rlab + rlabplus

© Marijan Koštrun
mkostrun@gmail.com
https://rlabplus.sourceforge.net
November 30, 2021
rlabplus legal stuff

This is rlabplus, a collection of numerical routines for scientific computing and other functions for rlab2 and rlab3 scripting languages. rlabplus is free software, you can redistribute it and/or modify it under the terms of the GNU General Public License.

The GNU General Public License does not permit this software to be redistributed in proprietary programs. However, releasing closed-source add-ons with explicit purpose of making money while making world a better place through solving complex scientific problems, is encouraged.

This collection is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Citation Info

If you find rlab2 or rlab3 useful, please cite them in your research work as following:

In Memoriam

## Contents

### I What is rlab?

#### 1 Introduction

1.1 About
1.2 About this Document
1.3 What is new in rlab3 compared to rlab2?
1.4 Examples of New Constructs
   1.4.1 if-else
   1.4.2 for-then, while-then
   1.4.3 do-while/until
   1.4.4 switch
   1.4.5 Modifiers +=, -=, *= and /=
   1.4.6 function-static variables
   1.4.7 EOF
   1.4.8 script
   1.4.9 classdef
   1.4.10 Future Developments
1.5 On the shoulders of
1.6 Commercial development

#### 2 Getting rlabplus to Run on a Linux System

2.1 Obtaining the Latest Source Code
2.2 Building rlabplus
2.3 gfortran vs. f77/g77, 32 vs. 64 bit
2.4 Examples and Tests
2.5 Default user paths

#### 3 Getting rlab to work for you

3.1 Basic types of variables
3.2 Matrix Optimization

### II Solvers

#### 4 Controlling rlab from command line

4.1 Command Line Input
4.2 Accessing Command Line Options Inside rlab
   4.2.1 argv
4.2.2 basename .................................................. 46
4.2.3 isinteractive ........................................... 46

5 Miscellaneous ................................................. 47
5.1 Input/output/control functions .................................. 47
  5.1.1 stat .................................................. 47
  5.1.2 inputs ............................................... 47
  5.1.3 prompt ............................................. 48
  5.1.4 randch ............................................. 48
  5.1.5 spinner ............................................ 49
  5.1.6 smiley ............................................. 49
  5.1.7 getmatrix ......................................... 49
  5.1.8 hostname ......................................... 49
  5.1.9 email ............................................. 50
  5.1.10 ls ................................................ 50
  5.1.11 mkdir ............................................ 50
  5.1.12 mv ............................................... 50
  5.1.13 pwd .............................................. 50
  5.1.14 touch ........................................... 50
  5.1.15 grep ............................................. 51
  5.1.16 rlab initstderr ..................................... 51
  5.1.17 view ............................................. 51
  5.1.18 edit ............................................... 52
  5.1.19 isfile ........................................... 52
  5.1.20 isdir ............................................ 52
  5.1.21 islink ........................................... 52
  5.1.22 isblkspec ......................................... 52
  5.1.23 ischarspec ....................................... 53

5.2 General purpose functions .................................... 54
  5.2.1 names ............................................. 54
  5.2.2 range ............................................. 54
  5.2.3 last, lastr, lastc .................................... 54

5.3 File locking functions ....................................... 55
  5.3.1 getpid ............................................ 55
  5.3.2 lock .............................................. 55
  5.3.3 unlock ............................................ 55

5.4 Timer Related Functions .................................... 56
  5.4.1 sleep ............................................. 56
  5.4.2 tic ................................................. 56
  5.4.3 toc ............................................... 56

