________ Scicos event visualization block

DESCRIPTION :
This block realizes the visualization of the input event signals.

DIALOGUE PARAMETERS :

Number of event inputs : an integer giving the number of event input ports colors : a vector of integers. The i-th element is the color number (>=0) or dash type (<0) used to draw the evolution of the i-th input port signal. See xset for color (dash type) definitions.

Output window number : The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don't have the same window numbers (unless superposition of the curves is desired). Output window position : a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.

Output window size : a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.

Refresh period : Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.

REMARKS :
Output window number, Output window size, Output window position are only taken into account at the initialisation time of the simulation.

SEE ALSO : SCOPXY.f 615, SCOPE.f 615, ANIMXY.f 599

2.10.2.27 EVTDLY.f ________________________________ Scicos event delay block

DESCRIPTION :
One event is generated Delay after an event enters the unique input event port. Block may also generate an initial output event.

DIALOGUE PARAMETERS :

Delay : scalar. Time delay between input and output event.
Auto-exec : scalar. If Auto-exec>=0 block initially generates an output event at date Auto-exec.

SEE ALSO : CLOCK.f 601

2.10.2.28 EVTGEN.f ________________________________ Scicos event firing block

DESCRIPTION :
One event is generated on the unique output event port if Event time is larger than equal to zero, if not, no event is generated.

DIALOGUE PARAMETERS :

Event time : scalar. date of the initial event

SEE ALSO : CLOCK.f 601, EVTDLY.f 605

Scilab Group Janvier 1997 605
2.10.2.29  EXPBLK  Scicos aˆu block

DIALOGUE PARAMETERS :

a : real positive scalar

DESCRIPTION :
This block realizes $y(i) = a\cdot u(i)$. The input and output port sizes are determined by the compiler.

2.10.2.30  GAINBLK  Scicos gain block

DIALOGUE PARAMETERS :

Gain : a real matrix.

DESCRIPTION :
This block is a gain block. The output is the Gain times the regular input (vector). The dimensions of Gain
determines the input (number of columns) and output (number of rows) port sizes.

2.10.2.31  GAIN  Scicos gain block

DIALOGUE PARAMETERS :

Gain : a real matrix.

DESCRIPTION :
This block is a gain block. The output is the Gain times the regular input (vector). The dimensions of Gain
determines the input (number of columns) and output (number of rows) port sizes.

This block is obsolete. Use GAINBLK block instead of it

2.10.2.32  GENERAL  Scicos general zero crossing detector

DESCRIPTION :
Depending on the sign (just before the crossing) of the inputs and the input numbers of the inputs that
have crossed zero, an event is programmed (or not) with a given delay, for each output. The number of
combinations grows so fast that this becomes unusable for blocks having more than 2 or 3 inputs. For the
moment this block is not documented.

DIALOGUE PARAMETERS :

Size of regular input : integer.
Number of output events : integer.
the routing matrix : matrix, number of rows is the number of output events. The columns cor-
respond to each possible combination of signs and zero crossings of the inputs. The entries of the
matrix give the delay for generating the output event ($\leq 0$ no event is generated).

SEE ALSO:  NEGTOPOS_f 611,  POSTONEG_f 611,  ZCROSS_f  619
2.10.2.33 GENERIC_f

DESCRIPTION:
This block can realize any type of block. The computational function must already be defined in Scilab, Fortran or C code.

DIALOGUE PARAMETERS:
- simulation function: a character string, the name of the computational function
- function type: a non-negative integer, the type of the computational function
- input port sizes: a vector of integers, size of regular input ports.
- output port sizes: a vector of integers, size of regular output ports.
- input event port sizes: a vector of ones, size of event input ports. The size of the vector gives the number of event input ports.
- output event port sizes: a vector of ones, size of event output ports. The size of the vector gives the number of event output ports.
- Initial continuous state: a column vector.
- Initial discrete state: a column vector.
- System type: a string: c, d, z or l (CBB, DBB, zero crossing or synchro).
- Real parameter vector: column vector. Any parameters used in the block can be defined here as a column vector.
- Integer parameter vector: column vector. Any integer parameters used in the block can be defined here as a column vector.
- initial firing: vector. Size of this vector corresponds to the number of event outputs. The value of the i-th entry specifies the time of the preprogrammed event firing on the i-th output event port. If less than zero, no event is preprogrammed.
- direct feedthrough: character "y" or "n", specifies if block has a direct input to output feedthrough.
- Time dependance: character "y" or "n", specifies if block output depends explicitly on time.

SEE ALSO: scifunc_block 619

2.10.2.34 GENSIN_f

DESCRIPTION:
This block is a sine wave generator: \( M \times \sin(F \times t + P) \)

DIALOGUE PARAMETERS:
- Magnitude: a scalar. The magnitude \( M \).
- Frequency: a scalar. The frequency \( F \).
- Phase: a scalar. The phase \( P \).

SEE ALSO: GENSQR_f 607, RAND_f 612, SAWTOOTH_f 614

2.10.2.35 GENSQR_f

DESCRIPTION:
This block is a square wave generator: output takes values \(-M\) and \(M\). Every time an event is received on the input event port, the output switches from \(-M\) to \(M\), or \(M\) to \(-M\).

DIALOGUE PARAMETERS:
- Amplitude: a scalar \( M \).

SEE ALSO: GENSIN_f 607, SAWTOOTH_f 614, RAND_f 612

Scilab Group Janvier 1996 607
2.10.2.36  HALT.f  Scicos Stop block

DIALOGUE PARAMETERS:
State on halt: scalar. A value to be placed in the state of the block. For debugging purposes this allows to distinguish between different halts.

DESCRIPTION:
This block has a unique input event port. Upon the arrival of an event, the simulation is stopped and the main Scicos window is activated. Simulation can be restarted or continued (Run button).

2.10.2.37  IFTHEL.f  Scicos if then else block

DIALOGUE PARAMETERS:
None.

DESCRIPTION:
One event is generated on one of the output event ports when an input event arrives. Depending on the sign of the regular input, the event is generated on the first or second output.
This is a synchro block, i.e., input and output event are synchronized.

2.10.2.38  INTEGRAL.f  Scicos simple integrator

DESCRIPTION:
This block is an integrator. The output is the integral of the input.

DIALOGUE PARAMETERS:
Initial state: a scalar. The initial condition of the integrator.

SEE ALSO:  CLSS_f 602,  CLR_f 601

2.10.2.39  INTRP2BLK.f  Scicos 2D linear interpolation block

DIALOGUE PARAMETERS:
X coord.: an n-vector (strictly increasing)
Y coord.: an m-vector (strictly increasing)
Z values: an mxn matrix

DESCRIPTION:
The output of this block is a function of the inputs obtained by bilinear interpolation. This block has two scalar inputs and a single scalar output. The X(i) and Y(i) give respectively the X coordinate and the Y coordinate of the i-th data point to be interpolated and Z(Y(i),X(i)) its value.

2.10.2.40  INTRPLBLK.f  Scicos linear interpolation block

DIALOGUE PARAMETERS:
X coord.: a vector (strictly increasing)
Y coord.: a vector (same size as X coord)

DESCRIPTION:
The output of this block is a function of the input obtained by linear interpolation. This block has a single scalar input and a single scalar output port. The X coord. and Y coord. give respectively the X coordinate and the Y coordinate of the data points to be interpolated. X coord must be strictly increasing.
2.10.2.41 INVBLK_f Scicos inversion block

**DIALOGUE PARAMETERS:**
None.

**DESCRIPTION:**
This block computes \( y(i) = 1/u(i) \). The input (output) size is determined by the context.

2.10.2.42 IN_f Scicos Super Block regular input port

**DESCRIPTION:**
This block must only be used inside Scicos Super Blocks to represent a regular input port. The input size is determined by the context.
In a Super Block, regular input ports must be numbered from 1 to the number of regular input ports.

**DIALOGUE PARAMETERS:**
- **Port number**: an integer giving the port number.

**SEE ALSO:** CLKIN_f 600, OUT_f 611, CLKOUT_f 600

2.10.2.43 LOGBLK_f Scicos logarithm block

**DIALOGUE PARAMETERS:**
- **a**: real scalar greater than 1

**DESCRIPTION:**
This block realizes \( y(i) = \log(u(i))/\log(a) \). The input and output port sizes are determined by the context.

2.10.2.44 LOOKUP_f Scicos Lookup table with graphical editor

**DESCRIPTION:**
This block realizes a non-linear function defined using a graphical editor.

2.10.2.45 MAX_f Scicos max block

**DIALOGUE PARAMETERS:**
None.

**DESCRIPTION:**
The block outputs the maximum of the input vector: \( y = \max(u_1, \ldots, u_n) \). The input vector size is determined by the compiler according to the connected blocks port sizes.

**SEE ALSO:** MIN_f 610
2.10.2.46  MCLOCK_f                      Scicos 2 frequency event clock

DESCRIPTION:
This block is a Super Block constructed by feeding back the outputs of an MFCLCK block into its input event port. The two outputs of this block generate regular train of events, the frequency of the first input being equal to that of the second output divided by an integer n. The two outputs are synchronized (this is impossible for standard blocks; this is a Super Block).

DIALOGUE PARAMETERS:

Basic period: scalar. equals 1/f, f being the highest frequency.
n: an integer > 1. the frequency of the first output event is f/n.

SEE ALSO: MFCLCK_f 610, CLOCK_f 601

2.10.2.47  MFCLCK_f                      Scicos basic block for frequency division of event clock

DESCRIPTION:
This block is used in the Super Block MCLOCK. The input event is directed once every n times to output 1 and the rest of the time to output 2. There is a delay of "Basic period" in the transmission of the event. If this period > 0 then the second output is initially fired. It is not if this period = 0. In the latter case, the input is driven by an event clock and in the former case, feedback can be used.

DIALOGUE PARAMETERS:

Basic period: positive scalar.
n: an integer greater than 1.

SEE ALSO: MCLOCK_f 610, CLOCK_f 601

2.10.2.48  MIN_f                        Scicos min block

DIALOGUE PARAMETERS:

None.

DESCRIPTION:
The block outputs the minimum of the input vector: \( y = \min(u_1, \ldots, u_n) \). The input vector size is determined by the compiler according to the connected blocks port sizes.

SEE ALSO: MAX_f 609

2.10.2.49  MUX_f                       Scicos multiplexer block

DIALOGUE PARAMETERS:

number of output ports: integer greater than or equal to 1 and less than 8

DESCRIPTION:
Given \( n \) vector valued inputs this block merges inputs in an single output vector. So \( y = [u_1; u_2; \ldots; u_n] \), where \( u_i \) are numbered from top to bottom. Input and Output port sizes are determined by the context.

SEE ALSO: MUX_f 610
2.10.2.50  **NEGTOPOS.f**  
Scicos negative to positive detector

**DESCRIPTION:**
An output event is generated when the unique input crosses zero with a positive slope.

**SEE ALSO:** POSTONEG.f, ZCROSS.f, GENERAL.f

2.10.2.51  **OUT.f**  
Scicos Super Block regular output port

**DIALOGUE PARAMETERS:**

*Port number*: an integer giving the port number.

**DESCRIPTION:**
This block must only be used inside Scicos Super Blocks to represent a regular output port. In a Super Block, regular output ports must be numbered from 1 to the number of regular output ports. The size of the output is determined by the compiler according to the connected blocks port sizes.

**SEE ALSO:** CLKIN.f, IN.f, CLKOUT.f

2.10.2.52  **POSTONEG.f**  
Scicos positive to negative detector

**DESCRIPTION:**
An output event is generated when the unique input crosses zero with a negative slope.

**SEE ALSO:** NEGTOPOS.f, ZCROSS.f, GENERAL.f

2.10.2.53  **POWBLK.f**  
Scicos u^a block

**DIALOGUE PARAMETERS:**

*a*: real scalar

**DESCRIPTION:**
This block realizes \( y(i) = u(i)^a \). The input and output port sizes are determined by the compiler according to the connected blocks port sizes.

2.10.2.54  **PROD.f**  
Scicos element wise product block

**DESCRIPTION:**
The output is the element wize product of the inputs.
2.10.2.55  QUANT  Scicos Quantization block

DIALOGUE PARAMETERS:

Step: scalar, Quantization step
Quantization method: scalar with possible values 1, 2, 3 or 4
1: Round method
2: Truncation method
3: Floor method
4: Ceil method

DESCRIPTION:
This block outputs the quantization of the input according to a choice of methods
for Round method
\[ y(i) = \text{Step} \cdot (\lfloor \frac{u(i)}{\text{Step}} + 0.5 \rfloor - 0.5) \text{ if } u(i) < 0. \]
\[ y(i) = \text{Step} \cdot (\lfloor \frac{u(i)}{\text{Step}} - 0.5 \rfloor + 0.5) \text{ if } u(i) \geq 0. \]
For truncation method
\[ y(i) = \text{Step} \cdot (\lfloor \frac{u(i)}{\text{Step}} + 0.5 \rfloor) \text{ if } u(i) < 0. \]
\[ y(i) = \text{Step} \cdot (\lfloor \frac{u(i)}{\text{Step}} - 0.5 \rfloor) \text{ if } u(i) \geq 0. \]
For floor method
\[ y(i) = \text{Step} \cdot (\lfloor \frac{u(i)}{\text{Step}} + 0.5 \rfloor). \]
For ceil method
\[ y(i) = \text{Step} \cdot (\lfloor \frac{u(i)}{\text{Step}} - 0.5 \rfloor). \]

2.10.2.56  RAND  Scicos random wave generator

DESCRIPTION:
This block is a random wave generator: each output component takes piecewise constant random values.
Every time an event is received on the input event port, the outputs take new independent random values.
output port size is given by the size of A and B vectors

DIALOGUE PARAMETERS:

flag: 0 or 1. 0 for uniform distribution on [A, A+B] and 1 for normal distribution N(A, B*B).
A: scalar
B: scalar

SEE ALSO: GENSIN_f 607, SAWTOOTH_f 614, GENSQR_f 607

2.10.2.57  READC  Scicos "read from C binary file" block

DIALOGUE PARAMETERS:

Time record Selection: an empty matrix or a positive integer. If an integer i is given the i-th element of the read record is assumed to be the date of the output event. If empty no output event exists.
Output record selection: a vector of positive integer. \([k1, \ldots, kn]\), The k-th element of the read record gives the value of the k-th output.
Input file name: a character string defining the path of the file
Input Format: a character string defining the format to use
"l", "s", "ul", "us", "d", "f", "c", "uc" : for reading respectively long, short, unsigned long,
unsigned short, double, float, char and unsigned char. If required by the swap mode, the bytes
which are read are automatically swapped if necessary (by checking little-endian status) in order to
produce machine independent binary files (in little-endian mode).
"ull", "uls", "ubl", "ubs", : can be used for reading respectively unsigned little-endian long or
short and unsigned big-endian long or short.
"dx", "fx", "lx", "sx" : with x=b or x=l can be used for reading double, float, long or short as
big or little endian.

Record size : The file is supposed to be formed by a sequence of data with same format. these data
are organized in a sequence of record each of them containing Record size data.
Buffer size : To improve efficiency it is possible to buffer the input data. read on the file is only done
after each Buffer size call to the block.
Initial record index : a scalar. This fixes the first record of the file to use.
Swap mode : With Swap mode=1 the file is supposed to be coded in "little endian IEEE format" and
data are swapped if necessary to match the IEEE format of the processor. If Swap mode=0 then
automatic bytes swap is disabled.

DESCRIPTION :
This block allows user to read datas in a C file. Output record selection and Time record
Selection allows the user to select data among file records.
Each call to the block advance one record in the file.
SEE ALSO : RFILE_f 613, mget 242

2.10.2.58 REGISTER_f Scicos shift register block

DESCRIPTION :
This block realizes a shift register. At every input event, the register is shifted one step.
DIALOGUE PARAMETERS :
Initial condition : a column vector. It contains the initial state of the register.
SEE ALSO : DELAY_f 603, DELAYV_f 603, EVTDLY_f 605

2.10.2.59 RELAY_f Scicos relay block

DIALOGUE PARAMETERS :
number of inputs : a scalar. Number of regular and event inputs.
initial connected input : an integer. It must be between 1 and the number of inputs.
DESCRIPTION :
This block routes one of the regular inputs to the unique regular output. the choice of which input is to be
routed is done, initially by the "initial connected input" parameter. Then, every time an input event arrives
on the i-th input event port, the i-th regular input port is routed to the regular output.

2.10.2.60 RFILE_f Scicos "read from file" block

DIALOGUE PARAMETERS :
Time record Selection : an empty matrix or a positive integer. If an integer i is given the i-th
element of the read record is assumed to be the date of the output event. If empty no output event
exists.
Output record selection: a vector of positive integer. \([k_1, \ldots, k_n]\). The \(k_i\)th element of the read record gives the value of \(i\)th output.

Input file name: a character string defining the path of the file.

Input Format: a character string defining the Fortran format to use or nothing for an unformatted (binary) write. If given, the format must began by a left parenthesis and end by a right parenthesis. Example: \((e10.3)\).

Buffer size: To improve efficiency it is possible to buffer the input data. Read on the file is only done after each Buffer size call to the block.

Size of output: a scalar. This fixes the number of "value" read.

**DESCRIPTION:**
This block allows user to read datas in a file, in formatted or binary mode. **Output record selection** and **Time record selection** allows the user to select data among file records.

Each call to the block advance one record in the file.

**SEE ALSO:** WFILE_f 618

### 2.10.2.61 SAMPLEHOLD_f Scicos Sample and hold block

**DIALOGUE PARAMETERS:**
None.

**DESCRIPTION:**
Each time an input event is received block copy its input on the output and hold it until input event. For periodic Sample and hold, event input must be generated by a **Clock**.

**SEE ALSO:** DELAY_f 603, CLOCK_f 601

### 2.10.2.62 SAT_f Scicos Saturation block

**DESCRIPTION:**
This block realizes the non-linear function: saturation.

**DIALOGUE PARAMETERS:**
- Min: a scalar. Lower saturation bound
- Max: a scalar. Upper saturation bound
- Slope: a scalar. The slope of the line going through the origin and describing the behaviour of the function around zero.

**SEE ALSO:** LOOKUP_f 609

### 2.10.2.63 SAWTOOTH_f Scicos sawtooth wave generator

**DESCRIPTION:**
This block is a sawtooth wave generator: output is \((t-t_j)\) from \(t_i\) to \(t_{(i+1)}\) where \(t_j\) and \(t_{(i+1)}\) denote the times of two successive input events.

**DIALOGUE PARAMETERS:**
None.

**SEE ALSO:** GENSIN_f 607, GENSQR_f 607, RAND_f 612
**DESCRIPTION:**
This block realizes the visualization of the evolution of the signals on the standard input port(s) at instants of events on the event input port.

**DIALOGUE PARAMETERS:**

- **Curve colors**: a vector of integers. The i-th element is the color number (>0) or dash type (<0) used to draw the evolution of the i-th input port signal. See plot2d for color (dash type) definitions.
- **Output window number**: The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don’t have the same window numbers (unless superposition of the curves is desired).
- **Output window position**: a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.
- **Output window size**: a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.
- **Ymin, Ymax**: Minimum and maximum values of the input; used to set up the Y-axis of the plot in the graphics window.
- **Refresh period**: Maximum value on the X-axis (time). The plot is redrawn when time reaches a multiple of this value.
- **Buffer size**: To improve efficiency it is possible to buffer the input data. The drawing is only done after each Buffer size call to the block.
- **Accept herited events**: if 0 SCOPE_f draws a new point only when an event occurs on its event input port. If 1 SCOPE_f draws a new point when an event occurs on its event input port and when it’s regular input changes due to an event on an other upstream block (herited events).

**REMARKS:**
Output window number, Output window size, Output window position are only taken into account at the initialisation time of the simulation.

**SEE ALSO:** SCOPXY_f 615, EVENTSCOPE_f 605, ANIMXY_f 599

---

**DESCRIPTION:**
This block realizes the visualization of the evolution of the two regular input signals by drawing the second input as a function of the first at instants of events on the event input port.

**DIALOGUE PARAMETERS:**

- **Curve colors**: an integer. It is the color number (>0) or dash type (<0) used to draw the evolution of the input port signal. See plot2d for color (dash type) definitions.
- **Line or mark size**: an integer.
- **Output window number**: The number of graphic window used for the display. It is often good to use high values to avoid conflict with palettes and Super Block windows. If you have more than one scope, make sure they don’t have the same window numbers (unless superposition of the curves is desired).
- **Output window position**: a 2 vector specifying the coordinates of the upper left corner of the graphic window. Answer [] for default window position.
- **Output window size**: a 2 vector specifying the width and height of the graphic window. Answer [] for default window dimensions.
- **Xmin, Xmax**: Minimum and maximum values of the first input; used to set up the X-axis of the plot in the graphics window.
Scicos Block

Ymin, Ymax: Minimum and maximum values of the second input; used to set up the Y-axis of the plot in the graphics window.
Buffer size: To improve efficiency it is possible to buffer the input data. The drawing is only done after each Buffer size call to the block.

REMARKS:
Output window number, Output window size, Output window position are only taken into account at the initialisation time of the simulation.
SEE ALSO: SCOPE_f 615, EVENTSCOPE_f 605, ANIMXY_f 599

2.10.2.66 SELECT_f Scicos selector block

DIALOGUE PARAMETERS:
number of inputs: a scalar. Number of regular and event inputs.
initial connected input: an integer. It must be between 1 and the number of inputs.

DESCRIPTION:
This block routes one of the regular inputs to the unique regular output. The choice of which input is to be routed is done, initially by the “initial connected input” parameter. Then, every time the block is activated through its i-th input activation port, the i-th regular input value port is put to the regular output.

2.10.2.67 SINBLK_f Scicos sine block

DIALOGUE PARAMETERS:
None.

DESCRIPTION:
This block realizes vector sine operation. \( y(i) = \sin(u(i)) \). The input and output port sizes are equal and determined by the context.

2.10.2.68 SOM_f Scicos addition block

DIALOGUE PARAMETERS:
Input signs: a (1x3) vector of +1 and -1. If -1, the corresponding input is multiplied by -1 before addition.

DESCRIPTION:
This block is a sum. The output is the element-wise sum of the inputs.
Input ports are located at up, left or right and down position. You must specify 3 gain numbers but if only two links are connected only the first values are used, ports are numbered anti-clock wise.
SEE ALSO: GAIN_f 606

2.10.2.69 SPLIT_f Scicos regular split block

DIALOGUE PARAMETERS:
None.

DESCRIPTION:
This block is a regular split block with an input and two outputs. The outputs reproduces the input port on each output ports. Strictly speaking, SPLIT is not a Scicos block because it is discarded at the compilation phase. This block is automatically created when creating a new link issued from a link.
Port sizes are determined by the context.
2.10.2.70 STOP \( f \) Scicos Stop block

**DIALOGUE PARAMETERS:**

*State on halt*: scalar. A value to be placed in the state of the block. For debugging purposes this allows to distinguish between different halts.

**DESCRIPTION:**

This block has a unique input event port. Upon the arrival of an event, the simulation is stopped and the main Scicos window is activated. Simulation can be restarted or continued (Run button).

2.10.2.71 SUPER \( f \) Scicos Super block

**DESCRIPTION:**

This block opens up a new Scicos window for editing a new block diagram. This diagram describes the internal functions of the super block. Super block inputs and outputs (regular or event) are designated by special (input or output) blocks. Regular input blocks must be numbered from 1 to the number of regular input ports. Regular input ports of the super block are numbered from the top of the block shape to the bottom. Regular output ports must be numbered from 1 to the number of regular output ports. Regular output ports of the super block are numbered from the top of the block shape to the bottom. Event input blocks must be numbered from 1 to the number of event input ports. Event input ports of the super block are numbered from the left of the block shape to the right. Event output ports must be numbered from 1 to the number of event output ports. Event output ports of the super block are numbered from the left of the block shape to the right.

**SEE ALSO:** CLKIN \( f \) 600, OUT \( f \) 611, CLKOUT \( f \) 600, IN \( f \) 609

2.10.2.72 TANBLK \( f \) Scicos tan block

**DIALOGUE PARAMETERS:**

None.

**DESCRIPTION:**

This block realizes vector tangent operation. input (output) port size is determined by the compiler.

**SEE ALSO:** SINBLK \( f \) 616

2.10.2.73 TCLSS \( f \) Scicos jump continuous-time linear state-space system

**DESCRIPTION:**

This block realizes a continuous-time linear state-space system with the possibility of jumps in the state. The number of inputs to this block is two. The first input is the regular input of the linear system, the second carries the new value of the state which is copied into the state when an event arrives at the unique event input port of this block. That means the state of the system jumps to the value present on the second input (of size equal to that of the state). The system is defined by the \( (A,B,C,D) \) matrices and the initial state \( x0 \). The dimensions must be compatible. The sizes of inputs and outputs are adjusted automatically.

**DIALOGUE PARAMETERS:**

\( A \) : square matrix. The A matrix
\( B \) : the B matrix


C : the C matrix  
D : the D matrix  
x0 : vector. The initial state of the system.

SEE ALSO :  CLSS_f 602,  CLR_f 601

2.10.2.74 TEXT_f Scicos text drawing block

DIALOGUE PARAMETERS :

txt : a character string, Text to be displayed
font : a positive integer less than 6, number of selected font (see xset)
siz : a positive integer, selected font size (see xset)

DESCRIPTION :
This special block is only use to add text at any point of the diagram window. It has no effect on the simulation.