5.5 Functions for Manipulating Dates ............................ 57
  5.5.1 seconds ........................................... 57
  5.5.2 clock ............................................. 57
  5.5.3 etime ............................................. 57
  5.5.4 gmtime ........................................... 58
  5.5.5 time2dstr .......................................... 58
  5.5.6 dstr2time ......................................... 58
  5.5.7 timezone ......................................... 59
5.5.8 datesinrange ................................................................. 59
5.5.9 dayofweek, dayofmonth, dayofyear .................................. 59
5.5.10 isleapyear ............................................................... 60
5.5.11 monthofyear, weekofyear .............................................. 60
5.5.12 daysinmonth, daysinyear, weeksinyear ............................... 60
5.5.13 date2jd, date2mjd, jd2mjd ............................................. 60
5.5.14 yearnum ............................................................... 61
5.5.15 otherday ............................................................... 61
5.5.16 earliertime, latertime .................................................. 61
5.5.17 days2hms, hms2days .................................................. 61
5.5.18 hms2sec, sec2hms ..................................................... 62
5.5.19 jd2date, jdate2jd, jd2mjd, mjd2jd .................................... 62
5.5.20 isdate, isjdate ........................................................ 62
5.6 Indices, row and column manipulations .................................. 64
5.6.1 unique ............................................................... 64
5.6.2 union ............................................................... 64
5.6.3 intersect ............................................................ 64
5.6.4 complement .......................................................... 64
5.6.5 merge ............................................................... 65
5.6.6 compact ............................................................. 66
5.6.7 rmrows .............................................................. 67
6 Data Plotting and Export .................................................... 69
6.1 Gnuplot ................................................................. 70
6.1.1 gnustart ............................................................. 70
6.1.2 gnuclose ............................................................. 71
6.1.3 gnunix .............................................................. 71
6.1.4 gnunixs .............................................................. 71
6.1.5 gnudefault .......................................................... 72
6.1.6 gnuxtext ............................................................ 73
6.1.7 gnultitle ............................................................. 74
6.1.8 gnuxtics, gnux2tics, gnuytics, gnuy2tics, gnuztics ................... 74
6.1.9 gnulimits, gnulimits2 ................................................ 74
6.1.10 gnuscale, gnuscale2 ................................................ 74
6.1.11 gnuxlabel, gnuxylabel, gnualabel ................................ 75
6.1.12 gnulegend .......................................................... 75
6.1.13 gnufomat ........................................................... 75
6.1.14 gnurect ............................................................. 75
6.1.15 gnuplot ............................................................. 76
6.1.16 gnusplot ............................................................ 76
6.1.17 gnucmd ............................................................. 77
6.1.18 gnuprint ............................................................ 77
6.2 Pgopt ................................................................. 90
6.2.1 plwins ............................................................... 91
6.2.2 plwin ............................................................... 91
6.2.3 pltext ............................................................... 92
6.2.4 pltitle ............................................................... 92
6.2.5 plxtics, plx2tics, plytics, ply2tics .................................. 92
6.2.6 plimits, plimits2
6.2.7 pscale, pscale2
6.2.8 plgrid, plxgrid, plygrid, plxgrid2, plygrid2
6.2.9 plxlabel, plx2label, plylabel, ply2label
6.2.10 plegend
6.2.11 plformat
6.2.12 plot

6.3 Plplot
6.3.1 plwins
6.3.2 plwin
6.3.3 pltext
6.3.4 ptitle
6.3.5 plxtics, plx2tics, plytics, ply2tics, plztics
6.3.6 plimits, plimits2
6.3.7 pscale, pscale2
6.3.8 plgrid, plgrid2
6.3.9 pixlabel, plx2label, plylabel, ply2label, plzlabel
6.3.10 gnulegend
6.3.11 gnuformat
6.3.12 gnuplot
6.3.13 gnusplot
6.3.14 gnucmd
6.3.15 gnuprint

6.4 Xfig
6.4.1 xfig
6.4.2 comment
6.4.3 justification
6.4.4 orientation
6.4.5 units
6.4.6 papersize
6.4.7 resolution
6.4.8 color
6.4.9 world
6.4.10 label
6.4.11 ellipse
6.4.12 arc
6.4.13 line
6.4.14 picture
6.4.15 export

6.5 Xmgrace
6.5.1 xmgcolormap
6.5.2 xmgtext
6.5.3 xmgline
6.5.4 xmpaper
6.5.5 xmgfilename
6.5.6 xmgtitle
6.5.7 xmgsubtitle
6.5.8 xmgxylabs
III  Builtin-type functions

7  General Purpose Functions

7.1  Miscellaneous mathematical and other functions

7.1.1  iseed
7.1.2  hash
7.1.3  crc32
7.1.4  showkey
7.1.5  floor, ceil, round
7.1.6  ifelse
7.1.7  swap
7.1.8  flipud, fliplr
7.1.9  shiftl, shiftr, shiftd, shiftu
7.1.10  rot90
7.1.11  polyval
7.1.12  polyder
7.1.13  polyint
7.1.14  conv
7.1.15  deconv
7.1.16  polyroots
7.1.17  poly
7.1.18  nseries
7.1.19  mexp
7.1.20  mpow
7.1.21  cosh
7.1.22  sinh
7.1.23  tanh
7.1.24  acosh
7.1.25  asinh
7.1.26  atanh