2.10.2.75 TIME_f Scicos time generator

DIALOGUE PARAMETERS :
None.

DESCRIPTION :
This block is a time generator. The unique regular output is the current time.

2.10.2.76 TRASH_f Scicos Trash block

DIALOGUE PARAMETERS :
None

DESCRIPTION :
This block does nothing. It simply allows to safely connect the outputs of other blocks which should be ignored. Useful for sinking outputs of no interest. The input size is determined by the compiler.

2.10.2.77 WFILE_f Scicos "write to file" block

DIALOGUE PARAMETERS :

input size : a scalar. This fixes the input size
Output file name : a character string defining the path of the file
Output Format : a character string defining the Fortran format to use or nothing for an unformatted (binary) write. If given, the format must began by a left parenthesis and end by a right parenthesis. example: (e10.3).
Buffer size : To improve efficiency it is possible to buffer the input data. write on the file is only done after each Buffer size calls to the block.

DESCRIPTION :
This block allows user to save data in a file, in formatted and binary mode. Each call to the block corresponding to a record in the file. Each record has the following form: [t,V1,...,Vn] where t is the value of time when block is called and Vi is the ith input value

SEE ALSO :  RFILE_f 613

Scilab Group  Janvier 1999

618
DIALOGUE PARAMETERS:

Input size: a scalar, the size of the input
Output file name: a character string defining the path of the file
Output Format: a character string defining the format to use
"l", "s", "ul", "us", "d", "f", "c", "uc": for reading respectively long, short, unsigned long,
unsigned short, double, float, char and unsigned char. If required by the swap mode, the bytes
which are read are automatically swapped if necessary (by checking little-endian status) in order to
produce machine independent binary files (in little-endian mode).
"ull", "uls", "ubl", "ubs", : can be used for reading respectively unsigned little-endian long or
short and unsigned big-endian long or short.
"dx", "fx", "lx", "sx": with x=b or x=l can be used for reading double, float, long or short as
big or little endian.
Buffer size: To improve efficiency it is possible to buffer the input data. read on the file is only done
after each Buffer size call to the block.
Swap mode: With Swap mode=1 the file is supposed to be coded in "little endian IEEE format" and
data are swapped if necessary to match the IEEE format of the processor. If Swap mode=0 then
automatic bytes swap is disabled.

DESCRIPTION:
This block allows user to write datas in a C binary file.

See Also: READC_f 612, mput 246

2.10.2.79 ZCROSS_f ________________________ Scicos zero crossing detector

DESCRIPTION:
An output event is generated when all inputs (if more than one) cross zero simultaneously.

DIALOGUE PARAMETERS:

Number of inputs: a positive integer.

See Also: POSTONEG_f 611, GENERAL_f 606

2.10.2.80 scifunc_block _______________________ Scicos block defined interactively

DESCRIPTION:
This block can realize any type of Scicos block. The function of the block is defined interactively using
dialogue boxes and in Scilab language. During simulation, these instructions are interpreted by Scilab;
the simulation of diagrams that include these types of blocks is slower. For more information see Scicos
reference manual.

DIALOGUE PARAMETERS:

number of inputs: a scalar. Number of regular input ports
number of outputs: a scalar. Number of regular output ports
number of input events: a scalar. Number of input event ports
number of output events: a scalar. Number of output event ports
Initial continuous state: a column vector.
Initial discrete state: a column vector.
System type: a string: c or d (CBB or DBB, other types are not supported).
System parameter: column vector. Any parameters used in the block can be defined here a column vector.

Initial firing: vector. Size of this vector corresponds to the number of event outputs. The value of the i-th entry specifies the time of the preprogrammed event firing on the i-th output event port. If less than zero, no event is preprogrammed.

Instructions: other dialogues are opened consecutively where used may input Scilab code associated with the computations needed (block initialization, outputs, continuous and discrete state, output events date, block ending).

See Also: GENERIC_f 607

2.10.3 Data Structures

2.10.3.1 scicos_main __________________________ Scicos editor main data structure

Definition:

```
scs_m=list (params,o_1,...,o_n)
```

Parameters:

- `params`: Scilab list, `params=list(wpar,title,tol,tf,context,void,options,void,void,doc))`
- `wpar`: viewing parameters: `[w,h,Xshift,Yshift]`
- `w`: real scalar, Scicos editor window width
- `h`: real scalar, Scicos editor window height
- `Xshift`: real scalar, diagram drawing x offset within Scicos editor window
- `Yshift`: real scalar, diagram drawing y offset within Scicos editor window
- `title`: character string, diagram title and default name of save file name
- `tol`: 1 x 4 vector `[atol,rtol,ttol,maxt]`, where `atol`, `rtol` are respectively absolute and relative tolerances for the ode solver, `ttol` is the minimal distance between to different events time and `maxt` is maximum integration time interval for a single call to the ode solver.
- `tf`: real scalar, final time for simulation.
- `context`: vector of character strings, Scilab instructions used to define Scilab variables used in block definitions as symbolic parameters.
- `void`: unused fields
- `options`: list(With3D,Color3D)
  - `With3D`: boolean, true for 3D shape blocks
  - `Color3D`: vector with three entries `[R,G,B]` defines the color of 3D shape
- `doc`: user defined diagram documentation structure, default value is `list()`
- `o_i`: block or link or deleted object data structure.

See scicos_block and scicos_link).

Deleted object data structure is marked `list(‘Deleted’).

Description:

Scicos editor uses and modifies the Scicos editor main data structure to keep all information relative to the edited diagram. Scicos compiler uses it as a input.

See Also: scicos 595, scicos_block 621, scicos_link 622

Scicos data structure Scilab Group 620
2.10.3.2  scicos_block  Scicos block data structure

DEFINITION:

blk=list('Block',graphics,model,void,gui)

PARAMETERS:

"Block" : keyword used to define list as a Scicos block representation
graphics : Scilab list, graphic properties data structure
model : Scilab list, system properties data structure.
void : unused, reserved for future use.
gui : character string, the name of the graphic user interface function (generally written in Scilab) associated with the block.
blk : Scilab list, Scicos block data structure

DESCRIPTION:

Scicos editor creates and uses for each block a data structure containing all information relative to the graphic interface and simulation part of the block. Each of them are stored in the Scicos editor main data structure. Index of these in Scicos editor main data structure is given by the creation order.

For Super blocks model(8) contains a data structure similar to the scicos_main data structure.

SEE ALSO:  scicos_graphics 621,  scicos_model 622

2.10.3.3  scicos_graphics  Scicos block graphics data structure

DEFINITION:

graphics=list(orig,sz,flip,exprs,pin,pout,pein,peout,gr_i)

PARAMETERS:

orig : 2 x 1 vector, the coordinate of down-left point of the block shape.
sz : vector [w,h], where w is the width and h the height of the block shape.
flip : boolean, the block orientation. if true the input ports are on the left of the box and output ports are on the right. if false the input ports are on the right of the box and output ports are on the left.
exprs : column vector of strings, contains expressions answered by the user at block set time.
pin : column vector of integers. If pin(k)<>0 then kth input port is connected to the pin(k)<>0 block, else the port is unconnected. If no input port exist pin=[].
pout : column vector of integers. If pout(k)<>0 then kth output port is connected to the pout(k)<>0 block, else the port is unconnected. If no output port exist pout=[].
pein : column vector of ones. If pein(k)<>0 then kth event input port is connected to the pein(k)<>0 block, else the port is unconnected. If no event input port exist pein=[].
peout : column vector of integers. If peout(k)<>0 then kth event output port is connected to the peout(k)<>0 block, else the port is unconnected. If no event output port exist peout=[].
gr_i : column vector of strings, contains Scilab instructions used to customize the block graphical aspect. This field may be set with "Icon" sub-menu.
graphics : Scilab list, Scicos block graphics data structure.

DESCRIPTION:

Scicos block graphics data structure contains all information relative to graphical display of the block and to user dialogue. Fields may be fixed by block definition or set as a result of user dialogue or connections.

SEE ALSO:  scicos 595,  scicos_model 622,  scicos_main 620
### 2.10.3.4 scicos_model

**DEFINITION:**

```plaintext
model=list(sim,in,out,evtin,evtout,state,dstate,..
    rpar,ipar,blocktype,firing,dep_ut,label,import,ID)
```

**PARAMETERS:**

- `sim`: list(fun,typ) or fun. In the latest case typ is supposed to be 0.
- `fun`: character string, the name of the block simulation function (a linked C or Fortran procedure or a Scilab function).
- `typ`: integer, calling sequence type of simulation function (see documentation for more precision).
- `in`: column vector of integers, input port sizes indexed from top to bottom of the block. If no input port exists `in==[]`.
- `out`: column vector of integers, output port sizes indexed from top to bottom of the block. If no output port exists `in==[]`.
- `evtin`: column vector of ones, the size of `evtin` gives the number of event input ports. If no event input port exists `evtin` must be equal to `[]`.
- `evtout`: column vector of ones, the size of `evtout` gives the number of event output ports. If no event output port exists `evtout` must be equal to `[]`.
- `state`: column vector, the initial continuous state of the block. Must be `[]` if no continuous state.
- `dstate`: column vector, the initial discrete state of the block. Must be `[]` if no discrete state.
- `rpar`: column vector, the vector of floating point block parameters. Must be `[]` if no floating point parameters.
- `ipar`: column vector, the vector of integer block parameters. Must be `[]` if no integer parameters.
- `blocktype`: a character with possible values:
  - `'c'`: block output depend continuously of the time.
  - `'d'`: block output changes only on input events.
  - `'z'`: zero crossing block
  - `'l'`: logical block
- `firing`: a vector whose size is equal to the size of `evtout`> It contains output initial event dates (Events generated before any input event arises). Negative values stands for no initial event on the corresponding port.
- `dep_ut`: 1x2 vector of boolean [dep_u, dep_t]. `dep_u` must be true if output depends continuously of the input, `dep_t` must be true if output depends continuously of the time.
- `label`: a character string, used as a label
- `import`: Unused.
- `ID`: a character string, used as an identifier.
- `model`: Scilab list, Scicos block model data structure.

**DESCRIPTION:**

Scicos block model data structure contains all information relative to the simulation functionality of the block. Fields may be fixed by block definition or set.

If block is a super block, the fields `state,dstate,ipar,blocktype,firing,dep_ut,are unused.`

The `rpar` field contains a data structure similar to the `scicos_main` data structure.

**SEE ALSO:** scicos 595, scicos_model 622, scicos_main 620

### 2.10.3.5 scicos_link

**DEFINITION:**

Scicos data structure 

| Scilab Group | 622 |
lnk=list('Link',xx,yy,'drawlink',id,[0,0],ct,from,to)

PARAMETERS:

"Link" : keyword used to define list as a Scicos link representation
xx : vector of x coordinates of the link path.
yy : vector of y coordinates of the link path.
id : Character string, the link id
color : 2 x 1 vector, [color,typ] where color defines the color used for the link drawing and typ defines its type (0 for regular link ,1 for event link).
from : 2 x 1 vector, [block,port] where block is the index of the block at the origin of the link and port is the index of the port.
to : 2 x 1 vector, [block,port] where block is the index of the block at the end of the link and port is the index of the port.

DESCRIPTION:
Scicos editor creates and uses for each link a data structure containing all information relative to the graphic interface and interconnection information. Each of them are stored in the Scicos editor main data structure. Index of these in Scicos editor main data structure is given by the creation order.

SEE ALSO: scicos 595, scicos_main 620, scicos_graphics 621, scicos_model 622

2.10.3.6 scicos_cpr __________________________ Scicos compiled diagram data structure

DEFINITION:

cpr=list(state,sim,cor,corinv)

PARAMETERS:

state : Scilab tlist contains initial state.
state('x') : continuous state vector.
state('z') : discrete state vector.
state('tevts') : vector of event dates
state('evtspt') : vector of event pointers
state('pointi') : pointer to next event state('npoint') : not used yet state('outtb') : vector of inputs/outputs initial values.
sim : Scilab tlist. Usually generated by Scicos Compile menu. Some useful entries are:
sim('rpar') : vector of blocks' floating point parameters
sim('rpptr') : (nblk+1) x 1 vector of integers,
sim('rpar')(rpptr(i):(rpptr(i+1)-1)) is the vector of floating point parameters of the ith block.
sim('ipar') : vector of blocks' integer parameters
sim('ipptr') : (nblk+1) x 1 vector of integers,
sim('ipar')(ipptr(i):(ipptr(i+1)-1)) is the vector of integer parameters of the ith block.
sim('funs') : vector of strings containing the names of each block simulation function
sim('xptr') : (nblk+1) x 1 vector of integers,
state('x')(xptr(i):(xptr(i+1)-1)) is the continuous state vector of the ith block.
sim('zptr') : (nblk+1) x 1 vector of integers,
state('z')(zptr(i):(zptr(i+1)-1)) is the discrete state vector of the ith block.
sim('inpptr') : (nblk+1) x 1 vector of integers,
inpptr(i+1)-inpptr(i) gives the number of input ports. inpptr(i) th points to the beginning of ith block inputs within the indirection table inplnk.
sim('inplnk') : nblink x 1 vector of integers,
    inplnk(inp.ptr(i)-1+j) is the index of the link connected to the jth input port of the ith block. where j goes from 1 to inp.ptr(i+1)-inp.ptr(i).

sim('outptr') : (nblk+1) x 1 vector of integers,
    outptr(i+1)-outptr(i) gives the number of output ports. outptr(i)th points to the beginning of ith block outputs within the indirection table outlnk.

sim('outlnk') : nblink x 1 vector of integers,
    outlnk(outptr(i)-1+j) is the index of the link connected to the jth output port of the ith block. where j goes from 1 to outptr(i+1)-outptr(i).

sim('lnkptr') : (nblink+1) x 1 vector of integers,
    kth entry points to the beginning of region within outtb dedicated to link indexed k.

sim('funs') : vector of strings containing the names of each block simulation function
sim('funtyp') : vector of block block types.
cor : is a list with same recursive structure as scs,m each leaf contains the index of associated block in cpr data structure.
corinv : corinv(i) is the path of ith block defined in cpr data structure in the scs,m data structure.

DESCRIPTION :
Scicos compiled diagram data structure contains all information needed to simulate the system (see scicosim).

SEE ALSO: scicos 595, scicos_model 622, scicos_main 620, scicosim 626

2.10.4 Useful Functions

2.10.4.1 standard_define __________________________ Scicos block initial definition function

CALLING SEQUENCE :

o=standard_define(sz,model,dlg,gr_i)

PARAMETERS :

o : Scicos block data structure (see scicos_block)
sz : 2 vector, giving the initial block width and height
model : initial model data structure definition (see scicos_model)
dlg : vector of character strings, initial parameters expressions
gr_i : vector of character strings, initial icon definition instructions

DESCRIPTION :
    This function creates the initial block data structure given the initial size sz, this initial model definition
    model, the initial parameters expressions dlg and initial icon definition instructions gr_i

SEE ALSO: scicos_model 622

2.10.4.2 standard_draw _____________________________ Scicos block drawing function

CALLING SEQUENCE :

standard_draw(o)

PARAMETERS :

o : Scicos block data structure (see scicos_block)

Scicos function Scilab Group 624
DESCRIPTION:
standard_draw is the Scilab function used to display standard blocks in interfacing functions.
It draws a block with a rectangular shape with any number of regular or event input respectively on the left
and right faces of the block (if not flipped), event input or output respectively on the top and bottom faces
of the block. Number of ports, size, origin, orientation, background color, icon of the block are taken from
the block data structure o.
SEE ALSO: scicos_block 621

2.10.4.3 standard_input ____________________________ get Scicos block input port positions

CALLING SEQUENCE:

[x,y,typ]=standard_input(o)

PARAMETERS:
o : Scicos block data structure (see scicos_block)
x : vector of x coordinates of the block regular and event input ports
y : vector of y coordinates of the block regular and event output ports
 typ : vector of input ports types (+1 : regular port; -1: event port)

DESCRIPTION:
standard_input is the Scilab function used to get standard blocks input port position and types in
interfacing functions.
Port positions are computed, each time they are required, as a function of block dimensions.
SEE ALSO: scicos_block 621

2.10.4.4 standard_origin ____________________________ Scicos block origin function

CALLING SEQUENCE:

[x,y]=standard_draw(o)

PARAMETERS:
o : Scicos block data structure (see scicos_block)
x : x coordinate of the block origin (bottom left corner)
y : y coordinate of the block origin (bottom left corner)

DESCRIPTION:
standard_origin is the Scilab function used to get standard blocks position in interfacing functions.
SEE ALSO: scicos_block 621

2.10.4.5 standard_output ____________________________ get Scicos block output port positions

CALLING SEQUENCE:

[x,y,typ]=standard_output(o)

PARAMETERS:
o : Scicos block data structure (see scicos_block)
curblock

x : vector of x coordinates of the block regular and event output ports
y : vector of y coordinates of the block regular and event output ports
typ : vector of output ports types (+1 : regular port; -1: event port)

DESCRIPTION :
standard_output is the Scilab function used to get standard blocks output port position and types in interfacing functions.
Port positions are computed, each time they are required, as a function of block dimensions.

SEE ALSO: scicos_block 621

2.10.4.6 scicosim ______________________________________ Scicos simulation function

CALLING SEQUENCE :

[state,t]=scicosim(state,0,tf,sim,’start’ [,tol])
[state,t]=scicosim(state,tcur,tf,sim,’run’ [,tol])
[state,t]=scicosim(state,tcur,tf,sim,’finish’ [,tol])

PARAMETERS :

state : Scilab tlist contains scicosim initial state. Usually generated by Scicos Compile or Run menus (see scicos_cpr for more details).
tcur : initial simulation time
tf : final simulation time (Unused with options ’start’ and ’finish’
sim : Scilab tlist. Usually generated by Scicos Compile menu (see scicos_cpr for more details).
tol : 4 vector [atol,rtol,ttol,deltat] where atol,rtol are respectively the absolute and relative tolerances for ode solver (see ode).ttol is the precision on event dates. deltat is maximum integration interval for each call to ode solver.
t : final reached time

DESCRIPTION :
Simulator for Scicos compiled diagram. Usually scicosim is called by scicos to perform simulation of a diagram.
But scicosim may also be called outside Scicos. Typical usage in such a case may be:

1 Use Scicos to define a block diagram, compile it.
2 Save the compiled diagram using Save,SaveAs Scicos menus .
3 In Scilab, load saved file using load function. You get variables scicos_ver,scsm,cpr

scs_m is the diagram Scicos main data structure.
cpr is the data structure list(state,sim,cor,corinv) if the diagram had been compiled before saved, else cpr=list()

4 Extract state,sim out of cpr
5 Execute [state,t]=scicosim(state,0,tf,sim,’start’ [,tolerances]) for initialisation.
6 Execute [state,t]=scicosim(state,0,tf,sim,’run’ [,tolerances]) for simulation from 0 to tf. Many successives such calls may be performed changing initial and final time.
7 Execute [state,t]=scicosim(state,0,tf,sim,’finish’ [,tolerances]) at the very end of the simulation to close files,...

For advanced user it is possible to ”manually” change some parameters or state values

SEE ALSO: scicos 595, scicos_cpr 623

Scicos function Scilab Group 626
2.10.4.7 curblock __________________ get current block index in a Scicos simulation function

CALLING SEQUENCE :

\[ k = \text{curblock}() \]

PARAMETERS :

\( k \): integer, index of the block corresponding to the Scilab simulation function where this function is called.

DESCRIPTION :

During simulation it may be interesting to get the index of the current block to trace execution, to get its label, to animate the block icon according to simulation...

For block with a computational function written in Scilab, Scilab primitive function \texttt{curblock()} allows to get the index of the current block in the compiled data structure.

To obtain path to the block in the Scicos main structure user may uses the \texttt{corinv} table (see \texttt{scicos\_cpr}).

For block with a computational function written in C user may uses the C function \( k = \text{C2F(getcurblock)}() \).

Where \texttt{C2F} is the C compilation macro defined in \texttt{<SCIDIR>/routines/machine.h}

For block with a computational function written in Fortran user may uses the integer function \( k = \text{getcurblock}() \).

SEE ALSO: \texttt{getblocklabel 627, getscicosvars 627, setscicosvars 628, scicos\_cpr 623, scicos\_main 620}

2.10.4.8 getblocklabel ____________________________ get label of a Scicos block at running time

CALLING SEQUENCE :

\[ \text{label} = \text{getblocklabel}() \]
\[ \text{label} = \text{getblocklabel}(k) \]

PARAMETERS :

\( k \): integer, index of the block. if \( k \) is omitted \( k \) is supposed to be equal to \texttt{curblock}().

\texttt{label}: a character string, The label of \( k \)th block (see Label button in Block menu).

DESCRIPTION :

For display or debug purpose it may be usefull to give label to particular blocks of a diagram. This may be done using Scicos editor (Label button in Block menu). During simulation, value of these labels may be obtained in any Scilab block with \texttt{getblocklabel} Scilab primitive function.

For C or fortran computational functions, user may use \texttt{C2F(getlabel)} to get a block label. See \texttt{routines/scicos/import.c} file for more details

Block indexes are those relative to the compile structure\texttt{cpr}.

SEE ALSO: \texttt{curblock 627, getscicosvars 627, setscicosvars 628}

2.10.4.9 getscicosvars ____________________________ get Scicos data structure while running

CALLING SEQUENCE :

\[ v = \text{getscicosvars(name)} \]

PARAMETERS :

\texttt{name}: a character string, the name of the required structure

\texttt{v}: vector of the structure value
setsicosvars

DESCRIPTION:
This function may be used in a Scilab block to get value of some particular global data while running. It allows to write diagram monitoring blocks.

for example the instruction `disp(getscicosvars('x'))` displays the entire continuous state of the diagram.

```scilab
x=getscicosvars('x');
xptr=getscicosvars('xptr');
disp(x(xptr(k):xptr(k+1)-1))
```

displays the continuous state of the \( k \) block

<table>
<thead>
<tr>
<th>name</th>
<th>data structure definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>'x'</td>
<td>continuous state</td>
</tr>
<tr>
<td>'xptr'</td>
<td>continuous state splitting vector</td>
</tr>
<tr>
<td>'z'</td>
<td>discrete state</td>
</tr>
<tr>
<td>'zptr'</td>
<td>discrete state splitting vector</td>
</tr>
<tr>
<td>'rpar'</td>
<td>real parameters vector</td>
</tr>
<tr>
<td>'rpptr'</td>
<td>rpar splitting vector</td>
</tr>
<tr>
<td>'ipar'</td>
<td>integer parameters vector</td>
</tr>
<tr>
<td>'ipptr'</td>
<td>ipar splitting vector</td>
</tr>
<tr>
<td>'outtb'</td>
<td>vector of all input/outputs values</td>
</tr>
<tr>
<td>'inplnk'</td>
<td>inplnk splitting vector</td>
</tr>
<tr>
<td>'outlnk'</td>
<td>outlnk splitting vector</td>
</tr>
<tr>
<td>'lnkptr'</td>
<td>vector of input port values address in lnkpdr</td>
</tr>
<tr>
<td>'linkt'</td>
<td>vector of output port values address in lnkpkr</td>
</tr>
</tbody>
</table>

See `scicos_cpr` for more detail on these data structures.

For C or fortran computational function the C procedure `C2F(getscicosvars)` may used. See `routines/scicos/importfile` for more details.

SEE ALSO: setsicosvars 628, scicosim 626, curblock 627, scicos_cpr 623, getblocklabel 627

### 2.10.4.10 setsicosvars

set Scicos data structure while running

CALLING SEQUENCE:
`setsicosvars(name,v)`

PARAMETERS:
- `name`: a character string, the name of the required structure
- `v`: vector of the new structure value

DESCRIPTION:
This function may be used in a Scilab block to set value of some particular global data while running. It allows to write diagram supervisor blocks.

for example the instructions

```scilab
x=getscicosvars('x');
xptr=getscicosvars('xptr');
x(xptr(k):xptr(k+1)-1)=xk
setsicosvars('x',x)
```

Changes the continuous state of the \( k \) block to \( x_k \).
<table>
<thead>
<tr>
<th>name</th>
<th>data structure definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>'x'</td>
<td>continuous state</td>
</tr>
<tr>
<td>'xptr'</td>
<td>continuous state splitting vector</td>
</tr>
<tr>
<td>'z'</td>
<td>discrete state</td>
</tr>
<tr>
<td>'zptr'</td>
<td>discrete state splitting vector</td>
</tr>
<tr>
<td>'rpar'</td>
<td>real parameters vector</td>
</tr>
<tr>
<td>'rpptr'</td>
<td>rpar splitting vector</td>
</tr>
<tr>
<td>'ipar'</td>
<td>integer parameters vector</td>
</tr>
<tr>
<td>'ipptr'</td>
<td>ipar splitting vector</td>
</tr>
<tr>
<td>'outtb'</td>
<td>vector of all input/outputs values</td>
</tr>
<tr>
<td>'inpptr'</td>
<td>inplnk splitting vector</td>
</tr>
<tr>
<td>'outptr'</td>
<td>outlnk splitting vector</td>
</tr>
<tr>
<td>'inplnk'</td>
<td>vector of input port values address in lnkptr</td>
</tr>
<tr>
<td>'outlnk'</td>
<td>vector of output port values address in lnkpr</td>
</tr>
<tr>
<td>'lnkptr'</td>
<td>outtb splitting vector</td>
</tr>
</tbody>
</table>

See `scicos_cpr` for more detail on these data structures.