7.2  Miscellaneous byte-wise functions

7.2.1  bytejoin
CONTENTS

7.3  
    7.3.1 ascen  ................................................................. 161
    7.3.2 char  ................................................................. 161
    7.3.3 blank  .............................................................. 162
    7.3.4 num2str,text  ...................................................... 162
    7.3.5 strlen  .............................................................. 163
    7.3.6 lstrip  .............................................................. 163
    7.3.7 rstrip  .............................................................. 164
    7.3.8 strip  ............................................................... 164
    7.3.9 strtod  ............................................................. 164
    7.3.10 strindex ........................................................... 166
    7.3.11 strindexr .......................................................... 167
    7.3.12 grep  ............................................................... 167
    7.3.13 findvec ............................................................ 167
    7.3.14 substr  ............................................................ 167
    7.3.15 gsub  .............................................................. 168
    7.3.16 tolower ............................................................ 169
    7.3.17 toupper ............................................................. 169
    7.3.18 unpack ............................................................. 170

7.4  
    7.4.1 clrscr ............................................................... 173
    7.4.2 mvcrsr ............................................................... 173
    7.4.3 clrpos ............................................................... 173
    7.4.4 colors .............................................................. 173

7.5  
    7.5.1 open ............................................................... 177
    7.5.2 close ............................................................... 178
    7.5.3 writem ............................................................. 179
    7.5.4 readm .............................................................. 181
    7.5.5 write ............................................................... 183
    7.5.6 read ............................................................... 183
    7.5.7 fwrite ............................................................. 184
    7.5.8 fread ............................................................... 184
    7.5.9 reads .............................................................. 184
    7.5.10 readmm ............................................................ 185
    7.5.11 writemm .......................................................... 185
    7.5.12 h5ls ............................................................... 186
    7.5.13 h5mv .............................................................. 186
    7.5.14 h5cp .............................................................. 187
    7.5.15 h5ln .............................................................. 187

8  
    8.1  
        8.1.1 hist ........................................................... 196
        8.1.2 hist2 ........................................................... 196
    8.2  
        8.2.1 csplinterp ..................................................... 198
        8.2.2 cspleval ........................................................ 198
CONTENTS

8.2.3 interp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 198
8.2.4 interp1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 199
8.2.5 harmsum . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 200
8.3 Wavelet Transforms in one and two dimensions . . . . . . . . . . . . . . . . . . . . . 203
8.3.1 dwt . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 203
8.3.2 idwt . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 203
8.3.3 dwt2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 204
8.3.4 idwt2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 204
8.4 Roots of a real vector function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 205
8.4.1 findroot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 205
8.4.2 findroots . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 207
8.4.3 crvtrack . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 208
8.5 Simulated annealing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 213
8.5.1 siman . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 213
8.5.2 asamin . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 214
8.6 Levin u-transformation for series summation acceleration . . . . . . . . . . . . . . . . . 220
8.6.1 levinu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 220
8.7 Fitting and modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 221
8.7.1 polyfit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 221
8.7.2 lsfit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 221
8.7.3 odrfit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 223
8.8 Numerical integration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 228
8.8.1 nintegrate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 228
8.8.2 nintegrate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 229
8.8.3 cauchypval . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 229
8.8.4 nintqaws . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 230
8.8.5 nintcos, nintsin . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 230
8.8.6 nintlt . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 231
8.8.7 invlt . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 231
8.8.8 nintconv . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 232
8.8.9 nintmd . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 233
8.8.10 nintsimplex . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 233
8.8.11 mcnint . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 235
8.9 Tchebyshev polynomials and differentiation . . . . . . . . . . . . . . . . . . . . . . . . . . 237
8.9.1 tchebyfit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237
8.9.2 tchebyval . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237
8.9.3 tchebwint . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237
8.9.4 tchebyder . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237
8.9.5 ndiff . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 238
8.10 Numerical differentiation of a real vector function . . . . . . . . . . . . . . . . . . . . . . 239
8.10.1 ndiffs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 239
8.10.2 ndiv . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 239
8.11 Minimization of a real function of a real vector variable . . . . . . . . . . . . . . . . . . . 240
8.11.1 findmin . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 240
8.11.2 diffevol . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 240
8.11.3 findmins . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 242
8.11.4 conmins . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 244
8.12 Statistics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 249
## 8.12 Statistical Functions

- 8.12.1 `isstat` ................................................................. 250
- 8.12.2 `mean` ................................................................. 250
- 8.12.3 `var` ................................................................. 250
- 8.12.4 `min`, `max`, `minmax`, `min_max` ......................... 251
- 8.12.5 `covar` ............................................................... 252
- 8.12.6 `absdev` ............................................................. 252
- 8.12.7 `skew` ............................................................... 253
- 8.12.8 `kurtosis` .......................................................... 253
- 8.12.9 `median` ............................................................ 253
- 8.12.10 `quantile` .......................................................... 254
- 8.12.11 `invperm` .......................................................... 254
- 8.12.12 `revperm` .......................................................... 254
- 8.12.13 `nextperm` .......................................................... 254
- 8.12.14 `prevperm` .......................................................... 255
- 8.12.15 `validperm` .......................................................... 255
- 8.12.16 `mulperm` ............................................................. 255
- 8.12.17 `nextcomb` .......................................................... 255
- 8.12.18 `prevcomb` .......................................................... 255
- 8.12.19 `validcomb` .......................................................... 255
- 8.12.20 `ncomb` ............................................................... 256