For C or fortran computational function the C procedure `C2F(setscicosvars)` may used. See `routines/scicos/import file` for more details.

Warning: The use of this function requires a deep knowledge on how scicosim works, it must be used very carefully. Unpredicted parameters, state, link values changes may produce erroneous simulations.

See Also: `getscicosvars 627, scicosim 626, curblock 627, scicos_cpr 623, getblocklabel 627`
2.11 Sound
2.11.1  analyze  

Calling Sequence:
```
analyze (w [,fmin,fmax,rate,points])
```

Parameters:
fmin, fmax, rate, points: scalars. default values fmin=100, fmax=1500, rate=22050, points=8192;

Description:
Make a frequency plot of the signal \( w \) with sampling rate \( \text{rate} \). The data must be at least \( \text{points} \) long. The maximal frequency plotted will be \( \text{fmax} \), the minimal \( \text{fmin} \).

Example:
```
// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);
// Then we generate the sound.
s=sin(440*t)+sin(220*t)/2+sin(880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(330*t(nc/2:nc));
analyze(s);
```

2.11.2  auread  

Calling Sequence:
```
y=auread(aufile)
y=auread(aufile,ext)
[y,Fs,bits]=auread(aufile)
[y,Fs,bits]=auread(aufile,ext)
```

Parameters:
aufile: string (The .au extension is appended if no extension is given)
Fs:
integer, frequency sampling in Hz.
ext: string ('size' or 'snd') or integer (to read n samples) or 1 x 2 integer vector [n1,n2] (to read from n1 to n2).

Description:
Utility function to read .au sound file. auread(aufile) loads a sound file specified by the string aufile, returning the sampled data in y. Amplitude values are in the range [-1,+1]. Supports multi-channel data in the following formats: 8-bit mu-law, 8-, 16-, and 32-bit linear, and floating point.

[y,Fs,bits]=auread(aufile) returns the sample rate (Fs) in Hertz and the number of bits per sample used to encode the data in the file.
auread(aufile,n) returns the first n samples from each channel.
auread(aufile,[n1,n2]) returns samples n1 to n2.
auread(aufile,'size') returns the size of the audio data contained in the file in place of the actual audio data, returning the vector as [samples channels].
auread(aufile,'snd') returns information about the sample and data as a tlist.

See Also: savewave 634, analyze 631, mapsound 633
2.11.3  auwrite  ----------------------------- writes .au sound file

CALLING SEQUENCE :

auwrite(y,aufile)
auwrite(y,Fs,aufile)
afwrite(y,Fs,bits,aufile)
afwrite(y,Fs,bits,method,aufile)

PARAMETERS :

y : real vector or matrix with entries in [-1,1].
afile : string (The .au extension is appended if no extension is given)
Fs : integer, frequency sampling in Hz.
bits : integer, number of bits in the encoding.
method : string , ’mu’ (default) or ’linear’, encoding method.

DESCRIPTION :
Utility function to save .au sound file. auwrite(y,aufile) writes a sound file specified by the
string aufile. The data should be arranged with one channel per column. Amplitude values outside the
range [-1..1] are ignored. Supports multi-channel data for 8-bit mu-law, and 8- and 16-bit linear formats.
afwrite(y,Fs,aufile) specifies in Fs the sample rate of the data in Hertz.
afwrite(y,Fs,bits,aufile) selects the number of bits in the encoder. Allowable settings are
bits=8 and bits=16.
afwrite(y,Fs,bits,method,aufile) allows selection of the encoding method, which can be either ’mu’ or ’linear’. Note that mu-law files must be 8-bit. By default, method=’mu’.

SEE ALSO : auread 631, wavread 635, savewave 634, analyze 631, mapsound 633

2.11.4  lin2mu  ---------------------- linear signal to mu-law encoding

CALLING SEQUENCE :

mu=lin2mu(y)

PARAMETERS :

y : real vector
mu : real vector

DESCRIPTION :
Utility fct: converts linear signal to mu-law encoding. mu = lin2mu(y) converts linear audio signal
amplitudes in the range -1 <= y <= 1 to mu-law in the range 0 <= mu <= 255.

SEE ALSO: mu2lin 633

2.11.5  loadwave  ---------------- load a sound <<wav>> file into scilab

CALLING SEQUENCE :

x=loadwave(’file-name’);
[x,y]=loadwave(’file-name’);
PARAMETERS:

x : vector
y : vector

DESCRIPTION:
Reads a .wav sound file into Scilab. If y is given, it is filled with information about the sample. (See the message sent by loadwave).

SEE ALSO: savewave 634, analyze 631, mapsound 633

2.11.6 mapsound _________________________________ Plots a sound map

CALLING SEQUENCE:

mapsound (w,dt,fmin,fmax,simpl,rate)

PARAMETERS:
dt,fmin,fmax,simpl,rate: scalars. default values dt=0.1,fmin=100,fmax=1500,simpl=1,rate=22050;

DESCRIPTION:
Plots a sound map for a sound. It does FFT at time increments dt. rate is the sampling rate. simpl points are collected for speed reasons. fmin and fmax are used for graphic boundaries.

EXAMPLE:

// At first we create 0.5 seconds of sound parameters.
t=soundsec(0.5);
// Then we generate the sound.
s=sin(440*t)+sin(220*t)/2+sin(880*t)/2;
[nr,nc]=size(t);
s(nc/2:nc)=sin(330*t(nc/2:nc));
mapsound(s);

2.11.7 mu2lin ___________________________ mu-law encoding to linear signal

CALLING SEQUENCE:

mu=lin2mu(y)

PARAMETERS:
y : real vector
mu : real vector

DESCRIPTION:
Utility fct: y=mu2lin(mu) converts mu-law encoded 8-bit audio signals, stored in the range 0 <= mu <= 255, to linear signal amplitude in the range -s < y < s where s = 32124/32768 ~ 0.9803. The input mu is often obtained using mget(...,'uc') to read byte-encoded audio files. Translation of C program by Craig Reese: IDA/Supercomputing Research Center Joe Campbell: Department of Defense

SEE ALSO: mu2lin 633
2.11.8 playsnd sound player facility

CALLING SEQUENCE:

[] = playsnd(y, fs, bits)

PARAMETERS:

y : real vector
fs : real number, sampling frequency
bits : real number, number of bits (usually 8 or 16)

DESCRIPTION:
Redirects a linear signal to /dev/audio/.

SEE ALSO: lin2mu 632

2.11.9 savewave save data into a sound <<wav>> file.

CALLING SEQUENCE:

savewave('file-name', x [, rate ]);

PARAMETERS:

x : vector
rate : a scalar. 22050 is the default value.

DESCRIPTION:
save x into a wav sound file. you can transform other sound files into wav file with the sox program.

EXAMPLE:

// At first we create 0.5 seconds of sound parameters.
t = soundsec(0.5);
// Then we generate the sound.
s = sin(440*t) + sin(220*t)/2 + sin(880*t)/2;
[nr, nc] = size(t);
s(nc/2:nc) = sin(330*t(nc/2:nc));
savewave(TMPDIR + '/foo.wav', s);

SEE ALSO: loadwave 632, analyze 631, mapsound 633

2.11.10 sound sound player facility

CALLING SEQUENCE:

sound(y)
sound(y, fs)
sound(y, fs, bits)

PARAMETERS:

y : real vector
fs : real number, sampling frequency
wavwrite Scilab Function

bits : real number, number of bits (usually 8 or 16)

DESCRIPTION:
sound(y,fs) sends the signal in vector y (with sample frequency fs) out to the speaker. Values in y are assumed to be in the range -1.0 <= y <= 1.0. Values outside that range are ignored. Stereo sounds are played, on platforms that support it, when y is an N-by-2 matrix. sound(y) plays the sound at the default sample rate of 8192 Hz. sound(y,fs,nbits) plays the sound using nbits bits/sample if possible. Most platforms support bits=8 or 16.

SEE ALSO: playsnd 634

2.11.11 wavread ____________________________ load .wav sound file

CALLING SEQUENCE:
y=wavread(wavfile)
y=wavread(wavfile,ext)
[y,Fs,bits]=wavread(wavfile)
[y,Fs,bits]=wavread(wavfile,ext)

PARAMETERS:
wavfile : string (The .wav extension is appended if no extension is given)
Fs : integer, frequency sampling in Hz.
ext : string ('size') or integer (to read n samples) or 1 x 2 integer vector [n1,n2] (to read from n1 to n2).

DESCRIPTION:
Utility function to read .wav sound file. wavread(wavfile) loads a sound file specified by the string wavfile, returning the sampled data in y. Amplitude values are in the range [-1,+1]. Supports multi-channel data in the following formats: 8-bit mu-law, 8-, 16-, and 32-bit linear, and floating point.
[y,Fs,bits]=wavread(wavfile) returns the sample rate (Fs) in Hertz and the number of bits per sample used to encode the data in the file.
wavread(wavfile,n) returns the first n samples from each channel.
wavread(wavfile,[n1,n2]) returns samples n1 to n2.
read(wavfile,'size') returns the size of the audio data contained in the file in place of the actual audio data, returning the vector as [samples channels].

SEE ALSO: auread 631, savewave 634, analyze 631, mapsound 633

2.11.12 wavwrite ____________________________ writes .wav sound file

CALLING SEQUENCE:
wavwrite(y,wavfile)
wavwrite(y,Fs,wavfile)
wavwrite(y,Fs,bits,wavfile)

PARAMETERS:
y : real vector or matrix with entries in [-1,1].
wavfile : string (The .wav extension is appended if no extension is given)
Fs : integer, frequency sampling in Hz.
bits : integer, number of bits in the encoding.
method : string, ‘mu’ (default) or ‘linear’, encoding method.
DESCRIPTION:
Utility function to save .wav sound file. `wavwrite(y,wavfile)` writes a sound file specified by the string wavfile. The data should be arranged with one channel per column. Amplitude values outside the range [-1,+1] are ignored. Supports multi-channel data for 8-bit mu-law, and 8- and 16-bit linear formats.

`wavwrite(y,Fs,wavfile)` specifies in Fs the sample rate of the data in Hertz.

`wavwrite(y,Fs,bits,wavfile)` selects the number of bits in the encoder. Allowable settings are bits=8 and bits=16.

SEE ALSO: auread 631, wavread 635, savewave 634, analyze 631, mapsound 633
2.12 Cumulative Distribution Functions, Inverses, Random variables
2.12.1 cdfbet__________ cumulative distribution function Beta distribution

CALLING SEQUENCE:

\[[P, Q] = \text{cdfbet}("PQ", X, Y, A, B)\]
\[[X, Y] = \text{cdfbet}("XY", A, B, P, Q)\]
\[[A] = \text{cdfbet}("A", B, P, Q, X, Y)\]
\[[B] = \text{cdfbet}("B", P, Q, X, Y, A)\]

PARAMETERS:

\(P, Q, X, Y, A, B\) : five real vectors of the same size.

\(P, Q \ (Q=1-P)\) : The integral from 0 to X of the beta distribution (Input range: [0, 1]).

\(Q\) : 1-P

\(X, Y \ (Y=1-X)\) : Upper limit of integration of beta density (Input range: [0,1], Search range: [0,1]).

\(A, B\) : The two parameters of the beta density (input range: (0, +infinity), Search range: [1D-300,1D300]).

DESCRIPTION:

Calculates any one parameter of the beta distribution given values for the others (The beta density is proportional to \(t^{(A-1)} \cdot (1-t)^{(B-1)}\). Cumulative distribution function (P) is calculated directly by code associated with the following reference.


Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.


2.12.2 cdfbin__________ cumulative distribution function Binomial distribution

CALLING SEQUENCE:

\[[P, Q] = \text{cdfbin}("PQ", S, Xn, Pr, Ompr)\]
\[[S] = \text{cdfbin}("S", Xn, Pr, Ompr, P, Q)\]
\[[Xn] = \text{cdfbin}("Xn", Pr, Ompr, P, Q, S)\]
\[[Pr, Ompr] = \text{cdfbin}("PrOmpr", P, Q, S, Xn)\]

PARAMETERS:

\(P, Q, S, Xn, Pr, Ompr\) : six real vectors of the same size.

\(P, Q \ (Q=1-P)\) : The cumulation from 0 to S of the binomial distribution. (Probablility of S or fewer successes in XN trials each with probability of success PR.) Input range: [0,1].

\(S\) : The number of successes observed. Input range: [0, XN] Search range: [0, XN].

\(Xn\) : The number of binomial trials. Input range: (0, +infinity). Search range: [1E-300, 1E300].

\(Pr, Ompr \ (Ompr=1-Pr)\) : The probability of success in each binomial trial. Input range: [0,1]. Search range: [0,1].

DESCRIPTION:

Calculates any one parameter of the binomial distribution given values for the others.

Formula 26.5.24 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the binomial distribution to the cumulative incomplete beta distribution.

Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

2.12.3  
cdfchi cumulative distribution function chi-square distribution

CALLING SEQUENCE :

[P,Q]=cdfchi("PQ",X,Df)
[X]=cdfchi("X",Df,P,Q);
[Df]=cdfchi("Df",P,Q,X)

PARAMETERS :

P,Q,Xn,Df : four real vectors of the same size.
P,Q (Q=1-P) : The integral from 0 to X of the chi-square distribution. Input range: [0, 1].
X : Upper limit of integration of the non-central chi-square distribution. Input range: [0, +infinity). Search range: [0,1E300]
Df : Degrees of freedom of the chi-square distribution. Input range: (0, +infinity). Search range: [ 1E-300, 1E300]

DESCRIPTION :

Calculates any one parameter of the chi-square distribution given values for the others.
Formula 26.4.19 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the chi-square distribution to the incomplete distribution.
Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

2.12.4  
cdfchn cumulative distribution function non-central chi-square distribution

CALLING SEQUENCE :

[P,Q]=cdfchn("PQ",X,Df,Pnonc)
[X]=cdfchn("X",Df,Pnonc,P,Q);
[Df]=cdfchn("Df",Pnonc,P,Q,X)

PARAMETERS :

P,Q,Xn,Df,Pnonc : five real vectors of the same size.
P,Q (Q=1-P) : The integral from 0 to X of the non-central chi-square distribution. Input range: [0, 1-1E-16].
X : Upper limit of integration of the non-central chi-square distribution. Input range: [0, +infinity). Search range: [0,1E300]
Df : Degrees of freedom of the non-central chi-square distribution. Input range: (0, +infinity). Search range: [ 1E-300, 1E300]
Pnonc : Non-centrality parameter of the non-central chi-square distribution. Input range: [0, +infinity).
Search range: [0,1E4]

DESCRIPTION :

Calculates any one parameter of the non-central chi-square distribution given values for the others.
Formula 26.4.25 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to compute the cumulative distribution function.
Computation of other parameters involve a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.
The computation time required for this routine is proportional to the noncentrality parameter (PNONC). Very large values of this parameter can consume immense computer resources. This is why the search range is bounded by 10,000.


### 2.12.5 cdff ________________ cumulative distribution function F distribution

**CALLING SEQUENCE:**

```
[P, Q] = cdff("PQ", F, Dfn, Dfd)
[F] = cdff("F", Dfn, Dfd, P, Q);
[Dfn] = cdff("Dfn", Dfd, P, Q, F);
[Dfd] = cdff("Dfd", P, Q, F, Dfn)
```

**PARAMETERS:**

- P, Q : five real vectors of the same size.
- Q (Q=1-P) : The integral from 0 to F of the f-density. Input range: [0,1].
- F : Upper limit of integration of the f-density. Input range: [0, +infinity). Search range: [0,1E300]
- Dfn : Degrees of freedom of the numerator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]
- Dfd : Degrees of freedom of the denominator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]

**DESCRIPTION:**

Calculates any one parameter of the F distribution given values for the others. Formula 26.6.2 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function for the F variate to that of an incomplete beta.

Computation of other parameters involves a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

The value of the cumulative F distribution is not necessarily monotone in either degrees of freedom. There thus may be two values that provide a given CDF value. This routine assumes monotonicity and will find an arbitrary one of the two values.


### 2.12.6 cdfnc ________ cumulative distribution function non-central f-distribution

**CALLING SEQUENCE:**

```
[P, Q] = cdfnc("PQ", F, Dfn, Dfd, Pnonc)
[F] = cdfnc("F", Dfn, Dfd, Pnonc, P, Q);
[Dfn] = cdfnc("Dfn", Dfd, Pnonc, P, Q, F);
[Dfd] = cdfnc("Dfd", Pnonc, P, Q, F, Dfn)
[Pnonc] = cdfnc("Pnonc", P, Q, F, Dfn, Dfd);
```

**PARAMETERS:**

- P, Q, F, Dfn, Dfd, Pnonc : six real vectors of the same size.
- P, Q (Q=1-P) : The integral from 0 to F of the non-central f-density. Input range: [0,1-1E-16].
- F : Upper limit of integration of the non-central f-density. Input range: [0, +infinity). Search range: [0,1E300]

Scilab Group Dec 1997 640
### cdfnbn

**Dfn**: Degrees of freedom of the numerator sum of squares. Input range: (0, +infinity). Search range: [1E-300, 1E300]

**Dfd**: Degrees of freedom of the denominator sum of squares. Must be in range: (0, +infinity). Input range: (0, +infinity). Search range: [1E-300, 1E300]

**Pnonc**: The non-centrality parameter. Input range: [0, infinity). Search range: [0, 1E4]

**DESCRIPTION**: Calculates any one parameter of the Non-central F distribution given values for the others. Formula 26.6.20 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to compute the cumulative distribution function.

Computation of other parameters involves a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

The computation time required for this routine is proportional to the noncentrality parameter (Pnonc). Very large values of this parameter can consume immense computer resources. This is why the search range is bounded by 10,000.

The value of the cumulative noncentral F distribution is not necessarily monotone in either degrees of freedom. Therefore, there may be two values that provide a given CDF value. This routine assumes monotonicity and will find an arbitrary one of the two values.


---

### 2.12.7 cdfgam

**cumulative distribution function gamma distribution**

### CALLING SEQUENCE:

```
[P,Q]=cdfgam("PQ",X,Shape,Scale)
[X]=cdfgam("X",Shape,Scale,P,Q)
[Shape]=cdfgam("Shape",Scale,P,Q,X)
[Scale]=cdfgam("Scale",P,Q,X,Shape)
```

### PARAMETERS:

- **P, Q, X, Shape, Scale**: five real vectors of the same size.
- **P, Q (Q=1-P)**: The integral from 0 to X of the gamma density. Input range: [0,1].
- **X**: The upper limit of integration of the gamma density. Input range: [0, +infinity). Search range: [0,1E300]
- **Shape**: The shape parameter of the gamma density. Input range: (0, +infinity). Search range: [1E-300, 1E300]
- **Scale**: The scale parameter of the gamma density. Input range: (0, +infinity). Search range: (1E-300, 1E300]

### DESCRIPTION:

Calculates any one parameter of the gamma distribution given values for the others. Cumulative distribution function (P) is calculated directly by the code associated with:


Computation of other parameters involves a search for a value that produces the desired value of P. The search relies on the monotonicity of P with the other parameter.

The gamma density is proportional to \( T^{(SHAPE - 1)} \times EXP(- SCALE \times T) \)

2.12.8  cdfbnb — cumulative distribution function negative binomial distribution

CALLING SEQUENCE:

\[ P, Q = cdfnbn("PQ", S, Xn, Pr, Ompr) \]
\[ S = cdfnbn("S", Xn, Pr, Ompr, P, Q) \]
\[ Xn = cdfnbn("Xn", Pr, Ompr, P, Q, S) \]
\[ Pr, Ompr = cdfnbn("PrOmpr", P, Q, S, Xn) \]

PARAMETERS:

- \( P, Q, S, Xn, Pr, Ompr \): six real vectors of the same size.
- \( P, Q (Q = 1 - P) \): The cumulation from 0 to \( S \) of the negative binomial distribution. Input range: \([0, 1]\).
- \( S \): The upper limit of cumulation of the binomial distribution. There are \( F \) or fewer failures before the \( XN \)th success. Input range: \([0, +\infty)\). Search range: \([0, 1E300]\)
- \( Xn \): The number of successes. Input range: \([0, +\infty)\). Search range: \([0, 1E300]\)
- \( Pr \): The probability of success in each binomial trial. Input range: \([0, 1]\). Search range: \([0, 1]\).
- \( Ompr \): \( 1 - PR \) Input range: \([0, 1]\). Search range: \([0, 1]\) \( PR + OMPR = 1.0 \)

DESCRIPTION:

Calculates any one parameter of the negative binomial distribution given values for the others.

The cumulative negative binomial distribution returns the probability that there will be \( F \) or fewer failures before the \( XN \)th success in binomial trials each of which has probability of success \( PR \).

The individual term of the negative binomial is the probability of \( S \) failures before \( XN \) successes and is

\[
\text{Choose}(S, XN+S-1) \times PR^{XN} \times (1-PR)^S
\]

The formula 26.5.26 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce calculation of the cumulative distribution function to that of an incomplete beta.

Computations of other parameters involve a search for a value that produces the desired value of \( P \). The search relies on the monotonicity of \( P \) with the other parameter.


2.12.9  cdfnor — cumulative distribution function normal distribution

CALLING SEQUENCE:

\[ P, Q = cdfnor("PQ", X, Mean, Std) \]
\[ X = cdfnor("X", Mean, Std, P, Q) \]
\[ Mean = cdfnor("Mean", Std, P, Q, X) \]
\[ Std = cdfnor("Std", P, Q, X, Mean) \]

PARAMETERS:

- \( P, Q, X, Mean, Std \): six real vectors of the same size.
- \( P, Q (Q = 1 - P) \): The integral from \(-\infty\) to \( X \) of the normal density. Input range: \((0, 1]\).
- \( X \): Upper limit of integration of the normal-density. Input range: \((-\infty, +\infty)\)
- \( Mean \): The mean of the normal density. Input range: \((-\infty, +\infty)\)
- \( Std \): Standard Deviation of the normal density. Input range: \((0, +\infty)\).

DESCRIPTION:

Calculates any one parameter of the normal distribution given values for the others.

A slightly modified version of ANORM from Cody, W.D. (1993). "ALGORITHM 715: SPECFUN - A Portable FORTRAN Package of Special Function Routines and Test Drivers" acm Transactions on Mathematical Software. 19, 22-32. is used to calculate the cumulative standard normal distribution.

Scilab Group Dec 1997 642
The rational functions from pages 90-95 of Kennedy and Gentle, Statistical Computing, Marcel Dekker, NY, 1980 are used as starting values to Newton’s Iterations which compute the inverse standard normal. Therefore no searches are necessary for any parameter.

For $X < -1.5$, the asymptotic expansion for the normal is used as the starting value in finding the inverse standard normal. This is formula 26.2.12 of Abramowitz and Stegun.

The normal density is proportional to $\exp(-0.5 \times ((X - \text{MEAN})/\text{SD})^2)$


### 2.12.10 cdfpoi cumulative distribution function poisson distribution

**CALLING SEQUENCE:**

$$[P, Q] = \text{cdfpoi}("PQ", S, Xlam)$$

$$[S] = \text{cdfpoi}("S", Xlam, P, Q)$$

$$[Xlam] = \text{cdfpoi}("Xlam", P, Q, S);$$

**PARAMETERS:**

$P, Q, S, Xlam$: four real vectors of the same size.

$P, Q$ ($Q = 1 - P$): The cumulation from 0 to $S$ of the poisson density. Input range: [0,1].

$S$: Upper limit of cumulation of the Poisson. Input range: [0, +infinity). Search range: [0,1E300]

$Xlam$: Mean of the Poisson distribution. Input range: [0, +infinity). Search range: [0,1E300]

**DESCRIPTION:**

Calculates any one parameter of the Poisson distribution given values for the others.

Formula 26.4.21 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function to that of computing a chi-square, hence an incomplete gamma function.

Cumulative distribution function ($P$) is calculated directly. Computation of other parameters involve a search for a value that produces the desired value of $P$. The search relies on the monotonicity of $P$ with the other parameter.


### 2.12.11 cdft cumulative distribution function Student’s T distribution

**CALLING SEQUENCE:**

$$[P, Q] = \text{cdft}("PQ", T, Df)$$

$$[T] = \text{cdft}("T", Df, P, Q)$$

$$[Df] = \text{cdft}("Df", P, Q, T)$$

**PARAMETERS:**

$P, Q, T, Df$: six real vectors of the same size.

$P, Q$ ($Q = 1 - P$): The integral from -infinity to $t$ of the t-density. Input range: (0,1].

$T$: Upper limit of integration of the t-density. Input range: (-infinity, +infinity). Search range: [-1E150, 1E150]

$Df$: Degrees of freedom of the t-distribution. Input range: (0, +infinity). Search range: [1e-300, 1E10]

**DESCRIPTION:**

Calculates any one parameter of the T distribution given values for the others.

Formula 26.5.27 of Abramowitz and Stegun, Handbook of Mathematical Functions (1966) is used to reduce the computation of the cumulative distribution function to that of an incomplete beta.

Computation of other parameters involve a search for a value that produces the desired value of $P$. The search relies on the monotonicity of $P$ with the other parameter.
### CALLING SEQUENCE:

- \( Y = \text{grand}(m,n,\text{'option'}) [,\text{arg1}, \ldots, \text{argn}] \)
- \( Y = \text{grand}(\text{'option'}) \)
- \( Y = \text{grand}(\text{'option'}) [,\text{arg1}, \ldots, \text{argn}] \)

### PARAMETERS:

- `\text{grand('advnst',K)}`: Advances the state of the current generator by \(2^K\) values and resets the initial seed to that value.
- \( Y = \text{grand}(m,n,\text{'bet'},A,B), \quad Y = \text{grand}(x,\text{'bet'},A,B) \): Returns random deviates from the beta distribution with parameters \(A\) and \(B\). The density of the beta is \(x^{(a-1)} \cdot (1-x)^{(b-1)} / B(a,b)\) for \(0 < x < 1\). Method: R. C. H. Cheng Generating Beta Variables with Nonintegral Shape Parameters Communications of the ACM, 21:317-322 (1978) (Algorithms BB and BC)
- \( Y = \text{grand}(m,n,\text{'bin'},N,P), \quad Y = \text{grand}(x,\text{'bin'},N,P) \): Generates random deviates from a binomial distribution whose number of trials is \(N\) and whose probability of an event in each trial is \(P\). \(N\) is the number of trials in the binomial distribution from which a random deviate is to be generated. \(P\) is the probability of an event in each trial of the binomial distribution from which a random deviate is to be generated. (\(0.0 \leq P \leq 1.0\)) Method: This is algorithm BTPE from: Kachitvichyanukul, V. and Schmeiser, B. W. Binomial Random Variate Generation. Communications of the ACM, 31, 2 (February, 1988) 216.
- \( Y = \text{grand}(m,n,\text{'chi'},Df), \quad Y = \text{grand}(x,\text{'chi'},Df) \): Generates random deviates from the distribution of a chisquare with \(DF\) degrees of freedom random variable. Uses relation between chisquare and gamma.
- \( Y = \text{grand}(m,n,\text{'def'}), \quad Y = \text{grand}(x,\text{'def'}) \): Returns random floating point numbers from a uniform distribution over \(0 - 1\) (endpoints of this interval are not returned) using the current generator
- \( Y = \text{grand}(m,n,\text{'exp'},Av), \quad Y = \text{grand}(x,\text{'exp'},Av) \): Generates random deviates from an exponential distribution with mean \(AV\). For details see: Ahrens, J.H. and Dieter, U. Computer Methods for Sampling From the Exponential and Normal Distributions. Comm. ACM, 15,10 (Oct. 1972), 873 - 882.
- \( Y = \text{grand}(m,n,\text{'f'},Dfn,Dfd), \quad Y = \text{grand}(x,\text{'f'},Dfn,Dfd) \): Generates random deviates from the \(F\) (variance ratio) distribution with \(DFN\) degrees of freedom in the numerator and \(DFD\) degrees of freedom in the denominator. Method: Directly generates ratio of chisquare variates
- \( Y = \text{grand}(m,n,\text{'gam'},\text{Shape},\text{Scale}), \quad Y = \text{grand}(x,\text{'gam'},\text{Shape},\text{Scale}) \): Generates random deviates from the gamma distribution whose density is \((\text{Scale}**\text{Shape})/\Gamma(\text{Shape}) \cdot x^{(\text{Shape}-1)} \cdot \exp(-\text{Scale} \cdot x)\) For details see: (Case \(R \geq 1.0\) ) Ahrens, J.H. and Dieter, U. Generating Gamma Variates by a Modified Rejection Technique. Comm. ACM, 25,1 (Jan. 1982), 47 - 54. Algorithm GD (Case \(0.0 < R < 1.0\) ) Ahrens, J.H. and Dieter, U. Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions. Computing, 12 (1974), 223-246/ Adapted algorithm GS.
- \( G = \text{grand('getcgn')} \): Returns in \(G\) the number of the current random number generator (1..32)
- \( Sd = \text{grand('getsd')} \): Returns the value of two integer seeds of the current generator \(Sd=[sd1, sd2]\)
- \( \text{grand('initgn',I)} \): Reinitializes the state of the current generator
- \( I = -1 \): sets the state to its initial seed
- \( I = 0 \): sets the state to its last (previous) seed
- \( I = 1 \): sets the state to a new seed \(2^w\) values from its last seed
- \( Y = \text{grand}(m,n,\text{'lgi'}), \quad Y = \text{grand}(x,\text{'lgi'}) \): Returns random integers following a uniform distribution over \((1, 2147483562)\) using the current generator.
- \( Y = \text{grand}(\text{'mn'},\text{Mean},\text{Cov}) \): Generate \(M\) Multivariate Normal random deviates \(\text{Mean}\) must be a \(Nx1\) matrix and \(\text{Cov}\) a \(NxN\) positive definite matrix \(Y\) is a \(NxM\) matrix.
Y = grand(n, 'markov', P, x0) Generates n successive states of a Markov chain described by the transition matrix P. Initial state is given by x0. If x0 is a matrix of size m = size(x0, '*') then Y is a matrix of size m*n. Y(1,:) is the sample path obtained from initial state x0(1).

Y = grand(M, 'mul', N, P) Generate M observations from the Multinomial distribution. N is the number of categories, P is the vector of probabilities. P(i) is the probability that an event will be classified into category i. Thus, P(i) must be [0,1]. P is of size N-1, (the probability of category N is 1-sum(P). Y(:,i) is an observation from multinomial distribution. All Y(:,i) will be nonnegative and their sum will be N. Y is of size NxM.


Y = grand(m, n, 'nbn', N, P), Y = grand(x, 'nbn', N, P) : Generates random deviates from a negative binomial distribution. N is the required number of events (N > 0). P is The probability of an event during a Bernoulli trial (0.0 < P < 1.0).


Y = grand(m, n, 'nch', Df, Xnon), Y = grand(x, 'nch', Df, Xnon) : Generates random deviates from the distribution of a noncentral chisquare with DF degrees of freedom and noncentrality parameter XNONC. DF is the degrees of freedom of the chisquare (Must be >= 1.0) XNONC the Noncentrality parameter of the chisquare (Must be >= 0.0) Uses fact that noncentral chisquare is the sum of a chisquare deviate with DF-1 degrees of freedom plus the square of a normal deviate with mean XNONC and standard deviation 1.

Y = grand(m, n, 'nf', Dfn, Dfd, Xnon), Y = grand(x, 'nf', Dfn, Dfd, Xnon) : Generates random deviates from the noncentral F (variance ratio) distribution with DFN degrees of freedom in the numerator, and DFD degrees of freedom in the denominator, and noncentrality parameter XNONC. DFN is the numerator degrees of freedom (Must be >= 1.0) DFD is the Denominator degrees of freedom (Must be positive) XNONC is the Noncentrality parameter (Must be nonnegative) Method: Directly generates ratio of noncentral numerator chisquare variate to central denominator chisquare variate.

Y = grand(m, n, 'nor', Av, Sd), Y = grand(x, 'nor', Av, Sd) : Generates random deviates from a normal distribution with mean, AV, and standard deviation, SD. AV is the mean of the normal distribution. SD is the standard deviation of the normal distribution. For details see: Ahrens, J.H. and Dieter, U. Extensions of Forsythe’s Method for Random Sampling from the Normal Distribution. Math. Comput., 27,124 (Oct. 1973), 927 - 937.

Sd = grand('phr2sd', 'string') : Uses a phrase (character string) to generate two seeds for the RGN random number generator. Sd is an integer vector of size 2 Sd=[Sd1, Sd2]

Y = grand(m, n, 'poi', mu), Y = grand(x, 'poi', mu) : Generates random deviates from a Poisson distribution with mean MU. MU is the mean of the Poisson distribution from which random deviates are to be generated (MU >= 0.0). For details see: Ahrens, J.H. and Dieter, U. Computer Generation of Poisson Deviates From Modified Normal Distributions. ACM Trans. Math. Software, 8, 2 (June 1982),163-179.

Mat = grand(M, 'prm', vect) : Generate M random permutation of column vector vect. Mat is of size (size(vect)xM)

grand('setall', ISEED1, ISEED2) : Sets the initial seed of generator 1 to ISEED1 and ISEED2. The initial seeds of the other generators are set accordingly, and all generators states are set to these seeds.

grand('setcgn', G) : Sets the current generator to G. All references to a generator are to the current generator.

grand('setsd', ISEED1, ISEED2) : Resets the initial seed and state of generator g to ISEED1 and ISEED2. The seeds and states of the other generators remain unchanged.

Y = grand(m, n, 'uin', Low, High), Y = grand(x, 'uin', Low, High) : Generates integers uniformly distributed between LOW and HIGH. LOW is the low bound (inclusive) on integer value to be generated. HIGH is the high bound (inclusive) on integer value to be generated. If (HIGH-LOW) > 2,147,483,561 prints error message

Y = grand(m, n, 'unf', Low, High), Y = grand(x, 'unf', Low, High) : Generates reals uniformly
distributed between LOW and HIGH. Low is the low bound (exclusive) on real value to be generated
High is the high bound (exclusive) on real value to be generated

DESCRIPTION :
Interface fo Library of Fortran Routines for Random Number Generation (Barry W. Brown and James
Lovato, Department of Biomathematics, The University of Texas, Houston)
This set of programs contains 32 virtual random number generators. Each generator can provide
1,048,576 blocks of numbers, and each block is of length 1,073,741,824. Any generator can be set to
the beginning or end of the current block or to its starting value. The methods are from the paper cited
immediately below, and most of the code is a transliteration from the Pascal of the paper into Fortran.
P. L'Ecuyer and S. Cote. Implementing a Random Number Package with Splitting Facilities. ACM
Transactions on Mathematical Software 17:1, pp 98-111.
Most users won’t need the sophisticated capabilities of this package, and will desire a single generator.
This single generator (which will have a non-repeating length of 2.3 X 10’18 numbers) is the default. In
order to accommodate this use, the concept of the current generator is added to those of the cited paper;
references to a generator are always to the current generator. The current generator is initially generator
number 1; it can be changed by 'setcgn', and the ordinal number of the current generator can be obtained
from 'getcgn'.
The user of the default can set the initial values of the two integer seeds with 'setall'. If the user does
not set the seeds, the random number generation will use the default values, 1234567890 and 123456789.
The values of the current seeds can be achieved by a call to 'getsd'. Random number may be obtained as
integers ranging from 1 to a large integer by reference to option 'lgi' or as a floating point number between
0 and 1 by a reference to option 'def'. These are the only routines needed by a user desiring a single stream
of random numbers.

CONCEPTS :
A stream of pseudo-random numbers is a sequence, each member of which can be obtained either as an
integer in the range 1..2,147,483,563 or as a floating point number in the range [0..1]. The user is in charge
of which representation is desired.
The method contains an algorithm for generating a stream with a very long period, 2.3 X 10’18. This
stream in partitioned into G (=32) virtual generators. Each virtual generator contains 2’20 (=1,048,576)
blocks of non-overlapping random numbers. Each block is 2’30 (=1,073,741,824) in length.
The state of a generator is determined by two integers called seeds. The seeds can be initialized by the
user; the initial values of the first must lie between 1 and 2,147,483,562, that of the second between 1 and
2,147,483,398. Each time a number is generated, the values of the seeds change. Three values of seeds
are remembered by the generators at all times: the value with which the generator was initialized, the value
at the beginning of the current block, and the value at the beginning of the next block. The seeds of any
generator can be set to any of these three values at any time.
Of the 32 virtual generators, exactly one will be the current generator, i.e., that one will be used to
generate values for 'lgi' and 'def'. Initially, the current generator is set to number one. The current generator
may be changed by calling 'setcgn', and the number of the current generator can be obtained using 'getcgn'.

TEST EXAMPLE :
An example of the need for these capabilities is as follows. Two statistical techniques are being compared
on data of different sizes. The first technique uses bootstrapping and is thought to be as accurate using less
data than the second method which employs only brute force.
For the first method, a data set of size uniformly distributed between 25 and 50 will be generated. Then
the data set of the specified size will be generated and analyzed. The second method will choose a data set
size between 100 and 200, generate the data and analyze it. This process will be repeated 1000 times.
For variance reduction, we want the random numbers used in the two methods to be the same for each of
the 1000 comparisons. But method two will use more random numbers than method one and without
this package, synchronization might be difficult.
With the package, it is a snap. Use generator 1 to obtain the sample size for method one and generator 2
to obtain the data. Then reset the state to the beginning of the current block and do the same for the second
method. This assures that the initial data for method two is that used by method one. When both have
concluded, advance the block for both generators.

Scilab Group
Dec 1997

Scilab Function
**INTERFACE**: 
A random number is obtained either as a random integer between 1 and 2,147,483,562 by using option 'lgi' (large integer) or as a random floating point number between 0 and 1 by using option 'def'.

The seed of the first generator can be set by using option 'setall'; the values of the seeds of the other 31 generators are calculated from this value.

The number of the current generator can be set by using option 'setcgn' The number of the current generator can be obtained by using option 'getcgn'.
2.13 TCL/Tk interface
2.13.1 ScilabEval ______ tcl instruction : Evaluate a string with scilab interpreter

CALLING SEQUENCE :
ScilabEval str

PARAMETERS :
- str : tcl string character Contains the string to evaluate with the current scilab interpreter.

DESCRIPTION :
This function must be called in a tcl/tk script executed from scilab. It allows to associate scilab actions to
tcl/tk widgets (graphic objects). The string str is put in the scilab interpreter buffer which then evaluates it.
This has in general no border effect in the tcl/tk interpreter.

EXAMPLE (TCL/TK SCRIPT) :

//Create a Tcl script using ScilabEval
tcl_script=[
"toplevel .w1"
'button .w1.b -text "Click here to see a new Scilab Graphic Window"
' -command {ScilabEval "xselect()"}
'pack .w1.b ']
mputl(tcl_script,TMPDIR+'/test.tcl')
// Execute the tcl script
TK_EvalFile(TMPDIR+'/test.tcl')

SEE ALSO: TK_EvalFile 649, TK_EvalStr 650, TK_GetVar 651, TK_Setvar ??

AUTHOR: Bertrand Guihenuef

2.13.2 TK_EvalFile ________________________ Reads and evaluate a tcl/tk file

CALLING SEQUENCE :
TK_EvalFile(filename)

PARAMETERS :
- filename : string character Contains the name of the file to read and evaluate.

DESCRIPTION :
With this routine, one can read and evaluate the content of a file containing tcl/tk scripts. This allows to
create powerful tk interfaces.
The filename might be relative or absolute.

ADVANTAGES AND DRAWBACKS OF THIS FUNCTIONALITY :
This routines allows to use directly tcl/tk scripts. This thus allows, for instance to use Interface Builders
such as SpecTcl to design the interface. The interfaces built directly with tcl/tk scripts are much faster than th ones built with the Scilab Graphic Object library provided with tksci (see uicontrol for example).
Indeed, those Objects are warpings around tk graphic widgets. Nevertheless, this way of creating graphic
user interface sould only be used when one aims at adressing directly specific tk/tcl features. There are two
main reasons for this. First of all, there is no simple way to manipulate scilab objects from within a tcl/tk
script. Thus, the interface designer has to write two sets of callbacks routines. One to describe the changes
occurring in the interface when the user acts on the widgets. The second set of call routines will perform the
(pure) scilab reactions to the user actions.
Here is an example: Suppose you design a scrollbar corresponding to a spline tension value. You want the
spline to be displayed in a graphic windows and updated each time the user moves the scrollbar. At the
same time, you want the value of this tension parameter to be displayed within the Interface. You will have
to write a first tcl/tk (callback) function which will be automatically called by the tk scrollbar (`-command` option). This callback function will update the displayed value of the parameter in the interface and will then call the scilab routine (`'ScilabEval' command) to update the graph.

**REMARKS ON THE TCL/TK SCRIPT STYLE:**
Because Scilab manages the tcl/tk events, it creates the root window `.`, this window should not be destroyed nor directly used by your tcl/tk scripts. You should thus always create your own toplevel windows. Moreover, since this module was written at a time when namespaces didn’t exist, some variables defined by scilab tcl/tk scripts could bother your code.

**AUTHOR:** Bertrand Guiheneuf

**EXAMPLE:**

TK_EvalFile(SCI+'/demos/tk/puzzle')

**SEE ALSO:**
ScilabEval, TK_EvalStr, TK_GetVar, TK_Setvar

### 2.13.3 TK_EvalStr

Evaluate a string within the tcl/tk interpreter

**CALLING SEQUENCE:**

TK_EvalStr(str)

**PARAMETERS:**

str : string or vector of strings, contains the tcl/tk instructions

**DESCRIPTION:**
This routine allows to evaluate tcl/tk instructions with the tcl/tk interpreter launched with scilab.

When tcl/tk support is enabled in scilab, you can evaluate tcl/tk expression from scilab interpreter. In fact, scilab launches a slave tcl/tk interpreter. The scilab instruction TK_EvalStr() can be used to evaluate expression without having to write a tcl/tk in a separated file (this is done using TK_EvalFile).

**AUTHOR:** Bertrand Guiheneuf

**EXAMPLE:**

//with one call
TK_EvalStr(['toplevel .foo1'
'    label .foo1.l -text "TK married Scilab !!!"'
'    pack .foo1.l'
'    button .foo1.b -text close -command {destroy .foo1}'
'    pack .foo1.b'])

//step by step (debugging)
TK_EvalStr('toplevel .foo2');
// creates a toplevel TK window.
TK_EvalStr('label .foo2.l -text "TK married Scilab !!!"');
// create a static label
TK_EvalStr('pack .foo2.l');
// pack the label widget. It appears on the screen.
text='button .foo2.b -text close -command {destroy .foo2}';
TK_EvalStr(text);
TK_EvalStr('pack .foo2.b');

**SEE ALSO:** ScilabEval 649, TK_EvalFile 649, TK_GetVar 651, TK_Setvar ??
2.13.4  TK_GetVar  ______________________ Get a tcl/tk variable value

CALLING SEQUENCE :
value=TK_SetVar(varname)

PARAMETERS :
- varname : string character Contains the name of the tcl/tk variable.
- value : string character Contains the value of the tcl/tk variable 'varname'.

DESCRIPTION :
When tcl/tk support is enabled in scilab, this routine can be used to retrieve the value of a tcl/tk variable.

EXAMPLE :
TK_EvalStr('toplevel .tst1');
// creates a toplevel TK window.
TK_EvalStr('entry .tst1.e -textvariable tvar');
// create an editable entry
TK_EvalStr('set tvar foobar');
// set the entry value
TK_EvalStr('pack .tst1.e');
// pack the entry widget. It appears on the screen.
text=TK_GetVar('tvar')
// retrieve the variable value
// change the entry text and repeat the last command ...

SEE ALSO :  ScilabEval 649,  TK_EvalFile 649,  TK_EvalStr 650,  TK_SetVar 651
AUTHOR : Bertrand Guiheneuf

2.13.5  TK_SetVar  ______________________ Set a tcl/tk variable value

CALLING SEQUENCE :
TK_SetVar(varname, value)

PARAMETERS :
- varname : string character Contains the name of the tcl/tk variable to set.
- value : string character Contains the value to set up in the tcl/tk variable

DESCRIPTION :
This routine allows to set a variable within the tcl/tk interpreter. When tcl/tk support is enabled in scilab, this routine can be used to set up the value of a tcl/tk variable. This can be useful to change some value in the tcl/tk without having to build a tcl/tk instruction (and use TK_EvalStr).

EXAMPLE :
TK_EvalStr('toplevel .tst2');
// creates a toplevel TK window.
TK_EvalStr('label .tst2.l -textvariable tvar');
// create a static label
TK_EvalStr('pack .tst2.l');
// pack the label widget. It appears on the screen.
TK_SetVar('tvar','This text has been set directly within scilab');

AUTHOR : Bertrand Guiheneuf

SEE ALSO :  ScilabEval, TK_EvalFile, TK_EvalStr, TK_GetVar

Fractales Group 04 June 1998 651
2.13.6 close

CALLING SEQUENCE:
   close(h)

PARAMETERS:
   o h: integer Handle of the window to close

DESCRIPTION:
   This routine close a tksci figure (toplevel window). If a handle is given, the figure corresponding to this handle is closed. Otherwise, the current (active) figure is closed.

EXAMPLE:
   h=figure();  // creates figure number 1.
   uicontrol( h, 'style','text', ...  
   'string','scilab is great', ...  
   'position',[50 70 100 100], ...  
   'fontsize',15);  // put a clever text in figure 1
   figure();  // create figure 2
   uicontrol( 'style','text', ...  
   'string','Really great', 'position',[50 70 100 100], 'fontsize',15);  // put a text in figure 2
   close();  // close the current graphic window (ie fig. 2)
   close(h);  // close figure 1

SEE ALSO: figure 652, gcf 654

AUTHOR: Bertrand Guiheneuf

2.13.7 figure

CALLING SEQUENCE:
   hn=figure(h, [prop1, value1 ...])

PARAMETERS:
   o h: integer Handle of the window to create. If not specified, the first free handle is used
   o prop[1,2 ...]: character string name of a property to set
   o val[1,2 ...]: scilab object value to give to the corresponding property
   o hn: handle of the newly created window

DESCRIPTION:
   This routine creates a tksci figure (toplevel window). If a handle is given, the figure corresponding to this handle is created. Otherwise, the window is created with the first free handle, that is the lowest integer not already used by a window. The property named 'position' allows to control the geometrical aspect of the control. It is a [1,4] real vector x y w h where the letters stand for the x location of the left bottom corner, the y location of the left bottom corner, the width and the height of the uicontrol.

   One can also set this property by giving a string where the fields are separated by a '|', ie "x|y|w|h".

EXAMPLE:

Fractales Group 04 June 1998 652
h=figure(3);
// creates figure number 1.
uicontrol( h, 'style','text', ... 
' string','This is a figure', ... 
' position',[50 70 100 100], ... 
' fontsize',15);
// put a text in figure 3
figure();
// create figure 1
uicontrol( 'style','text', ... 
' string','Another figure', ... 
' position',[50 70 100 100], ... 
' fontsize',15);
// put a text in figure 1
close();
// close the current graphic window (ie fig. 1)
close(h);
// close figure 3

SEE ALSO: close 652, gcf 654

AUTHOR: Bertrand Guiheneuf

### 2.13.8 findobj __________________ find an object with specified property

**CALLING SEQUENCE:**
h=findobj(prop,value)

**PARAMETERS:**
- prop : string character Name of the property to test.
- value : string character specify the value the tested property should be equal to.
- h : handle of the found object.

**DESCRIPTION:**
This routine is currently used to find objects knowing their 'tag' property. It returns handle of the first found object which property 'prop' is equal to 'value'. If such an object does not exist, the function returns a void matrix.

**EXAMPLE:**

h=figure();
// creates figure number 1.
uicontrol( h, 'style','text', ... 
' string','This is a figure', ... 
' position',[50 70 100 100], ... 
' fontsize',15, ... 
' tag','Alabel');
// put a text in figure 1
lab=findobj('tag','Alabel');
// find the object which 'tag' value is 'Alabel'
disp(’the handle of the label is ’+string(lab));
close();

SEE ALSO: uicontrol 655, uimenu 657, set 655, get 654

AUTHOR: Bertrand Guiheneuf
2.13.9  gcf ---------------------- gets the current active tksci figure

CALLING SEQUENCE :
   h=gcf()

PARAMETERS :
   o  h : handle of the current figure.

DESCRIPTION :
The current figure is the last created (and still existent) figure.

EXAMPLE :
   figure(5);
   figure();
   figure();
   gcf();
   // returns 2
   close(gcf());
   // close figure 2
   gcf();
   // returns 1
   close(1);
   gcf();
   // returns 5
   close(5);

SEE ALSO:  figure 652,  close 652

AUTHOR : Bertrand Guiheneuf

2.13.10  get ---------------------- Retrieve a property value from an User Interface object.

CALLING SEQUENCE :
   val=get(h,prop)

PARAMETERS :
   o  h : integer the handle of the object to retrieve a property
   o  Bprop : character string name of the property
   o  val : scilab object value of the property

DESCRIPTION :
This routine can be used to retrieve a specified property from a GUI object. Property name are character strings like 'style', 'position' .... This routine returns the value associated to the specified property. Obviously, the type of the returned object depends on the property one aims at querying. For example, the 'style' property which represents the kind of Object the UI control is (ie button, label, list, ..... ) will be represented as a string. On the contrary, the 'position' property, which represents the geometrical aspect of the UI control, will be coded as a [1,4] vector.

EXAMPLE :
   h=uicontrol('string', 'Button');
   // Opens a window with a button.
   p=get(h,'position');
   // get the geometric qспект of the button
   disp('Button width: ' + string(p(3)));
// print the width of the button
close();
// close figure

SEE ALSO:  uicontrol 655, uimenu 657, set 655

AUTHOR: Bertrand Guiheneuf

2.13.11  set ____________________ set a property value of a User Interface object.

CALLING SEQUENCE:
get(h,prop,val)

PARAMETERS:
- h : integer the handle of the object which to set a property up
- prop : character string name of the property
- val : scilab object value to give to the property

DESCRIPTION:
This routine can be used to set a GUI object specified property. Property name are character strings like 'style', 'position' ... The type of the value field depends on the property one aims at setting. For example, the 'style' property which represents the kind of Object the UI control is (ie button, label, list, .... ) will be represented as a string. On the contrary, the 'position' property, which represents the geometrical aspect of the UI control, will be coded as a [1,4] vector.

EXAMPLE:

h=uicontrol('string', 'Button');
// Opens a window with a button.
set(h,'position',[ 50 50 100 100]);
// set the geometric aspect of the button
close();
// close figure

SEE ALSO:  uicontrol 655, uimenu 657, get 654

AUTHOR: Bertrand Guiheneuf

2.13.12  uicontrol __________________ create a Graphic User Interface object

CALLING SEQUENCE:

h=uicontrol([prop1,val1] [,prop2, val2] ...)
h=uicontrol(f,[prop1, val1] [,prop2, val2] ...)