## 8.13 Random Number Generators

- 8.13.1 `irng` ............................................................... 257
- 8.13.2 `irand` ............................................................... 258
- 8.13.3 `irng.state` ............................................................. 258
- 8.13.4 `rng` ................................................................. 260
- 8.13.5 `rand` ................................................................. 261
- 8.13.6 `randomize` ............................................................ 262
- 8.13.7 `urandom` ............................................................. 262
- 8.13.8 `gaussian` ............................................................. 265
- 8.13.9 `drng` ................................................................. 265
- 8.13.10 `drand` ............................................................... 266
- 8.13.11 `hrng` ............................................................... 266
- 8.13.12 `hrand` ............................................................... 267
- 8.13.13 `shuffle` .............................................................. 267
- 8.13.14 `sample` .............................................................. 267
- 8.13.15 `pdf`, `cpf` .......................................................... 267

## 8.14 Ordinary Differential Equations (ODE)

- 8.14.1 `odeiv` ............................................................... 270
- 8.14.2 `odeby` ............................................................... 272
- 8.14.3 `odeset` ............................................................... 274
- 8.14.4 `odeshlt` .............................................................. 275

## 8.15 Special Functions

- 8.15.1 `AiryAi` ............................................................... 291
- 8.15.2 `AiryAiPrime` ......................................................... 291
- 8.15.3 `AiryZeroAi` .......................................................... 291
- 8.15.4 `AiryZeroAiPrime` .................................................. 291
- 8.15.5 `AiryBi` ............................................................... 292
- 8.15.6 `AiryBiPrime` .......................................................... 292
CONTENTS

8.15.7 AiryZeroBi ................................................. 292
8.15.8 AiryZeroBiPrime ........................................... 292
8.15.9 BesselI .................................................. 292
8.15.10 BesselJ .................................................. 293
8.15.11 BesselZeroJ .............................................. 293
8.15.12 BesselK .................................................. 293
8.15.13 BesselY .................................................. 293
8.15.14 Besseli .................................................. 294
8.15.15 Besselj .................................................. 294
8.15.16 Besselk .................................................. 294
8.15.17 Bessely .................................................. 294
8.15.18 Clausen .................................................. 294
8.15.19 Hydrogen ................................................. 294
8.15.20 Dawson .................................................. 295
8.15.21 Debye ................................................... 295
8.15.22 EllipticD ............................................... 295
8.15.23 EllipticE ............................................... 296
8.15.24 EllipticF ............................................... 296
8.15.25 EllipticJacobi .......................................... 296
8.15.26 EllipticK ............................................... 297
8.15.27 EllipticP ............................................... 297
8.15.28 EllipticRC, EllipticRD, EllipticRF, EllipticRJ ............ 297
8.15.29 Erf ...................................................... 297
8.15.30 Erfc .................................................... 298
8.15.31 ErfQ .................................................... 298
8.15.32 ErfZ .................................................... 298
8.15.33 ExpIntegralE ........................................... 298
8.15.34 ExpIntegralEi ......................................... 299
8.15.35 SinhIntegral ........................................... 299
8.15.36 CoshIntegral ........................................... 299
8.15.37 SinIntegral .............................................. 300
8.15.38 CosIntegral .............................................. 300
8.15.39 AtanIntegral ............................................ 300
8.15.40 FermiDiracIntegral .................................... 300
8.15.41 Beta ..................................................... 301
8.15.42 Beta ..................................................... 301
8.15.43 Gamma .................................................. 301
8.15.44 GammaRegularized ...................................... 301
8.15.45 GammaRegularizedC ..................................... 301
8.15.46 LogGamma ............................................... 302
8.15.47 RiemannGamma ......................................... 302
8.15.48 Pochhammer .............................................. 302
8.15.49 GegenbauerC ............................................. 302
8.15.50 Hypergeometric0F1 ..................................... 303
8.15.51 Hypergeometric1F1 ..................................... 303
8.15.52 Hypergeometric2F1 ..................................... 303
8.15.53 Hypergeometric2F0 ..................................... 303
8.15.54 Hypergeometric2F0 ..................................... 303
8.15.55 LaguerreL ......................................................... 304
8.15.56 ProductLog ....................................................... 304
8.15.57 LegendreP, LegendreQ ......................................... 304
8.15.58 LegendreSphericalP ............................................. 305
8.15.59 LegendreConicalP .............................................. 305
8.15.60 LegendreH3d ..................................................... 305
8.15.61 Digamma ......................................................... 305
8.15.62 PolyGamma ....................................................... 306
8.15.63 SynchrotronF .................................................... 306
8.15.64 TransportF ....................................................... 306
8.15.65 Zeta ............................................................ 306