PARAMETERS:
- f : integer handle of the figure which will contain the control
- prop{1, 2} : character string name of a property to set
- val{1, 2} : scilab object value to give to the corresponding property
- h : handle of the created object
DESCRIPTION:
this routine creates an object in a figure. If the handle of the figure is given (as the first parameter), the uicontrol is created in this figure. If no handle is given, the uicontrol is created in the current figure (which may be obtained with a call to gcf()). If there is no current figure, then one is created before the creation of the uicontrol. Then when the control is created, the properties given as parameters are set with the corresponding values. It is equivalent to create the uicontrol, and then set its properties with the set() command. Nevertheless, it generally more efficient to set the properties in the call to uicontrol(). This is particularly true concerning the 'style' field. Indeed, the default value for this property is 'pushbutton'. So if you do not set it at creation time, a button will be created, and will be transformed to another uicontrol when you call the get(h,'style', ... ) instruction. Scilab and all the graphic objects communicate through the property mechanism. Thus, to create adapted uicontrol, one has to know the use of the property fields. Those are described under:

PROPERTIES:

BackgroundColor [1,3] real vector or string Background color of the uicontrol. A color is specified as Red, Green and Blue values. Those values are real in [0,1]. The color can be given as a real vector, ie [R,G,B] or a string where each value is separated by a , ie "R|G|B"
callback string String evaluated by the scilab interpreter when an uicontrol is activated. (for example when you click on a button).
fontangle string : {'normal'} | italic | oblique For a control containing some text, this property sets the slant of the font.
fontsize real For a control containing some text, this property sets the size of the font in FontUnits.
fontunits string : {points} | pixels | normalized For a control containing some text, this property sets the units with which the fontsize is specified.
fontweight string : light | {normal} | demi | bold For a control containing some text, this property sets the weight of the used font.
ListboxTop integer For a ListBox, this property tells which item of the list appears on the first line of the visible area of the list.
Max scalar Specifies the largest value the 'value' property can be set to. It has however different meaning on each uicontrol:
  o Check Boxes : Max is the value the 'value' property take when control is checked
  o Siders : Maximum value of the slider
  o List boxes : if (Max-Min)>1 the list allows multiple selection, Otherwise not.
Min scalar Specifies the lowest value the 'value' property can be set to. It has however different meaning on each uicontrol:
  o Check Boxes : Min is the value the 'value' property take when control is unchecked
  o Siders : Minimum value of the slider
  o List boxes : if (Max-Min)>1 the list allows multiple selection, Otherwise not.
Parent integer Handle of the control parent. Changing this property allows to move a control from a figure to another.
Position [1,4] real vector or string This property is used to set or get the geometrical configuration of a control. It is a real; vector : x y w h where the letters stand for the x location of the left bottom corner, the y location of the left bottom corner, the width and the height of the uicontrol. The unit is determined by the 'Unit' property. One can also set this property by giving a string where the fields are separated by a , ie "x|y|w|h".
SliderStep [1,2] real vector or string small big This property represents the step a slider is moved when the user click on the arrow (small step) or on the slide bar (big step).
String string Generally, this property represents the text appearing in a uicontrol. Its exact meaning sometimes depends on the uicontrol style:
  o List Boxes, Popup Menu the value can be a vector of string or a string where the items are separated by a ".
  o Pushbutton A rectangular button generally used to run a callback.
  o Radiobutton A button with to states : on or off.
uimenu Scilab Function

- checkbox a small uicontrol that have to state: on or off
- edit an editable string control
- text a text control (generally static).
- slider a scale control, that is a scrollbar use to set values between in range with the mouse.
- frame a control representing a zone used to group of related controls.
- listbox a control representing a list of item that can be scrolled. The item can be selected with the mouse.
- popupmenu a button which make a menu appear when clicked.

Tag: string this property is generally used to identify the control. It allows to give it a "name". Mainly used in conjonction with findobj().

Units: string: {points} | pixels | normalized Set the units used to specify the 'position' property.

UserData: scilab object this can be used to associate any scilab object to an uicontrol.

Value: Value of the uicontrol. The exact meaning depends on the style of the uicontrol.
- Check boxes, Radio buttons value is set to Max (see above) when on and Min when off.
- List Boxes, Popu Menu value is a vector of indexes corresponding to the index of the selected entry in the list. 1 is the first item of the list.
- Sliders value indicated by the slider bar.

EXAMPLE:

```scilab
f=figure();
// create a figure
h=uicontrol(f,'style','listbox', ... 
  'position', [10 10 150 160]);
// create a listbox
set(h, 'string', "item 1|item 2|item3");
// fill the list
set(h, 'value', [1 3]);
// select item 1 and 3 in the list
close();
// close the figure
f=figure();
// create a figure
h=uicontrol(f,'style','listbox', ... 
  'position', [10 10 150 160]);
// create a listbox
set(h, 'string', "item 1|item 2|item3");
// fill the list
set(h, 'value', [1 3]);
// select (highlight) the item 1 and 3 in the list
close();
// close the figure
```

SEE ALSO: figure 652, set 655, get 654, uimenu 657

AUTHOR: Bertrand Guiheneuf

2.13.13 uimenu Create a menu or a submenu in a figure

CALLING SEQUENCE:

```
h=uimenu(parent,prop1, val1, prop2, val2 ...)
```

PARAMETERS:

- parent: integer Handle of menu's parent
- propi: string character name of a property to set up

Fractales Group 05 June 1998 657
uimenu Scilab Function

- vali : scilab object value to affect to the corresponding property
- h : integer handle of the corresponding menu

DESCRIPTION:
This allows to create menus in a figure. If 'parent' is a figure, then the menu item will be added to the menu bar of the figure. If 'parent' is a menu item, then the new item will be added to the parent item, allowing to create cascaded submenu. The 'callback' property allows to set up the scilab instruction to call when the item is selected by the user. The 'label' property allows to set up the text appearing for the item.

EXAMPLE:

```scilab
f=figure('position', [10 10 300 200]);
// create a figure
m=uimenu(f,'label', 'windows');
// create an item on the menu bar
m1=uimenu(m,'label', 'operations');
m2=uimenu(m,'label', 'quit scilab', 'callback', "exit");
// create two items in the menu "windows"
m11=uimenu(m1,'label', 'new window', 'callback','xselect()');
m12=uimenu(m1,'label', 'clear window', 'callback','xbasc()');
// create a submenu to the item "operations"
close(f);
// close the figure
```

SEE ALSO: figure 652, uicontrol 655, set 655, get 654

AUTHOR: Bertrand Guiheneuf
2.14 Language and data translation tools
2.14.1 ascii ______________________________________ string ascii conversions

CALLING SEQUENCE :

a=ascii(txt)
txt=ascii(a)

PARAMETERS :

txt : character string or matrix of strings.
a : vector of integer ascii codes

DESCRIPTION :

This function convert Scilab string to a vector of ascii code or vector of ascii code to Scilab strings.
If txt is a matrix of string, ascii(txt) is equivalent to ascii(strcat(txt))

SEE ALSO : code2str 280 , str2code 282

2.14.2 excel2sci ______________________________ reads ascii Excel files

CALLING SEQUENCE :

M=excel2sci(fname [,sep])

PARAMETERS :

fname : character string. The file path
sep : character string. Excel separator used, default value is ","
M : matrix of strings

DESCRIPTION :

Given an ascii file created by Excel using "Text and comma" format excel2sci(fname) returns the

See ALSO : read 254 , evstr 35

2.14.3 formatman ______ formats all help files in a directory in ascii, text or html

CALLING SEQUENCE :

formatman(path [,to])

PARAMETERS :

path : character string, giving the path of the directory
to : character string with possible values "ascii", "tex","html"

DESCRIPTION :

This function, not yet fully checked, formats all help files (*.man) of a given directory written in a subset
of TROFF (text formatting language) to ascii tex or html

Known TROFF directives are .TH, .SH, .TP, .RS, .RE, .LP, .IG, .nf, .fi, .TS, .TE, .ft, .IP

Font handling like

\fV, \fR, , , , ]

are ignored

ascii mode generates a .cat file for each .man file and a whatis file
tex mode generates a .tex file for each .man file and a whatis.tex file
html mode generates a .html file for each .man file and a index.html file

See ALSO : help 298, man 305
2.14.4 fun2string _______________ generates ascii definition of a scilab function

CALLING SEQUENCE:

```
txt=fun2string(fun,name)
```

PARAMETERS:

- `fun`: a function type variable
- `name`: a character string, the generated function name
- `txt`: a column vector of strings, the text giving the scilab instructions

DESCRIPTION:

Given a loaded Scilab function pseudo-code fun2string allow to re-generate the the code. Note that as comments are not retained in loaded functions the comments cannot be regenerated. The generated code is indented and beautified.

The mechanism is similar, but simpler than the mfile2sci one. It may be adapted for syntax translations.

EXAMPLE:

```
txt=fun2string(asinh,'foo');
write(%io(2),txt,'(a)')
```

SEE ALSO: getf 272, edit 267, macrovar 274

2.14.5 mfile2sci _______________ Matlab M_file to scilab translation function

CALLING SEQUENCE:

```
mfile2sci(M_file_path [,result_path [,Imode [,Recmode]]])
```

PARAMETERS:

- `M_file_path`: a character string which gives the path of Matlab M_file to translate
- `result_path`: a character string which gives the directory where the result has to be written. Default value is current directory.
- `Imode`: Boolean flag. If true mfile2sci ask user for variable type and sizes when he cannot infer them. Default value : %f
- `Recmode`: Boolean flag, used by translatepaths function. Must be %f to translate a single mfile.

DESCRIPTION:

mfile2sci, is Matlab M-file to Scilab function traduction tools. It tries whenever possible to replace call to Matlab functions by the equivalent scilab primitives and functions.

To translate a Matlab M-file just enter the scilab instruction: mfile2sci(file) where file is a character string giving the path name of the M-file mfile2sci will generate three files in the same directory

```
<function_name>.sci : the scilab equivalent of the m_file
<function_name>.cat : the scilab help file associated to the function
sci_<function_name>.sci : the scilab function required to translate the calls to this Matlab M_file in other Matlab M_files. This function may be improved "by hand". This function only useful for translation not for use of translated functions.
```

Scilab Group

April 1998
Some functions like eye, ones, size, sum,... behave differently according to the dimension of their arguments. When mfile2sci cannot infer dimensions it replaces these function call by a call to an emulation function named mtlb_<function_name>. For efficiency these functions may be replaced by the proper scilab equivalent instructions.

Some other functions like plot, has no straightforward translation in scilab. They are also replaced by an emulation function named mtlb_<function_name>.

When translation may be incorrect or may be improved mfile2sci adds a comment which began by "//!"

REMARKS:
This function is a still under developpement and is delivered as beta test.

Some Matlab4 basic functions are not yet translated. It is quite simple to add it. See <SCIDIR>/macros/m2sci/README for more details.

KNOWN BUGS:
1- : eval function instructions passed as strings are not translated.
2- : most of plot function are not yet translated
3- : if, for, ended by the end of file produce an error, add the closing end’s
4- : Loop variable of for clause is available afterwards if loops terminates normally in matlab; it is cleared in Scilab generated code.
5- : inequality comparison which implies complex numbers produce a run time error such as "undefined variable : %s_2.s". User can define these operation with Matlab meaning with the following function definition:
   ```
   deff('r=%s_1.s(a,b)','r=real(a)<real(b)')
   deff('r=%s_2.s(a,b)','r=real(a)>real(b)')
   deff('r=%s_3.s(a,b)','r=real(a)<=real(b)')
   deff('r=%s_4.s(a,b)','r=real(a)>=real(b)')
   ```
6- : When i is a vector, Matlab allows insertions like a(i)=v for any v. In scilab v must have the same shape as a(i). This produces run time errors "submatrix incorrectly defined". Rewrite them as a(i)=v.'

EXAMPLE:

```matlab
//create a simple m_file
write(TMPDIR+'/rot90.m',
['function B = rot90(A,k)'
 '  [m,n] = size(A);'
 '  if nargin == 1'
   '  k = 1;
   '  else'
   '  k = rem(k,4);
   '  if k < 0'
   '  k = k + 4;
   '  end'
   'if k == 1'
    '  A = A.'';'
    '  B = A(n:-1:1,:);'
    'elseif k == 2'
    '  B = A(m:-1:1,n:-1:1);'
    'elseif k == 3'
    '  B = A(m:-1:1,:);'
    '  B = B.'';'
    'else'
    '  B = A;
    '  end']);
// translate it dor scilab
```

Scilab Group April 1998 662
mtlb_load fname
mtlb_load xxx.yyy
mtlb_load fname -ascii

PARAMETERS:

fname : a file name
xxx.yyy : a file name with extension

DESCRIPTION:

mtlb_load fname loads in scilab all variables stored in file binary fname.mat.
mtlb_load fname -ascii loads in scilab variable stored in ascii file fname, which must contain
a rectangular array of numeric data, arranged in m lines with n values in each line. The result is an
m-by-n matrix named fname.
mtlb_load xxx.yyy reads the ascii file xxx.yyy, which must contain a rectangular array of numeric
data, arranged in m lines with n values in each line. The result is an m-by-n matrix named xxx.
"stdio" value for fname doesn't redirect load from standard input.

SEE ALSO:  mtib_save 663,  save 258,  load 239

2.14.7 mtlb_save __________________ save variables on file with matlab4 format.

CALLING SEQUENCE:

mtlb_save fname
mtlb_save fname X
mtlb_save fname X Y Z
mtlb_save fname X Y Z -ascii
mtlb_save fname X Y Z -ascii -double
mtlb_save fname X Y Z -ascii -double -tabs

PARAMETERS:

fname : a file name
X Y Z : variable names

AUTHOR : Serge Steer, INRIA

2.14.6 mtlb_load __________ load variables from file with matlab4 format.
DESCRIPTION:
mtbl_save save variables on file with matlab4 formats.

mtlb_save fname saves all the current scilab variables which have corresponding matlab type to the binary "MAT-file" named fname.mat. The data may be retrieved with mtlb_load.

mtlb_save fname X saves only variable X.

mtlb_save fname X Y Z saves variables X, Y, and Z.

mtlb_save fname X Y Z -ascii uses 8-digit ASCII form instead of binary.

mtlb_save fname X Y Z -ascii -double uses 16-digit ASCII form.

mtlb_save fname X Y Z -ascii -double -tabs delimits with tabs.

"stdio" value for fname doesn't redirect save to standard output.

SEE ALSO: mtlb_load 663, save 258, load 239

2.14.8 pol2tex ___________________________________ convert polynomial to TeX format

CALLING SEQUENCE:
[y]=pol2tex(x)

PARAMETERS:
x : polynomial
y : list

DESCRIPTION:
Latex source code for the polynomial x. (For use with texprint)

EXAMPLE:
s=poly(0,'s');
p=s^3+2*s-5;
pol2tex(p)

SEE ALSO: texprint 665

2.14.9 sci2for _______________________ scilab function to Fortran routine conversion

CALLING SEQUENCE:
txt=sci2for(fun,nam,vtps [,lvtps])

PARAMETERS:
fun : Scilab function
nam : character string, the name of generated subroutine
vtps : list
lvtps : list
txt : string, text of the subroutine Fortran code

DESCRIPTION:

The elements of the list vtps give the type and dimensions of the input variables of the calling sequence and lvtps optionally gives the type and dimensions of the output variables. This last parameter is usefull if type and/or dimension inference cannot be able to determine the desired values.

These lists are structured as described below:
Scilab Function

vtps(i)=list(typ,row_dim,col_dim)

where:

typ : is a character string giving the type of the variable :
"0" : constant, integer vector or matrix
"1" : constant, double precision vector or matrix
"10" : character string
row_dim : character string (row dimension)
col_dim : character string (column dimension)
txt : Fortran code

Generated code may use routines of scilab libraries and some others whose source code may be found in <SCIDIR>/util/sci2for.f

REMARKS :
This function is just a try. Only simple function may be translated. Many function calls have not yet Fortran equivalent, to add the translation of a new function call you may define a scilab function. whose name is f,<name of function>. see <SCIDIR>/macros/sci2for/f_*_.sci files for examples.
The following keywords :

work, iwork, ierr
i*w  i*iw*
ilbN  (N integer)

may not appear in the function code.

SEE ALSO: function 268

2.14.10 texprint _____________________________ TeX output of Scilab object

CALLING SEQUENCE :

[text]= texprint(a)

PARAMETERS :
a : Scilab object
text : list

DESCRIPTION :
returns the TeX source code of the Scilab variable a. a is a matrix (constant, polynomial, rational) or a linear system(syslin list).

EXAMPLE :

s=poly(0,'s');
texprint([[1/s, s^2]])

SEE ALSO: pol2tex 664, pol2str 494

2.14.11 translatepaths ______ translate a set of Matlab M_file directories to scilab

CALLING SEQUENCE :

translatepaths(dirs_path ,res_path)
PARAMETERS:

dir_path: a character string vector which gives the paths of Matlab M-file directories to translate
res_path: a character string which gives the path of the directory where the scilab functions are written to.

DESCRIPTION:
translatepaths, translate all Matlab M-file contained in a set of directories to Scilab functions. Each function is translated by mfile2sci. Trace of translation informations are stored in a file named "log" in the res_path directory

SEE ALSO: mfile2sci 661

AUTHOR: Serge Steer, INRIA
2.15 Interprocess communication toolbox
"Scilab description"

2.15.1 AdCommunications _______ advanced communication toolbox for parallel programming

DESCRIPTION:
This the beta version of the Advanced Communications Toolbox (ACT).
This toolbox is based on existing libraries, such as
PVM - Parallel Virtual Machine
PBLAS - Message Passing Library dedicated to Matrix
ScaLapack - Parallel linear algebra Library
ACT manage remote executions of softwares and allow efficient exchanges of messages between those softwares. It offers the possibility to exploit numerous machines on a network, as a virtual computer, by creating a distributed group of independent softwares.

SEE ALSO: Example 668

2.15.2 Example __________________________________ just to test the environment

DESCRIPTION:
We are the knights who say ni!

2.15.3 pvm ______ communications with other applications using Parallel Virtual Machine

DESCRIPTION:
PVM is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource.
The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines, or scalar workstations, that may be interconnected by a variety of networks, such as ethernet, FDDI.
Daemon programs (pvmd3) provide communication and process control between computers (see PVM manpage and manual for more details).
Most important functions of the PVM communication library are included in Scilab.

WARNING:
PVM must be installed in your environment before using it in Scilab. PVM scilab have been developped using the version 3.3.7 of the PVM library.

AUTHORS:
PVM have been developped by A. L. Beguelin, J. J. Dongarra, G. A. Geist, W. C. Jiang, R. J. Manchek, B. K. Moore, V. S. Sunderam (see http://www.netlib.org/pvm3)

SEE ALSO: pvm_barrier ??, pvm_mytid 676, pvm_bcast 669, pvm_parent ??, pvm_config 670, pvm_delhosts 671, pvm_recv 677, pvm_exit 672, pvm_send 679, pvm_getinst 673, pvm_spawn 681, pvm_gettid ??, pvm_spawn_independent 681, pvm_gsize 674, pvm_tasks ??, pvm_joingroup 675, pvm_tidtohost 683 pvm_kill, pvm_jvgroup, pvm_start, pvm_halt

Scilab // Group May 1998 668
2.15.4  **pvm_addhosts**  
add hosts to the virtual machine.

**CALLING SEQUENCE:**

```matlab
[infos] = pvm_addhosts(hosts)
```

**PARAMETERS:**

- `hosts`: row of strings, naming the hosts to be added.
- `infos`: row of integer, corresponding to the status for each host.

**DESCRIPTION:**

The `pvm_addhosts` function adds the computers named in `hosts` to the configuration of computers making up the virtual machine. The names should have the same syntax as lines of a `pvmd` hostfile (see man page for `pvmd3`): A hostname followed by options of the form `xx=y`.

The array `infos` can be checked to determine the status for each host. Values less than zero indicate an error, while positive values are TIDs of the new hosts.

The status of hosts can be requested by the application using `pvm_config`. If a host fails it will be automatically deleted from the configuration. Using `pvm_addhosts` a replacement host can be added by the application, however it is the responsibility of the application developer to make his application tolerant of host failure. Another use of this feature would be to add more hosts as they become available, for example on a weekend, or if the application dynamically determines it could use more computational power.

**EXAMPLE:**

```matlab
info = pvm_addhosts(['isostar', 'loop'])
```

**SEE ALSO:** `pvm_delhosts 671`, `pvm_config 670`

2.15.5  **pvm_bcast**  
broacasts a message to all members of a group

**CALLING SEQUENCE:**

```matlab
[info] = pvm_bcast(group, buff, msgtag)
```

**PARAMETERS:**

- `group`: string, string group name of an existing group.
- `buff`: data to be send.
- `msgtag`: integer, message tag supplied by the user.
- `info`: integer,

**DESCRIPTION:**

The `pvm_bcast` function broadcasts a message to all the members of `group`. In PVM 3.2 and later the broadcast message is not sent back to the sender. Any PVM task can call `pvm_bcast()`, it need not be a member of the group. The content of the message can be distinguished by `msgtag`. If `pvm_bcast` is successful, `info` will be 0. If some error occurs then `info` will be < 0.

`pvm_bcast` is asynchronous. Computation on the sending processor resumes as soon as the message is safely on its way to the receiving processors. This is in contrast to synchronous communication, during which computation on the sending processor halts until a matching receive is executed by all the receiving processors.

`pvm_bcast` first determines the tids of the group members by checking a group data base. A multicast is performed to these tids. If the group is changed during a broadcast the change will not be reflected in the broadcast. Multicasting is not supported by most multiprocessor vendors. Typically their native calls only support broadcasting to all the user’s processes on a multiprocessor. Because of this omission, `pvm_bcast` may not be an efficient communication method on some multiprocessors.

**EXAMPLE:**

Scilab // Group  
May 1998  669
info = pvm_bcast( "worker", [12+%i,4,5;3,4+%i,5],10)

SEE ALSO:  pvm_joingroup 675

2.15.6   pvm_bufinfo __________ Returns information about a message buffer.

CALLING SEQUENCE :
[bytes, msgtag, tid, info] = pvm_bufinfo(bufid)

PARAMETERS :
bufid :  integer, specifying a particular message buffer identifier.
bytes :  integer, size of the message in bytes (non very useful inside scilab).
msgtag :  integer, label of the message. Useful when the message was received
         with a wildcard msgtag.
tid :    integer, task ID of the source of the message.
info :   integer, status code returned by the routine. Values less than
         zero indicate an error.

DESCRIPTION :
pvm_bufinfo returns information about the requested message buffer. Typically it is used to determine
facts about the last received message such as its size or source. pvm_bufinfo is especially useful when an
application is able to receive any incoming message, and the action taken depends on the source tid and the
msgtag associated with the message that comes in first. If pvm_bufinfo is successful, info will be 0. If some
error occurs then info will be < 0.

EXAMPLE :
[bytes, msgtag, tid, info] = pvm_info(bufid)

SEE ALSO:  pvm_recv 677

2.15.7   pvm_config ___________________________ sends a message

CALLING SEQUENCE : 
res = pvm_config()

PARAMETERS :
res, list of 7 elements such that:

res(1):  integer returning the number of hosts (pvmds) in the virtual machine.
res(2):  integer returning the number of different data formats being used.
res(3):  integer returning pvmd task ID for host.
res(4):  character string returning name of host.
res(5):  character string returning architecture name of host
res(6): integer returning relative speed of host. Default value is 1000.
res(7): integer status code returned by the routine.

DESCRIPTION:
pvm_config returns information about the present virtual machine. The information returned is similar to that available from the console command conf.
The pvm_config function returns information about the entire virtual machine in one call.
If pvm_config is successful, info will be 0. If some error occurs then info will be < 0.

EXAMPLE:
res = pvm_config()

SEE ALSO: pvm_tasks ??

2.15.8 pvm_delhosts deletes hosts from the virtual machine.

CALLING SEQUENCE:
infos = pvm_delhosts(hosts)

PARAMETERS:
hosts: row of strings, containing the names of the machines to be deleted.
infos: row of integers, contains the status code returned by the routine for the individual hosts.

DESCRIPTION:
pvm_delhosts deletes the computers of hosts from the existing configuration of computers making up the virtual machine. All PVM processes and the pvmd running on these computers are killed as the computer is deleted. The array infos can be checked to determine the status of each host. Values less than zero indicate an error, while zero values indicate a success.
If a host fails, the PVM system will continue to function and will automatically delete this host from the virtual machine. It is the responsibility of the application developer to make his application tolerant of host failure.

EXAMPLE:
info = pvm_delhosts(['isostar','loop'])

SEE ALSO: pvm_addhosts

2.15.9 pvm_error Prints message describing an error returned by a PVM call.

CALLING SEQUENCE:
[errmsg] = pvm_error(err)

PARAMETERS:
err: integer, error code returned by a pvm function.
errmsg: string, the error message corresponding to the error code err.

DESCRIPTION:
pvm_error returns the error message corresponding to an PVM error code returned by a PVM call.

EXAMPLE:
[a] = pvm_error(0)
2.15.10  **pvm_exit**  __________ tells the local pvmd that this process is leaving PVM.