9 Special Topics Solvers ................................................. 309
9.1 Time Series Analysis (TISEAN) .................................... 309
  9.1.1 rmap ............................................................ 309
  9.1.2 xcorr ........................................................... 309
  9.1.3 runavg ............................................................ 310
  9.1.4 runvar ............................................................ 310
  9.1.5 ami ............................................................... 311
  9.1.6 falsenn .......................................................... 311
  9.1.7 poincare ........................................................ 311
9.2 The Gnu Linear Programming Kit (GLPK) ......................... 314
  9.2.1 glpk_read ....................................................... 314
  9.2.2 glpk_write ....................................................... 315
  9.2.3 glpk_solve ...................................................... 315
9.3 Signal Processing Toolkit ........................................... 320
  9.3.1 filter ............................................................ 320
  9.3.2 harminv .......................................................... 321
  9.3.3 stl ............................................................... 322
  9.3.4 window ............................................................ 322
  9.3.5 fft ............................................................... 323
  9.3.6 fftshift .......................................................... 323
  9.3.7 frid ............................................................... 323
  9.3.8 frid ............................................................... 324
  9.3.9 bsplineedit ...................................................... 324
  9.3.10 bsplineval ...................................................... 325
  9.3.11 bsplineedit2 ................................................... 326
  9.3.12 bsplineval2 ..................................................... 327
  9.3.13 gcvsplineedit ................................................ 327
  9.3.14 gcvsplineval .................................................. 328
9.4 Local Regression Models Fitting (loess) .......................... 330
  9.4.1 loess ............................................................ 330

10 Partial Differential Equations in 1-D ............................ 335
10.1 Introduction .......................................................... 335
10.2 CLAWPACK .......................................................... 335
  10.2.1 claw ............................................................ 335
  10.2.2 clawparams ..................................................... 338
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.3</td>
<td>EPDCOL and BACOL</td>
<td>342</td>
</tr>
<tr>
<td>10.3.1</td>
<td>pdecol</td>
<td>342</td>
</tr>
<tr>
<td>10.3.2</td>
<td>bacol</td>
<td>343</td>
</tr>
<tr>
<td>11</td>
<td>Pattern Recognition</td>
<td>353</td>
</tr>
<tr>
<td>11.1</td>
<td>Patterns, Their Recognition, and Data Analysis</td>
<td>353</td>
</tr>
<tr>
<td>11.1.1</td>
<td>isrelmono, isabsmono</td>
<td>353</td>
</tr>
<tr>
<td>11.1.2</td>
<td>locextri</td>
<td>353</td>
</tr>
<tr>
<td>11.1.3</td>
<td>distance</td>
<td>355</td>
</tr>
<tr>
<td>11.1.4</td>
<td>cluster</td>
<td>355</td>
</tr>
<tr>
<td>11.1.5</td>
<td>classify</td>
<td>356</td>
</tr>
<tr>
<td>11.1.6</td>
<td>sammon</td>
<td>356</td>
</tr>
<tr>
<td>12</td>
<td>3-D Surfaces</td>
<td>359</td>
</tr>
<tr>
<td>12.1</td>
<td>GTS</td>
<td>359</td>
</tr>
<tr>
<td>12.1.1</td>
<td>validgts</td>
<td>360</td>
</tr>
<tr>
<td>12.1.2</td>
<td>_gts_to_xyz</td>
<td>360</td>
</tr>
<tr>
<td>12.1.3</td>
<td>plot_gts_surface</td>
<td>360</td>
</tr>
<tr>
<td>12.1.4</td>
<td>snapshot</td>
<td>361</td>
</tr>
<tr>
<td>12.1.5</td>
<td>ls</td>
<td>361</td>
</tr>
<tr>
<td>12.1.6</td>
<td>info</td>
<td>361</td>
</tr>
<tr>
<td>12.1.7</td>
<td>stat</td>
<td>361</td>
</tr>
<tr>
<td>12.1.8</td>
<td>qstat</td>
<td>362</td>
</tr>
<tr>
<td>12.1.9</td>
<td>new</td>
<td>362</td>
</tr>
<tr>
<td>12.1.10</td>
<td>from_vertices</td>
<td>363</td>
</tr>
<tr>
<td>12.1.11</td>
<td>readm,writem</td>
<td>364</td>
</tr>
<tr>
<td>12.1.12</td>
<td>export</td>
<td>364</td>
</tr>
<tr>
<td>12.1.13</td>
<td>foreach_vertex, foreach_edge, foreach_face</td>
<td>366</td>
</tr>
<tr>
<td>12.1.14</td>
<td>foreach_face_remove</td>
<td>366</td>
</tr>
<tr>
<td>12.1.15</td>
<td>add_face</td>
<td>367</td>
</tr>
<tr>
<td>12.1.16</td>
<td>remove_face</td>
<td>367</td>
</tr>
<tr>
<td>12.1.17</td>
<td>remove_surface</td>
<td>368</td>
</tr>
<tr>
<td>13</td>
<td>Rlab2 Rel.3 Internals</td>
<td>369</td>
</tr>
<tr>
<td>13.1</td>
<td>Sparse Matrices</td>
<td>369</td>
</tr>
<tr>
<td>13.1.1</td>
<td>sparms</td>
<td>370</td>
</tr>
<tr>
<td>13.1.2</td>
<td>spsolve</td>
<td>371</td>
</tr>
<tr>
<td>13.2</td>
<td>Changes in Linear Algebra Routines</td>
<td>372</td>
</tr>
<tr>
<td>13.2.1</td>
<td>eig</td>
<td>372</td>
</tr>
<tr>
<td>13.2.2</td>
<td>eigs</td>
<td>373</td>
</tr>
<tr>
<td>14</td>
<td>New Data Types</td>
<td>375</td>
</tr>
<tr>
<td>14.1</td>
<td>Integers</td>
<td>375</td>
</tr>
<tr>
<td>14.2</td>
<td>Ordered Lists - Experimental Feature</td>
<td>376</td>
</tr>
<tr>
<td>14.2.1</td>
<td>cell</td>
<td>376</td>
</tr>
<tr>
<td>14.2.2</td>
<td>vcell</td>
<td>376</td>
</tr>
<tr>
<td>14.2.3</td>
<td>hcell</td>
<td>377</td>
</tr>
<tr>
<td>14.2.4</td>
<td>iscell</td>
<td>379</td>
</tr>
</tbody>
</table>
14.3 Protected Lists ................................. 381
  14.3.1 protect .................................. 381
  14.3.2 release .................................. 381
  14.3.3 isprot .................................. 381