**CALLING SEQUENCE:**

```
info = pvm_exit()
```

**PARAMETERS:**

- `info`: integer

**DESCRIPTION:**

`pvm_exit` tells the local pvmd that this process is leaving PVM. This routine does not kill the process, which can continue to perform tasks just like any other serial process.

`pvm_exit` should be called by all PVM processes before they stop or exit for good. It must be called by processes that were not started with `pvm_spawn`.

**EXAMPLE:**

```
pvm_exit()
```

**SEE ALSO:**  pvm 668

2.15.11  **pvm_sci2f77**  __________ Convert a F77 complex into a complex scalar

**CALLING SEQUENCE:**

```
res = pvm_f772sci(var)
```

**PARAMETERS:**

- `var`: local scilab variable. On return, the variable will be overwritten with the result of conversion operation.
- `res`: if the parameter `var` is not a variable, the result of the conversion is returned in `res`.

**DESCRIPTION:**

`pvm_f772sci` converts all the complex of a scilab variable represented in the F77 way into a scilab representation. May be useful if an application is receiving data from a non scilab application (directly from a C or F77 program for example).

Note that the parameter is passed by address. It means that if the parameter is a variable, this variable will be overwritten with the result of conversion operation. On the other case, if the parameter is not a variable, the result will be returned in `res`.

For example:

```
-->a = [1+%i, 2+2*%i,3+3*%i]
a =
   ! 1. + i 2. + 2.i 3. + 3.i !

-->pvm_f772sci(a)

-->a
a =
   ! 1. + 2.i 3. + i 2. + 3.i !

-->b = pvm_f772sci([1+%i, 2+2*%i,3+3*%i])
b =
   ! 1. + 2.i 3. + i 2. + 3.i !
```

Scilab // Group January 1998 672
2.15.12 **pvm_get_timer** Get the system’s notion of the current time.

**CALLING SEQUENCE:**

```
[time] = pvm_get_timer()
```

**PARAMETERS:**

- `time`: scalar

**DESCRIPTION:**

`pvm_get_timer` returns the time elapsed since the last call of `pvm_get_timer` or the last call of `pvm_set_timer`. The time is expressed in elapsed microseconds. The resolution of the system clock is hardware dependent; the time may be updated continuously or in clock ticks. `timer` will be 0 if some error occurs then `timer` will be -1.

**EXAMPLE:**

```matlab
B = rand(100,100);
A = rand(100,100);
pvm_set_timer();C=A*B;t=pvm_get_timer()
```

**SEE ALSO:** `pvm_set_timer` 680

2.15.13 **pvm_getinst** returns the instance number in a group of a PVM process.

**CALLING SEQUENCE:**

```
[inum] = pvm_getinst(group, tid)
```

**PARAMETERS:**

- `group`: string, string group name of an existing group.
- `tid`: integer, task identifier of a PVM process.
- `inum`: integer, instance number returned by the routine.

**DESCRIPTION:**

`pvm_getinst` takes a group name `group` and a PVM task identifier `tid` and returns the unique instance number that corresponds to the input. It can be called by any task whether in the group or not. If `pvm_getinst` is successful, `inum` will be >=0. If some error occurs then `inum` will be < 0.

**EXAMPLE:**

```matlab
inum = pvm_getinst( "worker", pvm_mytid() )
```

**SEE ALSO:** `pvm_joingroup` 675, `pvm_gettid`??
2.15.14  **pvm_gsize** returns the number of members presently in the named group.

**CALLING SEQUENCE:**

\[nb\] = pvm_gsize(group)

**PARAMETERS:**

- **group**: string, string group name of an existing group.
- **nb**: integer, returning the number of members presently in the group.

**DESCRIPTION:**

*pvm_gsize* returns the size of the group named group. If there is an error nb will be negative.

Since groups can change dynamically in PVM 3.0, this routine only guarantee to return the instantaneous size of a given group.

**EXAMPLE:**

```plaintext```

nb_worker = pvm_gsize( "worker" )
```

**SEE ALSO:**  pvm_joingroup 675

2.15.15  **pvm_halt** stops the PVM daemon

**CALLING SEQUENCE:**

[\[info\]] = pvm_halt()

**PARAMETERS:**

- **info**: integer, status code returned by the routine. Values less than zero indicate an error.

**DESCRIPTION:**

*pvm_halt* kills all PVM tasks, all the remote daemons, and the local daemon. If the master pvmd is killed manually it should be sent a SIGTERM signal to allow it to kill the remote pvmds and clean up various files.

The pvmd can be killed in a manner that leaves the file /tmp/pvmd.uid behind on one or more hosts. Uid is the numeric user ID (from /etc/passwd) of the user. This will prevent PVM from restarting on that host. Deletion of this file will fix this problem:

```
r '({ grep $user /etc/passwd \| ypmatch $user passwd )} awk -F: '{print "/tmp/pvmd.$3"} exit''
```

For example:

```plaintext```

-->pvm_halt()
ans =
   0.
```

```plaintext```

-->pvm_halt()
ans =
   -14.
```

Error -14 means: *pvm_halt()*: Can’t contact local daemon

**SEE ALSO:**  pvm_start 682,  pvm_addhosts 669,  pvm_config 670
2.15.16  **pvm_joingroup**  

enrolls the calling process in a named group.

**CALLING SEQUENCE:**

```
[inum] = pvm_joingroup(group)
```

**PARAMETERS:**

- `group`: string, string group name of an existing group.
- `inum`: integer, instance number returned by the routine.

**DESCRIPTION:**

`pvm_joingroup` enrolls the calling task in the group named `group` and returns the instance number `inum` of this task in this group. If there is an error, `inum` will be negative.

Instance numbers start at 0 and count up. When using groups, a `(group, inum)` pair uniquely identifies a PVM process. This is consistent with the PVM 2.4 naming schemes. If a task leaves a group by calling `pvm_lvgroup` and later rejoins the same group, the task is not guaranteed to get the same instance number. PVM attempts to reuse old instance numbers, so when a task joins a group it will get the lowest available instance number. A task can be a member of multiple groups simultaneously.

**EXAMPLE:**

```
inum = pvm_joingroup( "worker" )
```

**See Also:**  `pvm_lvgroup 676`

2.15.17  **pvm_kill**  

Terminates a specified PVM process.

**CALLING SEQUENCE:**

```
[infos] = pvm_kill(tids)
```

**PARAMETERS:**

- `tids`: row of integer, task identifier of the PVM process to be killed (not yourself).
- `infos`: row of integer, status code returned by the routine. Values less than zero indicate an error.

**DESCRIPTION:**

`pvm_kill` sends a terminate (SIGTERM) signal to the PVM process identified by `tids`. In the case of multiprocessors, the terminate signal is replaced with a host-dependent method for killing a process. If `pvm_kill` is successful,

The array `infos` can be checked to determine the status for each process. Values less than zero indicate an error, while zero values indicate a success.

`pvm_kill` is not designed to kill the calling process. To kill yourself in C call `pvm_exit()` followed by `quit()`.

**EXAMPLE:**

```
info = pvm_kill(262153)
```

**See Also:**  `pvm_exit 672`
2.15.18  

**pvm lvgroup** —— Unenrolls the calling process from a named group.

**CALLING SEQUENCE:**

```
[info] = pvm_lvgroupp(group)
```

**PARAMETERS:**

*group*: string, group name of an existing group.

*info*: integer, status code returned by the routine.

**DESCRIPTION:**

*pvm lvgroup* unenrolls the calling process from the group named group. If there is an error info will be negative.

If a process leaves a group by calling either *pvm lvgroup* or *pvm exit*, and later rejoins the same group, the process may be assigned a new instance number. Old instance numbers are reassigned to processes calling *pvm joingroup*.

**EXAMPLE:**

```plaintext
info = pvm_lvgroupp( "worker" )
```

**SEE ALSO:**  
*pvm joingroup* 675

2.15.19  

**pvm mytid** —— returns the *tid* of the calling process.

**CALLING SEQUENCE:**

```
[tid] = pvm_mytid()
```

**PARAMETERS:**

*tid*: integer, the task identifier of the calling PVM process. Values less than zero indicate an error.

**DESCRIPTION:**

*pvm mytid* enrolls this process into PVM on its first call. It also generates a unique tid if this process was not created by *pvm spawn*. *pvm mytid* returns the tid of the calling process and can be called multiple times in an application.

Any PVM system call (not just *pvm mytid*) will enroll a task in PVM if the task is not enrolled before the call.

The tid is a 32-bit positive integer created by the local pvmd. The 32 bits are divided into fields that encode various information about this process such as its location in the virtual machine (i.e., local pvmd address), the CPU number in the case where the process is on a multiprocessor, and a process ID field. This information is used by PVM and is not expected to be used by applications. Applications should not attempt to predict or interpret the tid with the exception of calling *tidtohost*().

If PVM has not been started before an application calls *pvm mytid* the returned tid will be < 0.

**EXAMPLE:**

```plaintext
tid = pvm_mytid()
```

**SEE ALSO:**  
*pvm tidtohost* 683,  
*pvm parent* ??
2.15.20  pvm_probe  ________________________ Check if message has arrived.

CALLING SEQUENCE :

[bufid] = pvm_probe(tid, msgtag)

PARAMETERS :

tid : integer, task identifier of sending process supplied by the user.
msgtag : integer, message tag supplied by the user. msgtag should be >= 0.
bufid : integer, returning the value of the new active receive buffer identifier. Values less than zero indicate an error.

DESCRIPTION :

pvm_probe checks to see if a message with label msgtag has arrived from tid. If a matching message has arrived pvm_probe returns a buffer identifier in bufid. This bufid can be used in a pvm_bufinfo call to determine information about the message such as its source and length.

If the requested message has not arrived, then pvm_probe returns with a 0 in bufid. If some error occurs bufid will be < 0.

A -1 in msgtag or tid matches anything. This allows the user the following options. If tid = -1 and msgtag is defined by the user, then pvm_probe will accept a message from any process which has a matching msgtag. If msgtag = -1 and tid is defined by the user, then pvm_probe will accept any message that is sent from process tid. If tid = -1 and msgtag = -1, then pvm_probe will accept any message from any process.

pvm_probe can be called multiple times to check if a given message has arrived yet. After the message has arrived, pvm_recv must be called before the message can be unpacked into the user’s memory using the unpack routines.

EXAMPLE :

arrived = pvm_probe( tid, msgtag );
if (arrived >= 0) then [bytes, msgtag, tid, info] = pvm_info(arrived); end

SEE ALSO:  pvm_recv 677,  pvm_info ??

2.15.21  pvm_recv  ________________________ receive a message.

CALLING SEQUENCE :

[buff, info, msgtid, tag] = pvm_recv(tid, msgtag)

PARAMETERS :

tid : integer, task identifier of sending process supplied by the user.
msgtag : integer, message tag supplied by the user. msgtag should be >= 0.
buff : scilab variable, where the received message will be stored.
info : integer, status code returned by the routine. Values less than zero indicate an error.
msgtid : integer, returning the source of the message. Useful when the message was received with a wildcard tid.
tag : integer, returning the message label. Useful when the message was received with a wildcard msgtag.
DESCRIPTION:
pvm_recv blocks the process until a message with label msgtag has arrived from tid. pvm_recv then places the message in buff.

A -1 in msgtag or tid matches anything. This allows the user the following options. If tid = -1 and msgtag is defined by the user, then pvm_recv will accept a message from any process which has a matching msgtag. If msgtag = -1 and tid is defined by the user, then pvm_recv will accept any message that is sent from process tid. If tid = -1 and msgtag = -1, then pvm_recv will accept any message from any process.

When wildcard are used, the application is able to receive any incoming message. If the action taken depends on the source tid and the msgtag associated with the message that comes in first, the information given by msgtid and tag can be very useful.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

info will be the status code returned by the routine. If some error occurs then info will be < 0.

pvm_recv is blocking which means the routine waits until a message matching the user specified tid and msgtag values arrives at the local pvmd. If the message has already arrived then pvm_recv returns immediately with the message.

Once pvm_recv returns, the data in the message can be unpacked into the user’s memory using the unpack routines.

EXAMPLE:

[b, info, msgtid, tag] = pvm_recv(pvm_parent(),100)
g = pvm_recv(pvm_parent(),200)

SEE ALSO: pvm_send 679, pvm_bcast 669

2.15.22 pvm_reduce - Performs a reduce operation over members of the specified group.

CALLING SEQUENCE:

[buff, info] = pvm_reduce(func, buff, msgtag, group, rootginst)

PARAMETERS:

func : string, defines the operation performed on the global data. Should be: Max, Min, Sum or Pro.
buff : scalar, local Scilab variable. On return, the data array on the root will be overwritten with the result of the reduce operation over the group.
msgtag : integer, message tag supplied by the user. msgtag should be > = 0. It allows the user’s program to distinguish between different kinds of messages.
group : string, group name of an existing group.
rootginst : integer, instance number of group member who gets the result.
info : integer, status code returned by the routine. Values less than zero indicate an error.

DESCRIPTION:
pvm_reduce performs global operations such as max, min, sum or product over all the tasks in a group. All group members call pvm_reduce with their local data, and the result of the reduction operation appears on the user specified root task root. The root task is identified by its instance number in the group.

Max and Min are implemented for scalar datatypes (double, complex). For complex values the minimum [maximum] is that complex pair with the minimum [maximum] modulus. Sum and Product are implemented for scalar datatypes.
Note: `pvm_reduce` does not block. If a task calls `pvm_reduce` and then leaves the group before the root has called `pvm_reduce` an error may occur.

**EXAMPLE:**

```plaintext
A = rand(5,5);
[buff, info] = pvm_reduce("Max", A, msgtag, "Workers", 0)
```

**SEE ALSO:** `pvm_bcast` 669, `pvm_barrier` ??, `pvm_psend` ??, `pvm_getinst` 673, `pvm_gsize` 674, ?? `pvm_joingroup`, `pvm_leavegroup`

### 2.15.23 `pvm_sci2f77` Convert complex scalar into F77

**CALLING SEQUENCE:**

```plaintext
[res] = pvm_sci2f77(var)
```

**PARAMETERS:**

- `var`: local Scilab variable. On return, the variable will be overwritten with the result of the conversion operation.
- `res`: if the parameter `var` is not a variable, the result of the conversion is returned in `res`.

**DESCRIPTION:**

`pvm_sci2f77` converts all the complex of a Scilab variable (array, list...) in the F77 representation. May be useful if an application is sending data to a non Scilab application (directly to a C or F77 program for example).

Note that the parameter is passed by address. It means that if the parameter is a variable, this variable will be overwritten with the result of the conversion operation. On the other case, if the parameter is not a variable, the result will be returned in `res`.

For example:

```plaintext
-->a = [1+%i, 2+2*%i, 3+3*%i]
    a =
    1. + i  2. + 2.i  3. + 3.i !

-->pvm_sci2f77(a)

-->a
    a =
    1. + 2.i  1. + 3.i  2. + 3.i !

-->b = pvm_sci2f77([1+%i, 2+2*%i, 3+3*%i])
    b =
    1. + 2.i  1. + 3.i  2. + 3.i !
```

**SEE ALSO:** `pvm_f772sci` ??

### 2.15.24 `pvm_send` immediately sends (or multicast) data.

**CALLING SEQUENCE:**

```plaintext
[info] = pvm_send(tids, buff, msgtag)
```
PARAMETERS:

tids : row of integers, contains the task IDs of the tasks to be sent to.

buff : scilab variable.

msgtag : integer, message tag supplied by the user. msgtag should be >= 0. It allows the user’s program to distinguish between different kinds of messages.

info : integer, status code returned by the routine. Values less than zero indicate an error.

DESCRIPTION:

pvm_send sends (or multicasts) a message to the PVM process identified in the tids array. Note that the message is not sent to the caller even if listed in the array of tids. msgtag is used to label the content of the message. If pvm_send is successful, info will be 0. If some error occurs then info will be < 0.

The pvm_send routine is asynchronous. Computation on the sending processor resumes as soon as the message is safely on its way to the receiving processor. This is in contrast to synchronous communication, during which computation on the sending processor halts until the matching receive is executed by the receiving processor.

If a multicast is performed, pvm_send first determines which other pvmds contain the specified tasks. Then passes the message to these pvmds which in turn distribute the message to their local tasks without further network traffic.

The PVM model guarantees the following about message order. If task 1 sends message A to task 2, then task 1 sends message B to task 2, message A will arrive at task 2 before message B. Moreover, if both messages arrive before task 2 does a receive, then a wildcard receive will always return message A.

Terminating a PVM task immediately after sending a message or messages from it may result in those messages being lost. To be sure, always call pvm_exit() before stopping.

EXAMPLE:

A = rand(5,5)*(1+%i);
def('x = 1/y')
info = pvm_send([262150, 262152], A(1:2:5,:), 100)
pvm_send(262146,f,200)

SEE ALSO: pvm_recv 677, pvm_bcast 669

2.15.25 pvm_set_timer ----------- Sets the system’s notion of the current time.

CALLING SEQUENCE:

[info] = pvm_set_timer()

PARAMETERS:

info : scalar

DESCRIPTION:

pvm_set_timer initialized the timer. info will be 0. If some error occurs then info will be -1.

EXAMPLE:

pvm_set_timer()

SEE ALSO: pvm_get_timer 673
### 2.15.26 `pvm_spawn` 

**Starts new Scilab processes.**

**CALLING SEQUENCE:**

```
[tids, numt] = pvm_spawn(task, ntask, [nw], [where])
```

**PARAMETERS:**

- `task`: string, which is the file name of the scilab script (see exec) to be started. The Scilab script must already reside on the host on which it is to be started. The name must be an absolute path.
- `ntask`: integer, specifying the number of copies of the scilab script to start.
- `win`: string (optional). If `win` is equal to “nw” the Scilab process will be spawned in background without any window coming up.
- `where`: string (optional), can be a host name such as “tequila.ens-lyon.fr” or a PVM architecture class such as “SUN4”.
- `numt`: integer, the actual number of tasks started. Values less than zero indicate a system error. `tids`: row of integers, array of the tids of the PVM processes started by this `pvm_spawn` call.

**DESCRIPTION:**

`pvm_spawn` starts `ntask` copies of the scilab script `task`. On systems that support environment, `spawn` passes selected variables from parent environment to children tasks. If set, the envar `PVM_EXPORT` is passed. If `PVM_EXPORT` contains other names (separated by ‘:’) they will be passed too. This is useful for e.g.:

```
setenv DISPLAY myworkstation:0.0
setenv MYSTERYVAR 13
setenv PVM_EXPORT DISPLAY:MYSTERYVAR
```

The hosts on which the PVM processes are started are determined by the `where` arguments. On return the array `tids` contains the PVM task identifiers for each process started.

If `pvm_spawn` starts one or more tasks, `numt` will be the actual number of tasks started. If a system error occurs then `numt` will be < 0. If `numt` is less than `ntask` then some executables have failed to start and the user should check the last `ntask - numt` locations in the `tids` array which will contain error codes (see below for meaning). The first `numt` tids in the array are always valid.

When the argument where is omitted a heuristic (round-robin assignment) is used to distribute the `ntask` processes across the virtual machine.

In the special case where a multiprocessor is specified by `where`, `pvm_spawn` will start all `ntask` copies on this single machine using the vendor’s underlying routines.

**EXAMPLE:**

```plaintext
// create an exec file (script)
write(TMPDIR+'/foo.sce',['a=1';'plot2d();'])
// start a new Scilab on the same host to execute the script
[tids, numt] = pvm_spawn(TMPDIR+'/foo.sce',1)
pvm_kill(tids) //terminate the new scilab
```

**SEE ALSO:** `pvm` 668, `pvm_spawn_independent` 681

### 2.15.27 `pvm_spawn_independent` 

**Starts new PVM processes.**

**CALLING SEQUENCE:**

```
Scilab // Group January 1998 681
```
[tids, numt] = pvm_spawn_independent(task, ntask, [where])

PARAMETERS:

- **task**: string, which is the executable file name of the PVM process to be started. The executable must already reside on the host on which it is to be started. The name may be a file in the PVM search path or an absolute path. The default PVM search path is $HOME/pvm3/bin/SPVM_ARCH/.
- **ntask**: integer, specifying the number of copies of the executable file to start.
- **where**: string (optional), can be a host name such as “tequila.ens-lyon.fr” or a PVM architecture class such as “SUN4”.
- **numt**: integer, the actual number of tasks started. Values less than zero indicate a system error.
- **tids**: row of integers, array of the tids of the PVM processes started by this pvm_spawn_independent call.

DESCRIPTION:

*pvm_spawn_independent* starts *ntask* copies of the executable named *task*. On systems that support environment, *spawn* passes selected variables from parent environment to children tasks. If set, the envar PVM_EXPORT is passed. If PVM_EXPORT contains other names (separated by ':') they will be passed too. This is useful for e.g.:

```plaintext
setenv DISPLAY myworkstation:0.0
setenv MYSTERYVAR 13
setenv PVM_EXPORT DISPLAY:MYSTERYVAR
```

The hosts on which the PVM processes are started are determined by the *where* arguments. On return the array tids contains the PVM task identifiers for each process started.

If *pvm_spawn_independent* starts one or more tasks, *numt* will be the actual number of tasks started. If a system error occurs then *numt* will be < 0. If *numt* is less than *ntask* then some executables have failed to start and the user should check the last *ntask* - *numt* locations in the tids array which will contain error codes (see below for meaning). The first *numt* tids in the array are always valid.

When the argument *where* is omitted a heuristic (round-robin assignment) is used to distribute the *ntask* processes across the virtual machine.

In the special case where a multiprocessor is specified by *where*, *pvm_spawn_independent* will start all *ntask* copies on this single machine using the vendor’s underlying routines.

EXAMPLE:

```plaintext
[tids, numt] = pvm_spawn_independent("a.out",2)
```

SEE ALSO:  pvm 668,  pvm_spawn 681

2.15.28    **pvm_start**------------------------------------- Start the PVM daemon

CALLING SEQUENCE:

```plaintext
[info] = pvm_start(["hostfile"])
```

PARAMETERS:

- **hostfile**: name of the hostfile describing the configuration for each host of the virtual machine.
- **info**: integer, status code returned by the routine. Values less than zero indicate an error.

DESCRIPTION:

*pvm_start* starts the Pvmd3 which is a daemon process which coordinates unix hosts in a virtual machine. One pvmd3 must run on each host in the group. They provide the communication and process control functions needed by the user’s PVM processes. The local and remote pvmds can also be started from the PVM console program pvm.
The optional parameter is the name of a host file. See pvmd3 for more details on the host file format. If no argument is given to pvm_start, but the variable PVM_ROOT is set, scilab will try to load the file $HOME/pvmd.conf. If this file does not exist, or the variable PVM_ROOT is not set, scilab will try to load the default file $SCI/pvmd.conf. In all other cases, scilab will supposed that PVM and scilab are in standard place on your net.

Note that, to be able to start a PVM daemon, scilex must know the place to find both scilex and pvmd. Normally, scilex will start a new PVM daemon by using rsh. See the help on pvmd3 and pvm for more detail on how to start/stop pvm.

For example:

```scilab
cvm_start()
ans = 0.
-->pvm_start()
an = -28.
```

Error -28 means: pvm_start_pvmd(): Duplicate host

See Also: pvmd3 683, pvm_halt 674, pvm_addhosts 669, pvm_config 670

2.15.29 pvm_tidtohost ---------- returns the host of the specified PVM process.

Calling Sequence:

```
[dtid] = pvm_tidtohost(tid)
```

Parameters:

- `tid`: integer, task identifier of the PVM process in question.
- `dtid`: integer, the tid of the host’s pvmd3 or a negative value if an error.

Description:

`pvm_tidtohost` returns the host id on which the process identified by tid is located.

Example:

```
dtid = pvm_tidtohost(pvm_mytid())
```

See Also: pvm_config 670, pvm_tasks ??

2.15.30 pvmd3 ------------------------------------------ PVM daemon

Synopsis:

```
pvmd3 [ -options ] [ hostfile ] &
```

Description:

`pvmd3` is a daemon process which coordinates hosts in a virtual machine. One pvmd must run on each host in the group. They provide the communication and process control functions needed by the user’s PVM processes. The daemon can be started manually with a host file argument that will automatically start the remote pvmds. The local and remote pvmds can also be started from the PVM console program pvm.

The name of the daemon executable is pvmd3. It is usually started by a shell script, $PVM_ROOT/lib/pvmd.
Local daemon may also be started by the scilab instruction `pvm.start()` and remote daemons may also be started by the scilab function `pvm.addhosts`

**OPTIONS:**
The following options may be specified on the command line when starting the master pvmd or PVM console:

- `-dmask` Set pvmd debug mask. Used to debug the pvmd or libpvm (not intended to be used to debug application programs). Mask is a hexadecimal number which is the sum of the following bits: Bit Information
  1   Packet routing
  2   Message routing and entry points
  4   Task management
  8   Slave pvmd startup
 10   Host table updates
 20   Select loop (below packet layer)
 40   IP network
 80   Multiprocessor port debugging
100  Resource manager interface
200  Application (messages with no destination, etc.)
- `-nname` Specify an alternate hostname for the master pvmd to use. Useful when gethostname() returns a name not assigned to any network interface.

The following options are used by the master pvmd when starting slaves and are only of interest to someone writing a hoster. Don’t just go using them, now.

- `-s` Start pvmd in slave mode. Hostfile cannot be used, five additional parameters must be supplied: master pvmd index, master IP, master MTU, slave pvmd index, and slave IP.
- `-S` Same as -s, but slave pvmd doesn’t wait for its stdin to be closed after printing its parameters. Used for manual startup.
- `-f` Slave doesn’t fork after configuration (useful if the slave is to be controlled or monitored by some process).