14.4 Subroutines ................................. 383
  14.4.1 assign .................................. 383

14.5 System Constants ......................... 384
  14.5.1 mks - a list of physical constants and physical unit conversion factors ................................. 384
  14.5.2 const - a list of mathematical constants .................................. 388

IV Miscellaneous Topics .......................... 389

15 Embedded Interpreters ...................... 391
  15.1 ngspice - Electronic Circuit Simulator .......... 391
    15.1.1 exec .................................. 391
    15.1.2 raw .................................. 392
    15.1.3 init .................................. 392
    15.1.4 cmd .................................. 392
    15.1.5 circuit ................................ 392
    15.1.6 runckt ................................ 393
    15.1.7 getvals ................................ 393
    15.1.8 isrunning ................................ 393
    15.1.9 kill .................................. 393
  15.2 Python .................................. 400
    15.2.1 init .................................. 400
    15.2.2 kill .................................. 401
    15.2.3 ver .................................. 401
    15.2.4 file .................................. 401
    15.2.5 cmd .................................. 401
    15.2.6 var .................................. 401
    15.2.7 eval .................................. 402
  15.3 JVM .................................. 404
    15.3.1 create ................................ 404
    15.3.2 destroy ................................ 404
    15.3.3 call_method ................................ 404

16 Cameras and Images ......................... 407
  16.1 GPhoto2 .................................. 407
    16.1.1 supported .................................. 407
    16.1.2 init .................................. 408
    16.1.3 exit .................................. 408
    16.1.4 info .................................. 408
    16.1.5 config_options .................................. 409
    16.1.6 config .................................. 410
    16.1.7 capture ................................ 410
  16.2 ImageMagick ................................ 412
    16.2.1 init .................................. 413
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.2.2 exit</td>
<td>413</td>
</tr>
<tr>
<td>16.2.3 wand</td>
<td>413</td>
</tr>
<tr>
<td>16.2.4 write</td>
<td>413</td>
</tr>
<tr>
<td>16.2.5 read</td>
<td>414</td>
</tr>
<tr>
<td>16.2.6 iter</td>
<td>414</td>
</tr>
<tr>
<td>16.2.7 clone</td>
<td>414</td>
</tr>
<tr>
<td>16.2.8 clear</td>
<td>414</td>
</tr>
<tr>
<td>16.2.9 close</td>
<td>415</td>
</tr>
<tr>
<td>16.2.10 join</td>
<td>415</td>
</tr>
<tr>
<td>16.2.11 append</td>
<td>415</td>
</tr>
<tr>
<td>16.2.12 combine</td>
<td>415</td>
</tr>
<tr>
<td>16.2.13 diff</td>
<td>416</td>
</tr>
<tr>
<td>16.2.14 composite</td>
<td>416</td>
</tr>
<tr>
<td>16.2.15 deconstruct</td>
<td>417</td>
</tr>
<tr>
<td>16.2.16 distort</td>
<td>417</td>
</tr>
<tr>
<td>16.2.17 clut</td>
<td>417</td>
</tr>
<tr>
<td>16.2.18 pixels</td>
<td>417</td>
</tr>
<tr>
<td>16.2.19 f</td>
<td>418</td>
</tr>
<tr>
<td>16.2.20 method</td>
<td>418</td>
</tr>
<tr>
<td>16.2.21 specs</td>
<td>420</td>
</tr>
<tr>
<td>16.2.22 art</td>
<td>420</td>
</tr>
<tr>
<td>16.2.23 disp</td>
<td>421</td>
</tr>
<tr>
<td>16.2.24 method</td>
<td>421</td>
</tr>
<tr>
<td>16.3 Random topics unfit for other sections</td>
<td>426</td>
</tr>
<tr>
<td>16.3.1 sep</td>
<td>426</td>
</tr>
</tbody>
</table>

### 17 Other Hardware Interfaces

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.1 General Purpose Interface Bus (GPIB)</td>
<td>427</td>
</tr>
<tr>
<td>17.1.1 ibdev</td>
<td>427</td>
</tr>
<tr>
<td>17.1.2 ibfind</td>
<td>428</td>
</tr>
<tr>
<td>17.1.3 ibonl</td>
<td>428</td>
</tr>
<tr>
<td>17.1.4 ibcll</td>
<td>428</td>
</tr>
<tr>
<td>17.1.5 ibsrc</td>
<td>428</td>
</tr>
<tr>
<td>17.1.6 ibsic</td>
<td>429</td>
</tr>
<tr>
<td>17.1.7 ibhm</td>
<td>429</td>
</tr>
<tr>
<td>17.1.8 ibwrt</td>
<td>430</td>
</tr>
<tr>
<td>17.1.9 ibrd</td>
<td>430</td>
</tr>
<tr>
<td>17.1.10 ibqrd</td>
<td>430</td>
</tr>
</tbody>
</table>

### V References

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>References</td>
<td>437</td>
</tr>
<tr>
<td>References</td>
<td>439</td>
</tr>
</tbody>
</table>

### VI Appendices

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.2 Pattern Matching</td>
<td>443</td>
</tr>
<tr>
<td>17.2 Pattern Matching</td>
<td>445</td>
</tr>
</tbody>
</table>
Part I

What is rlab?
Chapter 1

Introduction

1.1 About

rlab is an interactive, interpreted scientific programming environment for fast numerical prototyping and program development. rlab is a very high level language intended to provide fast prototyping and program development, as well as easy data-visualization, and processing. While rlab is not a clone of languages such as those used by tools like Matlab or Matrix-X/Xmath, it can be called “Matlab-like.”

Overall, since rlab2 rel.1 the focus was on creating a good experimental environment (or laboratory) in which to do matrix math, with minimal learning curve. This continues with with the latest rlab3-programming language, which adds more programming constructs from c-language and python, while resolving some lexical idiosyncrasies of rlab2 in favor of, again, c-language.

by Ian Searle (2001) and Marijan Kostrun (2016).

1.2 About this Document

This manual is intended as a continuation of rlab2 Primer by Ian Searle. It implies that the reader has familiarized themselves with the basics of rlab programming language, and has some knowledge of c-language and python.
### 1.3 What is new in rlab3 compared to rlab2?