Lines beginning with a splat ( # ), optionally preceded by whitespace, are ignored.

A simple host file might look like:

```plaintext
# my first host file
thud
fred
wilma
barney
betty
```

This specifies the names of five hosts to be configured in the virtual machine.
The master pvmd for a group is started by hand on the localhost, and it starts slaves on each of the remaining hosts using the rsh or rexec command. The master host may appear on any line of the hostfile, but must have an entry.
The simple format above works fine if you have the same login name on all five machines and the name of the master host in your .rhosts files on the other four.

There are several host file options available:

- `-lo=NAME` Specifies an alternate login name (NAME) to use.
- `-so=pw` This is necessary when the remote host cannot trust the master. Causes the master pvmd to prompt for a password for the remote host in the tty of the pvmd (note you can’t start the master using the console or background it when using this option) you will see: Password (honk.cs.utk.edu:manchek): you should type your password for the remote host. The startup will then continue as normal.
- `-dx=FILE` Specifies the path of the pvmd executable. FILE may be a simple filename, an absolute path-name, or a path relative to the user’s home directory on the remote host. This is mainly useful to aid in debugging new versions of PVM, but may have other uses.
ep=PATH  Specifies a path for the pvmd to search for executable program components when spawning a new process. The path may have multiple elements, separated by colons (:).

wd=PATH  Specifies a working directory in which all spawned tasks on this host will execute.

sp=VALUE  Specifies the relative computational speed of this host compared to other hosts in the configuration. VALUE is an integer in the range [1 - 1000000].

bx=PATH  Specifies the debugger program path. Note: the environment variable PVM_DEBUGGER can also be set.

so=ms  Rarely used. Causes the master pvmd to request user to manually perform the startup of a pvmd on a slave host when rsh and rexec network services are disabled but IP connectivity exists. See section "MANUAL STARTUP".

A dollar sign ($) in an option introduces a variable name, for example $PVM_ARCH. Names are expanded from environment variables by each pvmd.

Each of the flags above has a default value. These are:

- lo  The loginname on the master host.
- so  Nothing
- dx  $PVM_ROOT/lib/pvmd (or environment variable PVM_DPATH)
- ep  pvm3/bin/$PVM_ARCH:$PVM_ROOT/bin/$PVM_ARCH
- wd  $HOME
- sp  1000
- bx  $PVM_ROOT/lib/debugger

You can change these by adding a line with a star (*) in the first field followed by the options, for example:

* lo=afriend so=pw  This sets new default values for ‘lo’ and ‘so’ for the remainder of the host file, or until the next ‘*’ line. Options set on the last ‘*’ line also apply to hosts added dynamically using pvm_addhosts().

Host options can be set without starting the hosts automatically. Information on host file lines beginning with ‘&’ is stored, but the hosts are not started until added using pvm_addhosts().

Example hostfile:

```
# hostfile for testing on various platforms fonebone
refuge
# installed in /usr/local/here

sigi.cs
  dx=/usr/local/pvm3/lib/pvmd  # borrowed accts, "guest", don’t trust fonebone

* lo=guest so=pw sn666.jrandom.com cubie.misc.edu  # really painful one, must start it by hand and share a homedir & igor.firewall.com lo=guest2 so=ms ep=bob/pvm3/bin/$PVM_ARCH
```

MANUAL STARTUP:

When adding a host with this option set you will see on the tty of the pvmd:

```
*** Manual startup ***

Login to "honk" and type:

$PVM_ROOT/lib/pvmd -S -d0 -nhonk 1 80a9ca95:0cb6 4096 2 80a95c43:0000 Type response:
```

after typing the given command on host honk, you should see a line like:

```
ddpro<2312> arch<ALPHA> ip<80a95c43:0a8e> mtu<4096>
```

type this line on the tty of the master pvmd. You should then see:
Thanks

and the two pvmds should be able to communicate.

Note you can’t start the master using the console or background it when using this option.

**STOPPING PVMD3**:

The preferred method of stopping all the pvmds is to give the halt command in the PVM console. This kills all pvm tasks, all the remote daemons, the local daemon, and finally the console itself. If the master pvmd is killed manually it should be sent a SIGTERM signal to allow it to kill the remote pvmds and clean up various files.

The pvmd can be killed in a manner that leaves the file /tmp/pvmd.uid behind on one or more hosts. Uid is the numeric user ID (from /etc/passwd) of the user. This will prevent PVM from restarting on that host. Deletion of this file will fix this problem:

```
rm '( grep $user /etc/passwd || ypmatch $user passwd ) | \nawk -F: '{print "/tmp/pvmd."$3; exit}''
```

### 2.15.31 Communications — communications with other applications using GeCi

**DESCRIPTION**:
This the beta version of the Communications Toolbox.

GeCi manages communications between Scilab and other applications (included Scilab itself) by using GeCi.

GeCi is an interactive communication manager created in order to manage remote executions of softwares and allow exchanges of messages between those softwares. It offers the possibility to exploit numerous machines on a network, as a virtual computer, by creating a distributed group of independent softwares.

In order to communicate, the other applications must have been prepared for, by including a communication library in them. The way to do this is described in the Communication Toolbox documentation.

**SEE ALSO**: CreateLink 686, DestroyLink 687, ExecAppli 687, GetMsg 688, SendMsg 688, WaitMsg 689

### 2.15.32 CreateLink ____________ creates a link between two applications

**CALLING SEQUENCE**:

```
CreateLink(appli1,appli2)
```

**PARAMETERS**:

- `appli1`, name of an application : string
- `appli2` : string, name of an application

**DESCRIPTION**:

CreateLink creates a link from `appli1` to `appli2`. Note that this link is directed. This link being created, `appli1` can send messages to `appli2` and `appli2` can receive messages from `appli1`.

If the name of `appli1` or `appli2` is "SELF", it corresponds to the name of the application where we execute CreateLink.

If the name of `appli1` or `appli2` is "XGeCI", it corresponds to the name of the first Scilab application started.

**SEE ALSO**: DestroyLink 687, GetMsg 688, SendMsg 688
2.15.33  **DestroyLink**  

**CALLING SEQUENCE**:

`DestroyLink(appli1,appli2)`

**PARAMETERS**:

- `appli1`, name of an application : string
- `appli2`, name of an application : string

**DESCRIPTION**:

`DestroyLink` destroys the link from `appli1` to `appli2`.

If the name of `appli1` or `appli2` is "SELF", it corresponds to the name of the application where we execute `DestroyLink`.

If the name of `appli1` or `appli2` is "XGeCI", it corresponds to the name of the first Scilab application started.

**SEE ALSO**:

CreateLink 686

2.15.34  **ExecAppli**  

**CALLING SEQUENCE**:

`ExecAppli(command,h,appli)`

**PARAMETERS**:

- `command`, command of the application : string
- `h`, host name : string
- `appli`, name of the application : string

**DESCRIPTION**:

`ExecAppli` executes the application described by `command` on the host `h` and gives it the name `appli`. Arguments of the application can be also given in the string `command`.

After executing `ExecAppli`, it is necessary to create links with `CreateLink` to be able to send messages between applications.

**EXAMPLE**:

```plaintext
h=unix_g("hostname")
ExecAppli(SCI+"/bin/scilex",h,"Scilab2")
CreateLink("SELF","Scilab2")
ExecAppli(SCI+"/bin/scilex -ns",h,"Scilab3")
DestroyLink("SELF","Scilab2")
```

**SEE ALSO**:

CreateLink 686, ExecScilab 687, ExeclScilab 688

2.15.35  **ExecScilab**  

**CALLING SEQUENCE**:

`ExecScilab(appli)`

**PARAMETERS**:

- `appli`, name of the application : string

Scilab Group

March 1997

687
appli : string, name of new Scilab

DESCRIPTION:
ExecScilab executes a new Scilab application on the same host. After executing ExecScilab, it is necessary to create links with CreateLink to be able to send messages to new Scilab.
Use ExecAppli to execute a new Scilab application on a remote host.
SEE ALSO: CreateLink 686, ExecAppli 687, ExeclScilab 688

2.15.36 ExeclScilab process executes another linked local Scilab

CALLING SEQUENCE:
ExeclScilab(appli)

PARAMETERS:
appli : string, name of new Scilab

DESCRIPTION:
ExecScilab executes a new Scilab application on the same host and creates links between them.
Use ExecAppli to execute a new Scilab application on a remote host.
SEE ALSO: CreateLink 686, ExecAppli 687, ExeclScilab 687

2.15.37 GetMsg gets a pending message

CALLING SEQUENCE:
[type,msg,appli] = GetMsg()

PARAMETERS:
type : string
msg : string
appli : string, name of an application

DESCRIPTION:
GetMsg gets, in an asynchronous way, a message sent by another application. The type of the message is string and the message itself is msg. The name of the application which has sent the message is appli.
SEE ALSO: SendMsg 688, WaitMsg 689

2.15.38 SendMsg sends a message

CALLING SEQUENCE:
SendMsg(type,msg)

PARAMETERS:
type : string
msg : string

DESCRIPTION:
SendMsg sends a message to ALL applications linked to this application.
SEE ALSO: CreateLink 686, SendMsg 688, WaitMsg 689
**2.15.39 WaitMsg**

**Calling Sequence:**

\[\text{[type,msg]} = \text{WaitMsg}(\text{appli})\]

**Parameters:**

- `appli`: string, name of an application
- `type`: string
- `msg`: string

**Description:**

`WaitMsg` waits for a message sent by another application. As long as any message has not been sent, `WaitMsg` waits. This is a way to send and get messages in a synchronous way. The type of the message is `string` and the message itself is `msg`.

By default, the name of the first application (the one executed by GeCI) is "XGeCI".

**See Also:** GetMsg 688, SendMsg 688
Index

Symbols
%asn, 445
%helps, 296
%k, 445
%sn, 446

A
abcd, 318
abinv, 318
abort, 26
abs, 162
ABSBLK_f, 598
acos, 162
acosh, 162
acoshm, 163
acosm, 163
AdCommunications, 668
add_edge, 541
add_node, 541
addcolor, 85
addf, 164
addinter, 265
addmenu, 287
adj2sp, 164
adj_lists, 542
aff2ab, 501
AFFICH_f, 598
alufunctions, 85
amell, 165
analpf, 446
analyze, 631
and, 165
ANDLOG_f, 598
ANIMXY_f, 599
ans, 26
apropos, 26
arc_graph, 543
arc_number, 543
argn, 265
arhnk, 321
ar12, 321
arma, 393
arma2p, 393
armac, 394
armax, 395
armax1, 396
arsimul, 396
artext, 400
articul, 544
ascii, 660
asin, 166
asinh, 166
asinhm, 167
asinn, 167
atan, 167
atanh, 168
atanhm, 169
atanm, 169
augment, 373
auread, 631
auwrite, 632

B
backslash, 26
balanc, 502
balreal, 322
bandwr, 544
bdiag, 502
besseli, 169
besselj, 170
bessely, 171
besselj, 170
best_match, 545
bezout, 484
bifish, 400
BIGSOM_f, 599
bilin, 323
binomial, 171
black, 85
bloc2exp, 172
bloc2ss, 173
bode, 86
bool2s, 27
boolean, 27
boucle, 401
brackets, 28
break, 28
bstap, 374
buttmag, 447
bvoide, 411

C
690
<table>
<thead>
<tr>
<th>Scilab function</th>
<th>Scilab function</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_link, 297</td>
<td>cmndred, 485</td>
</tr>
<tr>
<td>cainv, 323</td>
<td>code2str, 280</td>
</tr>
<tr>
<td>calerf, 175</td>
<td>coeff, 485</td>
</tr>
<tr>
<td>calfrq, 324</td>
<td>coff, 505</td>
</tr>
<tr>
<td>call, 28</td>
<td>coffg, 485</td>
</tr>
<tr>
<td>canon, 325</td>
<td>colcomp, 505</td>
</tr>
<tr>
<td>casc, 447</td>
<td>colcompr, 486</td>
</tr>
<tr>
<td>case, 30</td>
<td>colinout, 375</td>
</tr>
<tr>
<td>ccontrg, 374</td>
<td>colnew, 414</td>
</tr>
<tr>
<td>cdfbet, 638</td>
<td>colon, 31</td>
</tr>
<tr>
<td>cdfbin, 638</td>
<td>colormap, 89</td>
</tr>
<tr>
<td>cdfchi, 639</td>
<td>colregul, 326</td>
</tr>
<tr>
<td>cdfchnc, 639</td>
<td>comma, 32</td>
</tr>
<tr>
<td>cdf, 640</td>
<td>comments, 32</td>
</tr>
<tr>
<td>cdfnc, 640</td>
<td>Communications, 686</td>
</tr>
<tr>
<td>cdff, 640</td>
<td>comp, 266</td>
</tr>
<tr>
<td>cdffnc, 640</td>
<td>companion, 506</td>
</tr>
<tr>
<td>cdffgam, 641</td>
<td>con_nodes, 548</td>
</tr>
<tr>
<td>cdffnbn, 642</td>
<td>cond, 506</td>
</tr>
<tr>
<td>cdffnor, 642</td>
<td>conj, 176</td>
</tr>
<tr>
<td>cdffpoi, 643</td>
<td>connex, 549</td>
</tr>
<tr>
<td>cdft, 643</td>
<td>CONST.f, 602</td>
</tr>
<tr>
<td>ceil, 176</td>
<td>cont.frm, 327</td>
</tr>
<tr>
<td>cepstrum, 448</td>
<td>cont.mat, 327</td>
</tr>
<tr>
<td>chain_struct, 546</td>
<td>contour, 90</td>
</tr>
<tr>
<td>chaintest, 401</td>
<td>contour2d, 91</td>
</tr>
<tr>
<td>champ, 87</td>
<td>contour2di, 92</td>
</tr>
<tr>
<td>champ1, 88</td>
<td>contourf, 93</td>
</tr>
<tr>
<td>chart, 88</td>
<td>contr, 328</td>
</tr>
<tr>
<td>chdir, 297</td>
<td>contract.edge, 549</td>
</tr>
<tr>
<td>cheb1mag, 448</td>
<td>contrss, 328</td>
</tr>
<tr>
<td>cheb2mag, 449</td>
<td>convex_hull, 550</td>
</tr>
<tr>
<td>check_graph, 547</td>
<td>convol, 450</td>
</tr>
<tr>
<td>chepol, 449</td>
<td>convstr, 280</td>
</tr>
<tr>
<td>chfact, 503</td>
<td>copfac, 375</td>
</tr>
<tr>
<td>chol, 503</td>
<td>corr, 450</td>
</tr>
<tr>
<td>chsolve, 504</td>
<td>cos, 177</td>
</tr>
<tr>
<td>circuit, 548</td>
<td>COSBLK.f, 602</td>
</tr>
<tr>
<td>classmarkov, 504</td>
<td>cosh, 177</td>
</tr>
<tr>
<td>clean, 484</td>
<td>coshm, 178</td>
</tr>
<tr>
<td>clear, 30</td>
<td>cosm, 178</td>
</tr>
<tr>
<td>clearfun, 265</td>
<td>cotg, 178</td>
</tr>
<tr>
<td>clearglobal, 31</td>
<td>coth, 179</td>
</tr>
<tr>
<td>CLINDUMMY.f, 599</td>
<td>cothm, 179</td>
</tr>
<tr>
<td>CLKIN.f, 600</td>
<td>CreateLink, 686</td>
</tr>
<tr>
<td>CLKINV.f, 600</td>
<td>csim, 329</td>
</tr>
<tr>
<td>CLKOUT.f, 600</td>
<td>espect, 452</td>
</tr>
<tr>
<td>CLKOUTV.f, 600</td>
<td>ctr_gram, 330</td>
</tr>
<tr>
<td>CLKSOM.f, 601</td>
<td>cumprod, 179</td>
</tr>
<tr>
<td>CLKSOMV.f, 600</td>
<td>cumsum, 180</td>
</tr>
<tr>
<td>CLKSPLIT.f, 601</td>
<td>curbblock, 627</td>
</tr>
<tr>
<td>CLOCK.f, 601</td>
<td>CURV.f, 602</td>
</tr>
<tr>
<td>close, 652</td>
<td>cycle_basis, 551</td>
</tr>
<tr>
<td>CLR.f, 601</td>
<td>czt, 453</td>
</tr>
<tr>
<td>cls2dls, 326</td>
<td></td>
</tr>
<tr>
<td>CLSS.f, 602</td>
<td></td>
</tr>
<tr>
<td>cmb_lin, 176</td>
<td></td>
</tr>
</tbody>
</table>

Scilab Group March 1997 691
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dasrt</td>
<td>414</td>
</tr>
<tr>
<td>dassl</td>
<td>416</td>
</tr>
<tr>
<td>datafit</td>
<td>417</td>
</tr>
<tr>
<td>date</td>
<td>314</td>
</tr>
<tr>
<td>dbphi</td>
<td>330</td>
</tr>
<tr>
<td>def</td>
<td>375</td>
</tr>
<tr>
<td>ddp</td>
<td>331</td>
</tr>
<tr>
<td>debug</td>
<td>32</td>
</tr>
<tr>
<td>dec2hex</td>
<td>298</td>
</tr>
<tr>
<td>deff</td>
<td>266</td>
</tr>
<tr>
<td>degree</td>
<td>486</td>
</tr>
<tr>
<td>DELAY_f</td>
<td>603</td>
</tr>
<tr>
<td>DELAYV_f</td>
<td>603</td>
</tr>
<tr>
<td>delbpt</td>
<td>267</td>
</tr>
<tr>
<td>delete_arcs</td>
<td>551</td>
</tr>
<tr>
<td>delete_nodes</td>
<td>552</td>
</tr>
<tr>
<td>delip</td>
<td>180</td>
</tr>
<tr>
<td>delmenu</td>
<td>288</td>
</tr>
<tr>
<td>demos</td>
<td>298</td>
</tr>
<tr>
<td>DEMUX_f</td>
<td>603</td>
</tr>
<tr>
<td>denom</td>
<td>486</td>
</tr>
<tr>
<td>derivat</td>
<td>487</td>
</tr>
<tr>
<td>derivative-</td>
<td>419</td>
</tr>
<tr>
<td>des2ss</td>
<td>376</td>
</tr>
<tr>
<td>des2tf</td>
<td>332</td>
</tr>
<tr>
<td>DestroyLink</td>
<td>687</td>
</tr>
<tr>
<td>det</td>
<td>507</td>
</tr>
<tr>
<td>determ</td>
<td>487</td>
</tr>
<tr>
<td>detr</td>
<td>487</td>
</tr>
<tr>
<td>dft</td>
<td>454</td>
</tr>
<tr>
<td>dnorm</td>
<td>376</td>
</tr>
<tr>
<td>diag</td>
<td>181</td>
</tr>
<tr>
<td>diary</td>
<td>233</td>
</tr>
<tr>
<td>diophant</td>
<td>488</td>
</tr>
<tr>
<td>disp</td>
<td>233</td>
</tr>
<tr>
<td>dispbpt</td>
<td>267</td>
</tr>
<tr>
<td>dispfile</td>
<td>233</td>
</tr>
<tr>
<td>dlgamma</td>
<td>182</td>
</tr>
<tr>
<td>DLR_f</td>
<td>604</td>
</tr>
<tr>
<td>DLRADAPT_f</td>
<td>604</td>
</tr>
<tr>
<td>DLSS_f</td>
<td>604</td>
</tr>
<tr>
<td>dot</td>
<td>32</td>
</tr>
<tr>
<td>double</td>
<td>182</td>
</tr>
<tr>
<td>dragrect</td>
<td>94</td>
</tr>
<tr>
<td>drawaxis</td>
<td>94</td>
</tr>
<tr>
<td>driver</td>
<td>95</td>
</tr>
<tr>
<td>dscr</td>
<td>333</td>
</tr>
<tr>
<td>dsimul</td>
<td>333</td>
</tr>
<tr>
<td>dtility</td>
<td>334</td>
</tr>
<tr>
<td>dtsi</td>
<td>377</td>
</tr>
<tr>
<td>E</td>
<td>edge_number, 553</td>
</tr>
<tr>
<td></td>
<td>edit, 267</td>
</tr>
<tr>
<td></td>
<td>edit_curv, 96</td>
</tr>
<tr>
<td></td>
<td>eigenmarkov, 507</td>
</tr>
<tr>
<td></td>
<td>ell1mag, 455</td>
</tr>
<tr>
<td></td>
<td>else, 33</td>
</tr>
<tr>
<td></td>
<td>elseif, 33</td>
</tr>
<tr>
<td></td>
<td>empty, 33</td>
</tr>
<tr>
<td></td>
<td>emptystr, 280</td>
</tr>
<tr>
<td></td>
<td>end, 34</td>
</tr>
<tr>
<td></td>
<td>endfunction, 268</td>
</tr>
<tr>
<td></td>
<td>eqfir, 455</td>
</tr>
<tr>
<td></td>
<td>eqiir, 456</td>
</tr>
<tr>
<td></td>
<td>equal, 34</td>
</tr>
<tr>
<td></td>
<td>equil, 334</td>
</tr>
<tr>
<td></td>
<td>equil1, 335</td>
</tr>
<tr>
<td></td>
<td>ereduc, 508</td>
</tr>
<tr>
<td></td>
<td>erf, 182</td>
</tr>
<tr>
<td></td>
<td>erfc, 183</td>
</tr>
<tr>
<td></td>
<td>erfcx, 183</td>
</tr>
<tr>
<td></td>
<td>errbar, 96</td>
</tr>
<tr>
<td></td>
<td>errcatch, 34</td>
</tr>
<tr>
<td></td>
<td>errclear, 35</td>
</tr>
<tr>
<td></td>
<td>error, 35</td>
</tr>
<tr>
<td></td>
<td>eval, 184</td>
</tr>
<tr>
<td></td>
<td>eval3d, 97</td>
</tr>
<tr>
<td></td>
<td>eval3dp, 98</td>
</tr>
<tr>
<td></td>
<td>evans, 98</td>
</tr>
<tr>
<td></td>
<td>EVENTSCOPE_f, 605</td>
</tr>
<tr>
<td></td>
<td>evstr, 35</td>
</tr>
<tr>
<td></td>
<td>EVTDLY_f, 605</td>
</tr>
<tr>
<td></td>
<td>EVTGEN_f, 605</td>
</tr>
<tr>
<td></td>
<td>Example, 668</td>
</tr>
<tr>
<td></td>
<td>excel2sci, 660</td>
</tr>
<tr>
<td></td>
<td>exec, 36</td>
</tr>
<tr>
<td></td>
<td>ExecAppli, 687</td>
</tr>
<tr>
<td></td>
<td>ExecScilab, 687</td>
</tr>
<tr>
<td></td>
<td>execstr, 37</td>
</tr>
<tr>
<td></td>
<td>exists, 37</td>
</tr>
<tr>
<td></td>
<td>exit, 38</td>
</tr>
<tr>
<td></td>
<td>exp, 508</td>
</tr>
<tr>
<td></td>
<td>EXPBLK_f, 606</td>
</tr>
<tr>
<td></td>
<td>expm, 509</td>
</tr>
<tr>
<td></td>
<td>external, 38</td>
</tr>
<tr>
<td></td>
<td>extraction, 39</td>
</tr>
<tr>
<td></td>
<td>eye, 184</td>
</tr>
<tr>
<td>F</td>
<td>fac3d, 99</td>
</tr>
<tr>
<td></td>
<td>factors, 488</td>
</tr>
<tr>
<td></td>
<td>faurre, 456</td>
</tr>
<tr>
<td></td>
<td>fhchamp, 99</td>
</tr>
<tr>
<td></td>
<td>fecontour, 100</td>
</tr>
<tr>
<td></td>
<td>fecontour2d, 100</td>
</tr>
<tr>
<td></td>
<td>fec, 101</td>
</tr>
<tr>
<td></td>
<td>feedback, 335</td>
</tr>
</tbody>
</table>
WaitMsg Scilab function

feval, 41
gainplot, 104
ffilt, 457
gamitg, 378
fft, 457
gamma, 186
fgrayplot, 102
gammaln, 186
figure, 652
gcare, 379
file, 234
gcd, 489
fileinfo, 235
gcf, 654
filter, 458
gen.get, 554
find, 41
GENERAL_f, 606
find_freq, 458
GENERIC_f, 607
find_path, 553
genfac3d, 105
findm, 459
genlib, 270
findobj, 653
genmarkov, 511
fit, 419
GENSIN_f, 607
fix, 184
GENSQR_f, 607
floor, 185
geom3d, 105
flts, 336
get, 654
for, 42
get_function_path, 271
format, 42
getblocklabel, 627
formatman, 660
cwd, 67
fort, 43
get, 271
fourplan, 377
date, 314
fplot2d, 102
getenv, 45
fplot3d, 103
getf, 272
fplot3d1, 103
getfield, 45
fprintf, 235
getfont, 106
fprintfMat, 236
getio, 238
grep, 235
getlinestyle, 106
tprint, 235
getmark, 107
tprintMat, 236
GetMsg, 688
gfreq, 338
getscicosvars, 627
gfreq, 339
getsymbol, 107
gfrep2tf, 338
getvalue, 288
gfreq, 339
getversion, 46
freson, 339
gfare, 379
gfspecg, 340
gfrexp, 185
gfFrancis, 340
gfrfit, 459
ergth, 555
gfrfit, 459
givens, 511
gfrfit, 459
glcver, 512
gfrfit, 459
glist, 555
gfrfit, 459
get, 182
glist, 555
gfullrf, 510
gglobal, 46
gfullrfk, 510
ggpeche, 402
fun2string, 661
ggpeche, 402
funcprot, 268
gr_menu, 107
function, 268
graduate, 108
functions, 269
grand, 644
funptr, 45
graph_list, 555
fusee, 402
graph_2_mat, 558

G

G_make, 296
graph_center, 558
G_margin, 340
graph_complement, 559
GAIN_f, 606
graph_diameter, 560
GAINBLK_f, 606
graph_power, 560

Scilab Group
Scilab function

March 1997
693
graph_union, 562
Graphics, 80
graycolormap, 108
grayplot, 109
graypolarplot, 109
grep, 281
group, 461
gschur, 513
gsort, 187
gspec, 514
gstacksize, 47
gtild, 380

H
h2norm, 381
h_cl, 381
h_inf, 381
h_inf_st, 382
h_norm, 382
halt, 289
HALT_f, 608
hamilton, 563
hank, 461
hankelsv, 383
hat, 47
havewindow, 289
help, 298
hermit, 489
hess, 514
hex2dec, 299
hilb, 462
hist3d, 110
histplot, 110
horner, 490
host, 48
hotcolormap, 111
householder, 515
hrmt, 490
htrianr, 491
hypermat, 48
hypermatrices, 49

I
iconvert, 49
ieee, 50
if, 50
IFTHEL_f, 608
iir, 463
iirgroup, 463
iirlp, 464
ilib_build, 299
ilib_compile, 300
ilib_forlink, 301
ilib_gen_gateway, 302
ilib_gen_loader, 302

Scilab Group

March 1997

694
<table>
<thead>
<tr>
<th>Scilab function</th>
<th>WaitMsg Scilab function</th>
</tr>
</thead>
<tbody>
<tr>
<td>kron, 194</td>
<td>lsslist, 57</td>
</tr>
<tr>
<td>kroneck, 517</td>
<td>lstcat, 57</td>
</tr>
<tr>
<td>L</td>
<td>ltitr, 347</td>
</tr>
<tr>
<td>lasterror, 55</td>
<td>lu, 519</td>
</tr>
<tr>
<td>lattn, 465</td>
<td>ludel, 519</td>
</tr>
<tr>
<td>lattp, 466</td>
<td>lufact, 520</td>
</tr>
<tr>
<td>lcm, 492</td>
<td>lusolve, 521</td>
</tr>
<tr>
<td>lcmdiag, 492</td>
<td>lyap, 522</td>
</tr>
<tr>
<td>lcf, 383</td>
<td></td>
</tr>
<tr>
<td>lcmdiag, 492</td>
<td></td>
</tr>
<tr>
<td>ldiv, 493</td>
<td></td>
</tr>
<tr>
<td>ldivf, 194</td>
<td></td>
</tr>
<tr>
<td>leastsq, 427</td>
<td></td>
</tr>
<tr>
<td>left, 56</td>
<td></td>
</tr>
<tr>
<td>legends, 112</td>
<td></td>
</tr>
<tr>
<td>length, 281</td>
<td></td>
</tr>
<tr>
<td>leqr, 383</td>
<td></td>
</tr>
<tr>
<td>less, 56</td>
<td></td>
</tr>
<tr>
<td>lev, 466</td>
<td></td>
</tr>
<tr>
<td>levin, 466</td>
<td></td>
</tr>
<tr>
<td>lex_sort, 194</td>
<td></td>
</tr>
<tr>
<td>lft, 384</td>
<td></td>
</tr>
<tr>
<td>lgfft, 468</td>
<td></td>
</tr>
<tr>
<td>lib, 272</td>
<td></td>
</tr>
<tr>
<td>lin, 343</td>
<td></td>
</tr>
<tr>
<td>lin2mu, 632</td>
<td></td>
</tr>
<tr>
<td>lindquist, 469</td>
<td></td>
</tr>
<tr>
<td>line_graph, 564</td>
<td></td>
</tr>
<tr>
<td>lines, 238</td>
<td></td>
</tr>
<tr>
<td>linf, 385</td>
<td></td>
</tr>
<tr>
<td>linfn, 385</td>
<td></td>
</tr>
<tr>
<td>link, 303</td>
<td></td>
</tr>
<tr>
<td>linpro, 429</td>
<td></td>
</tr>
<tr>
<td>linsolve, 518</td>
<td></td>
</tr>
<tr>
<td>linspace, 195</td>
<td></td>
</tr>
<tr>
<td>list, 57</td>
<td></td>
</tr>
<tr>
<td>lmisolver, 430</td>
<td></td>
</tr>
<tr>
<td>lmitool, 431</td>
<td></td>
</tr>
<tr>
<td>load, 239</td>
<td></td>
</tr>
<tr>
<td>load_graph, 565</td>
<td></td>
</tr>
<tr>
<td>loadwave, 632</td>
<td></td>
</tr>
<tr>
<td>locate, 112</td>
<td></td>
</tr>
<tr>
<td>log, 195</td>
<td></td>
</tr>
<tr>
<td>log10, 196</td>
<td></td>
</tr>
<tr>
<td>log2, 196</td>
<td></td>
</tr>
<tr>
<td>LOGBLK_f, 609</td>
<td></td>
</tr>
<tr>
<td>logm, 196</td>
<td></td>
</tr>
<tr>
<td>logspace, 197</td>
<td></td>
</tr>
<tr>
<td>LOOKUP_f, 609</td>
<td></td>
</tr>
<tr>
<td>lotest, 403</td>
<td></td>
</tr>
<tr>
<td>lqe, 344</td>
<td></td>
</tr>
<tr>
<td>lqg, 344</td>
<td></td>
</tr>
<tr>
<td>lqg2stan, 345</td>
<td></td>
</tr>
<tr>
<td>lqgltr, 386</td>
<td></td>
</tr>
<tr>
<td>lqr, 345</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
</tr>
<tr>
<td>m_circle, 113</td>
<td></td>
</tr>
<tr>
<td>macglov, 387</td>
<td></td>
</tr>
<tr>
<td>macr2lst, 273</td>
<td></td>
</tr>
<tr>
<td>macro, 273</td>
<td></td>
</tr>
<tr>
<td>macrovar, 274</td>
<td></td>
</tr>
<tr>
<td>make_graph, 566</td>
<td></td>
</tr>
<tr>
<td>man, 305</td>
<td></td>
</tr>
<tr>
<td>manedit, 239</td>
<td></td>
</tr>
<tr>
<td>mapsound, 633</td>
<td></td>
</tr>
<tr>
<td>markp2ss, 347</td>
<td></td>
</tr>
<tr>
<td>mat2_graph, 566</td>
<td></td>
</tr>
<tr>
<td>Matplot, 82</td>
<td></td>
</tr>
<tr>
<td>Matplot1, 83</td>
<td></td>
</tr>
<tr>
<td>matrices, 58</td>
<td></td>
</tr>
<tr>
<td>matrix, 58</td>
<td></td>
</tr>
<tr>
<td>max, 197</td>
<td></td>
</tr>
<tr>
<td>max_cap_path, 567</td>
<td></td>
</tr>
<tr>
<td>max_clique, 568</td>
<td></td>
</tr>
<tr>
<td>MAX_f, 609</td>
<td></td>
</tr>
<tr>
<td>max_flow, 568</td>
<td></td>
</tr>
<tr>
<td>maxi, 198</td>
<td></td>
</tr>
<tr>
<td>mclearerr, 240</td>
<td></td>
</tr>
<tr>
<td>MCLOCK_f, 610</td>
<td></td>
</tr>
<tr>
<td>mclose, 240</td>
<td></td>
</tr>
<tr>
<td>mean, 198</td>
<td></td>
</tr>
<tr>
<td>median, 199</td>
<td></td>
</tr>
<tr>
<td>meof, 241</td>
<td></td>
</tr>
<tr>
<td>mese, 469</td>
<td></td>
</tr>
<tr>
<td>mesh2d, 569</td>
<td></td>
</tr>
<tr>
<td>metanet, 571</td>
<td></td>
</tr>
<tr>
<td>metanet_sync, 572</td>
<td></td>
</tr>
<tr>
<td>MFCLCK_f, 610</td>
<td></td>
</tr>
<tr>
<td>mfft, 470</td>
<td></td>
</tr>
<tr>
<td>mfile2sci, 661</td>
<td></td>
</tr>
<tr>
<td>mfprintf, 245</td>
<td></td>
</tr>
<tr>
<td>mfsca, 241, 248</td>
<td></td>
</tr>
<tr>
<td>mg, 242</td>
<td></td>
</tr>
<tr>
<td>mget, 242</td>
<td></td>
</tr>
<tr>
<td>mgetl, 243</td>
<td></td>
</tr>
<tr>
<td>mgetstr, 244</td>
<td></td>
</tr>
<tr>
<td>milk_drop, 114</td>
<td></td>
</tr>
<tr>
<td>min, 199</td>
<td></td>
</tr>
<tr>
<td>MIN_f, 610</td>
<td></td>
</tr>
<tr>
<td>min_lcost_cflow, 572</td>
<td></td>
</tr>
</tbody>
</table>
min\textsubscript{lcost}\_flow1, 573
min\textsubscript{lcost}\_flow2, 574
min\textsubscript{lcost}\_flow, 576
min\textsubscript{weight}\_tree, 577
mine, 404
mini, 200
minreal, 348
minss, 348
minus, 201
mlist, 59
mode, 59
modulo, 201
mopen, 244
mprinf, 245
mps2linpro, 202
mput, 246
mputl, 247
mpustr, 247
mrfit, 470
mscanf, 241, 248
mseek, 248
mprintf, 245
msprintf, 245
msscanf, 241, 248
mtell, 249
mtlb\_load, 663
mtlb\_mode, 60
mtlb\_save, 663
mtlb\_sparse, 202
mu2lin, 633
mulf, 203
MUX\_f, 610

N
names, 61
narsimul, 397
NEGTOPOS\_f, 611
nehari, 387
neighbors, 577
netclose, 578
netwindow, 578
netwindows, 579
newest, 250
newfun, 274
nf3d, 114
nlev, 522
nnz, 203
node\_number, 579
nodes\_2\_path, 579
nodes\_degrees, 580
noisegen, 397
norm, 203
not, 204
null, 61
numer, 493
nyquist, 115

O
obs\_gram, 349
obscont, 349
obscont1, 405
observer, 350
obs\_mat, 351
obs\_vss, 352
ode, 431
ode\_discrete, 434
ode\_root, 435
ode\_dec, 436
odedi, 397
ode\_options, 437
oldload, 250
oldsave, 251
ones, 204
optim, 438
or, 205
orth, 523
OUT\_f, 611
overloading, 61

P
p\_margin, 352
param3d, 115
param3d1, 116
paramfplot2d, 117
parents, 63
parrot, 387
part, 282
path\_2\_nodes, 580
pause, 64
pbig, 523
pdiv, 493
pen2ea, 205
pencan, 524
penlaur, 524
percent, 64
perfect\_match, 581
pertrans, 206
pfss, 353
phasemag, 353
phc, 471
pinv, 525
pipe\_network, 582
playsnd, 634
plot, 118
plot2d, 118
plot2d1, 121
plot2d2, 122
plot2d3, 122
plot2d4, 123
plot3d, 123
plot3d1, 125
plot3d2, 126
<table>
<thead>
<tr>
<th>Scilab function</th>
<th>WaitMsg Scilab function</th>
</tr>
</thead>
<tbody>
<tr>
<td>plot3d3, 127</td>
<td>plot3d3, 127</td>
</tr>
<tr>
<td>plot_graph, 582</td>
<td>plot_graph, 582</td>
</tr>
<tr>
<td>plotframe, 127</td>
<td>plotframe, 127</td>
</tr>
<tr>
<td>plotprofile, 274</td>
<td>plotprofile, 274</td>
</tr>
<tr>
<td>plus, 64</td>
<td>plus, 64</td>
</tr>
<tr>
<td>plzr, 128</td>
<td>plzr, 128</td>
</tr>
<tr>
<td>pmodulo, 201</td>
<td>pmodulo, 201</td>
</tr>
<tr>
<td>pol2des, 494</td>
<td>pol2des, 494</td>
</tr>
<tr>
<td>pol2str, 494</td>
<td>pol2str, 494</td>
</tr>
<tr>
<td>pol2tex, 664</td>
<td>pol2tex, 664</td>
</tr>
<tr>
<td>polar, 525</td>
<td>polar, 525</td>
</tr>
<tr>
<td>polarplot, 128</td>
<td>polarplot, 128</td>
</tr>
<tr>
<td>polfact, 495</td>
<td>polfact, 495</td>
</tr>
<tr>
<td>poly, 65</td>
<td>poly, 65</td>
</tr>
<tr>
<td>port3d, 405</td>
<td>port3d, 405</td>
</tr>
<tr>
<td>portrait, 406</td>
<td>portrait, 406</td>
</tr>
<tr>
<td>POSTONEG, f, 611</td>
<td>POSTONEG, f, 611</td>
</tr>
<tr>
<td>POWBLK, 611</td>
<td>POWBLK, 611</td>
</tr>
<tr>
<td>power, 66</td>
<td>power, 66</td>
</tr>
<tr>
<td>ppol, 354</td>
<td>ppol, 354</td>
</tr>
<tr>
<td>prbs_a, 398</td>
<td>prbs_a, 398</td>
</tr>
<tr>
<td>predecessors, 584</td>
<td>predecessors, 584</td>
</tr>
<tr>
<td>predef, 66</td>
<td>predef, 66</td>
</tr>
<tr>
<td>print, 251</td>
<td>print, 251</td>
</tr>
<tr>
<td>printf, 252</td>
<td>printf, 252</td>
</tr>
<tr>
<td>printf_conversion, 252</td>
<td>printf_conversion, 252</td>
</tr>
<tr>
<td>printing, 129</td>
<td>printing, 129</td>
</tr>
<tr>
<td>prod, 206</td>
<td>prod, 206</td>
</tr>
<tr>
<td>PROD, f, 611</td>
<td>PROD, f, 611</td>
</tr>
<tr>
<td>profile, 275</td>
<td>profile, 275</td>
</tr>
<tr>
<td>proj, 526</td>
<td>proj, 526</td>
</tr>
<tr>
<td>projsl, 354</td>
<td>projsl, 354</td>
</tr>
<tr>
<td>projspec, 526</td>
<td>projspec, 526</td>
</tr>
<tr>
<td>psmall, 527</td>
<td>psmall, 527</td>
</tr>
<tr>
<td>paspect, 472</td>
<td>paspect, 472</td>
</tr>
<tr>
<td>pvm, 668</td>
<td>pvm, 668</td>
</tr>
<tr>
<td>pvm_addhosts, 669</td>
<td>pvm_addhosts, 669</td>
</tr>
<tr>
<td>pvm_bcast, 669</td>
<td>pvm_bcast, 669</td>
</tr>
<tr>
<td>pvm_bufinfo, 670</td>
<td>pvm_bufinfo, 670</td>
</tr>
<tr>
<td>pvm_config, 670</td>
<td>pvm_config, 670</td>
</tr>
<tr>
<td>pvm_delhosts, 671</td>
<td>pvm_delhosts, 671</td>
</tr>
<tr>
<td>pvm_error, 671</td>
<td>pvm_error, 671</td>
</tr>
<tr>
<td>pvm_exit, 672</td>
<td>pvm_exit, 672</td>
</tr>
<tr>
<td>pvm_gettimer, 673</td>
<td>pvm_gettimer, 673</td>
</tr>
<tr>
<td>pvm_getinst, 673</td>
<td>pvm_getinst, 673</td>
</tr>
<tr>
<td>pvm_gsize, 674</td>
<td>pvm_gsize, 674</td>
</tr>
<tr>
<td>pvm_hal, 674</td>
<td>pvm_hal, 674</td>
</tr>
<tr>
<td>pvm_join, 675</td>
<td>pvm_join, 675</td>
</tr>
<tr>
<td>pvm_kill, 675</td>
<td>pvm_kill, 675</td>
</tr>
<tr>
<td>pvm_lvgroup, 676</td>
<td>pvm_lvgroup, 676</td>
</tr>
<tr>
<td>pvm_mytid, 676</td>
<td>pvm_mytid, 676</td>
</tr>
<tr>
<td>pvm_probe, 677</td>
<td>pvm_probe, 677</td>
</tr>
<tr>
<td>pvm_recv, 677</td>
<td>pvm_recv, 677</td>
</tr>
<tr>
<td>pvm_reduce, 678</td>
<td>pvm_reduce, 678</td>
</tr>
<tr>
<td>pvm_sci2f77, 672, 679</td>
<td>pvm_sci2f77, 672, 679</td>
</tr>
<tr>
<td>pvm_send, 679</td>
<td>pvm_send, 679</td>
</tr>
<tr>
<td>pvm_settimer, 680</td>
<td>pvm_settimer, 680</td>
</tr>
<tr>
<td>pvm_spawn, 681</td>
<td>pvm_spawn, 681</td>
</tr>
<tr>
<td>pvm_spawn_independent, 681</td>
<td>pvm_spawn_independent, 681</td>
</tr>
<tr>
<td>pvm_start, 682</td>
<td>pvm_start, 682</td>
</tr>
<tr>
<td>pvm_tidtohost, 683</td>
<td>pvm_tidtohost, 683</td>
</tr>
<tr>
<td>pvmd3, 683</td>
<td>pvmd3, 683</td>
</tr>
<tr>
<td>pwd, 67</td>
<td>pwd, 67</td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>qassign, 584</td>
<td>qassign, 584</td>
</tr>
<tr>
<td>qr, 528</td>
<td>qr, 528</td>
</tr>
<tr>
<td>QUANT, f, 612</td>
<td>QUANT, f, 612</td>
</tr>
<tr>
<td>quapro, 440</td>
<td>quapro, 440</td>
</tr>
<tr>
<td>quaskro, 528</td>
<td>quaskro, 528</td>
</tr>
<tr>
<td>quit, 67</td>
<td>quit, 67</td>
</tr>
<tr>
<td>quote, 67</td>
<td>quote, 67</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>rand, 207</td>
<td>rand, 207</td>
</tr>
<tr>
<td>RAND, f, 612</td>
<td>RAND, f, 612</td>
</tr>
<tr>
<td>randpencil, 529</td>
<td>randpencil, 529</td>
</tr>
<tr>
<td>range, 530</td>
<td>range, 530</td>
</tr>
<tr>
<td>rank, 530</td>
<td>rank, 530</td>
</tr>
<tr>
<td>rat, 207</td>
<td>rat, 207</td>
</tr>
<tr>
<td>rational, 68</td>
<td>rational, 68</td>
</tr>
<tr>
<td>rcond, 531</td>
<td>rcond, 531</td>
</tr>
<tr>
<td>rdivf, 208</td>
<td>rdivf, 208</td>
</tr>
<tr>
<td>read, 254</td>
<td>read, 254</td>
</tr>
<tr>
<td>read4b, 255</td>
<td>read4b, 255</td>
</tr>
<tr>
<td>readb, 255</td>
<td>readb, 255</td>
</tr>
<tr>
<td>readc, 256</td>
<td>readc, 256</td>
</tr>
<tr>
<td>READC, f, 612</td>
<td>READC, f, 612</td>
</tr>
<tr>
<td>readmps, 256</td>
<td>readmps, 256</td>
</tr>
<tr>
<td>real, 208</td>
<td>real, 208</td>
</tr>
<tr>
<td>recur, 406</td>
<td>recur, 406</td>
</tr>
<tr>
<td>REGISTER, f, 613</td>
<td>REGISTER, f, 613</td>
</tr>
<tr>
<td>reglin, 398</td>
<td>reglin, 398</td>
</tr>
<tr>
<td>RELAY, f, 613</td>
<td>RELAY, f, 613</td>
</tr>
<tr>
<td>remez, 473</td>
<td>remez, 473</td>
</tr>
<tr>
<td>remezb, 473</td>
<td>remezb, 473</td>
</tr>
<tr>
<td>repfreq, 355</td>
<td>repfreq, 355</td>
</tr>
<tr>
<td>replot, 130</td>
<td>replot, 130</td>
</tr>
<tr>
<td>residu, 495</td>
<td>residu, 495</td>
</tr>
<tr>
<td>resume, 68</td>
<td>resume, 68</td>
</tr>
<tr>
<td>return, 68</td>
<td>return, 68</td>
</tr>
<tr>
<td>RFILE, f, 613</td>
<td>RFILE, f, 613</td>
</tr>
<tr>
<td>ric_desc, 388</td>
<td>ric_desc, 388</td>
</tr>
<tr>
<td>ricc, 356</td>
<td>ricc, 356</td>
</tr>
<tr>
<td>riccati, 389</td>
<td>riccati, 389</td>
</tr>
<tr>
<td>rlist, 69</td>
<td>rlist, 69</td>
</tr>
<tr>
<td>roots, 496</td>
<td>roots, 496</td>
</tr>
<tr>
<td>rotate, 131</td>
<td>rotate, 131</td>
</tr>
<tr>
<td>round, 209</td>
<td>round, 209</td>
</tr>
<tr>
<td>routh, f, 496</td>
<td>routh, f, 496</td>
</tr>
</tbody>
</table>

Scilab Group March 1997
rowcomp, 531
rowcompr, 496
rowinout, 389
rowregul, 357
rowshuff, 532
rrem, 474
rref, 532
rtitr, 358

S
salesman, 585
SAMPLEHOLD, 614
SAT, 614
save, 258
save_graph, 585
savewave, 634
SAWTOOTH, 614
scaling, 131
scanf, 259
scanf_conversion, 259
schur, 533
sci2exp, 307
sci2for, 664
sci2map, 308
sciargs, 69
scicos, 595
scicos_block, 621
scicos_cpr, 623
scicos_graphics, 621
scicos_link, 622
scicos_main, 620
scicos_menus, 595
scicos_model, 622
scicosim, 626
scifunc_block, 619
scilab, 308
ScilabEval, 649
scilink, 309
SCOPE, 615
SCOPXY, 615
ds2sci, 131
secto3d, 132
select, 69
SELECT, 616
semi, 70
semicolumn, 70
semidef, 441
SendMsg, 688
sensi, 390
set, 655
setbpt, 276
setfield, 70
setmenu, 290
setscicosvars, 628
sfact, 497

Scilab Group March 1997
ss2tf, 363
sscanf, 260
sskf, 477
ssprint, 220
ssrand, 220
stdeviation, 221
stility, 364
stabil, 364
stacksize, 71
standarddefine, 624
standarddraw, 624
standardinput, 625
standardorigin, 625
standardoutput, 625
star, 72
startup, 261
STOP.f, 617
str2code, 282
strcat, 283
strindex, 283
string, 284
strings, 284
stripblanks, 284
strongconnodes, 589
strongconnex, 590
strsubst, 285
subf, 222
subgraph, 590
subplot, 133
successors, 591
sum, 222
SUPER.f, 617
supernode, 592
sva, 538
svd, 538
svplot, 365
sylm, 499
sylv, 539
symbols, 72
sysconv, 223
sysdiag, 224
sysfact-, 366
sylv, 224
sysize, 367
system, 478
systems, 407
stmat, 499
T
tan, 225
TANBLK.f, 617
tangent, 408
tanh, 226
tanhm, 226
tanm, 227
TCLSS.f, 617
tdinit, 409
testmatrix, 73
texprint, 665
TEXT.f, 618
tf2des, 390
tf2ss, 367
then, 73
tilda, 73
TIME.f, 618
time_id, 367
timer, 315
titlepage, 134
TK_EvalFile, 649
TK_EvalStr, 650
TK_GetVar, 651
TK_SetVar, 651
tlist, 73
toeplitz, 227
trace, 539
trans, 478
trans_closure, 592
translatepaths, 665
TRASH.f, 618
trfmod, 228
trianfml, 228
tril, 228
trisolve, 229
triu, 229
trzeros, 368
type, 74
typename, 75
typeof, 229
U
uiobserver, 369
uilcontrol, 655
uimenu, 657
uint16, 188
uint32, 188
uint8, 188
ulink, 309
union, 230
unique, 230
unix, 310
unix.g, 310
unix.s, 311
unix.w, 311
unix.x, 312
unobs, 370
unsetmenu, 290
user, 75
V
varargin, 277
WaitMsg Scilab function

varargout, 277
delvar, 277

W
WaitMsg, 689
warning, 261
wavread, 635
wavwrite, 635
WFILE_f, 618
w fir, 479
what, 76
where, 76
whereami, 76
whereis, 77
while, 77
who, 77
whos, 78
wiener, 479
wigner, 480
window, 480
winsid, 134
writb, 261
write, 262
write4b, 262
WRITEC_f, 619

X
xchoices, 291
xcho, 291
xdialog, 292
xmatrix, 292
xmdialog, 293
xmessage, 293
xmessage_modeless, 294
xarc, 134
xarcs, 135
xarrows, 135
xaxis, 136
xbasc, 137
xbasimp, 137
xbasr, 138
xchange, 138
xclea, 138
xeclear, 139
xclick, 139
xclip, 140
xdel, 140
xend, 141
xf arc, 141
xfarcs, 142
xfpoly, 142
xfpolys, 143
xrect, 143
xget, 144
xgetech, 145

Y
yulewalk, 480

Z
ZCROSS_f, 619
zeropen, 371
zeros, 231
zgrid, 160
zp butt, 481
zpch1, 481
zpch2, 482
zp ell, 482