<table>
<thead>
<tr>
<th>rlab2</th>
<th>rlab3</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>if(cond){ ⋯ else ⋯}</td>
<td>if(cond){ ⋯ else if(cond){ ⋯ }</td>
<td>c-language</td>
</tr>
<tr>
<td></td>
<td>else { ⋯ }</td>
<td></td>
</tr>
<tr>
<td>for(iter){ ⋯ }</td>
<td>for(iter){ ⋯ then: ⋯ }</td>
<td>python (then→else)</td>
</tr>
<tr>
<td>while(cond){ ⋯ }</td>
<td>while(cond){ ⋯ then: ⋯ }</td>
<td>python (then→else)</td>
</tr>
<tr>
<td></td>
<td>do{ ⋯ }</td>
<td>c-language</td>
</tr>
<tr>
<td></td>
<td>while(cond)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>do{ ⋯ }</td>
<td>m*tlab, octave</td>
</tr>
<tr>
<td></td>
<td>until(cond)</td>
<td></td>
</tr>
<tr>
<td>switch (expr0){</td>
<td></td>
<td>c-language, however expr’s</td>
</tr>
<tr>
<td></td>
<td>case expr1: ⋯</td>
<td>can be anything that</td>
</tr>
<tr>
<td></td>
<td>default : ⋯</td>
<td>can be compared to (expr0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c-language elemental operations that avoid intermediate storage</td>
</tr>
<tr>
<td>static (⋯) is used for file-static variables</td>
<td>static(⋯) is for function- and file-static variables (former preceeds the latter)</td>
<td>c-language for static statement inside function declaration</td>
</tr>
<tr>
<td></td>
<td>EOF</td>
<td>stops execution of current file, and returns the control to calling file or prompt.</td>
</tr>
<tr>
<td></td>
<td>script {⋯}</td>
<td>specifies rlab script without using string designators.</td>
</tr>
<tr>
<td></td>
<td>classdef (⋯){⋯}</td>
<td>allows programmatic construction of class members with their properties and methods. Supported are private(⋯) for class-private, public(⋯) for class-member public, and static(⋯) for class-member private entities.</td>
</tr>
</tbody>
</table>
1.3. WHAT IS NEW IN RLAB3 COMPARED TO RLAB2?

The programming constructs that have changed between rlab2 and rlab3, and the new constructs that are supported only in rlab3 are shown in the table above.

As scripts for rlab3 might not run in rlab2 it was decided to change the extension of rlab3 scripts to

(rlab3) *.r3,

whereas rlab2 scripts keep the extension *.r.

To convert rlab2 scripts to run under rlab3 the format of if-else command has to be changed:

(rlab2) ‘else’ \rightarrow (rlab3) ‘} else {’.

This will work in 99.5% of the scripts. In the remaining 0.5% the replacement might expose some logical errors in the script. The easiest ways to convert rlab2 scripts to rlab3 is to copy them with the new extension, and then use sed

    sed -i -e 's/ else/ } else {/g' *.r3

For the same reasons, the new rlab executable is appropriately named

rlab3

and it will coexist with the existing rlab2 installation on the system.
1.4  Examples of New Constructs

1.4.1  if-else

In rlab3, the if-else statement follows the c-language definition. Consider the following trivial example:

```c
u = uniform();
if (u > 2/3)
{
    printf("Greater than 2/3\n");
}
else if (u > 1/3)
{
    printf("Between 1/3 and 2/3\n");
}
else
{
    printf("Less then 1/3\n");
}
```

Parser Limitations:

1. The else part of the statement following the if has to be within one-blank line from the end of if-statement. If there is more empty lines between the two, parser will report an error.

2. Between the end of the if-statement that is not followed by else-statement and the end of file there has to be at least one blank line. If this is not obeyed then parser will ignore the if-statement.
1.4.2 for-then, while-then

In rlab3, the for-then statement follows the python definition of for-else and while-else statements. Consider the following two trivial examples:

```plaintext
for (i in 5:101:2)
{
    for (k in 2:ceil(sqrt(i)))
    {
        if (!mod(i,k))
        {
            break;
        }
    }
then:
printf("\t%g is prime number\n", i);
}
```

and

```plaintext
i=5;
while (i<102)
{
    k = 2;
    while (k<=ceil(sqrt(i)))
    {
        if (!mod(i,k))
        {
            break;
        }
    k++;
then:
printf("\t%g is prime number\n", i);
}
i = i + 2;
```

Notes:

1. The loops stretch between their definition and the then: statement. Everything between the then: statement and the end of the loop is executed once, after the last iteration of the loop.

2. The break-statement stops the execution of the loop, and causes a jump to the end of the loop (and so skips the statements following the then:-statement).

3. The continue and break statements past then: to the remainder of the loop body are considered syntax error by the parser. In other words, if the loop is embedded in another loop, the continue and break of the top loop should be outside the body of the embedded loop.
1.4.3  do-while/until

In rlab3, the do-while statement follows the c-language definition. The until statement is introduced for compatibility with octave. Consider the following trivial example:

```c
i=1;
do 
  j=1;
j_end = floor(20*uniform() + 1);
printf("[Test1] count %03.0f for j_end=%2.0f: ", i,j_end);
do
  printf(" %02.0f", j);
s = uniform();
if (s > 0.9)
  { printf("[OOPS s=%.2f, I am out!]", s);
    break;
  }
u until ((j++)>=j_end);
printf("\n");
while ((i++) <= 10);
```

One remembers that while(cond) keeps iterations in the loop for as long as the cond is true. Conversely, until(cond) keeps iterating as long as the cond is false.
1.4.4 switch

In rlab3, the switch statement extends upon the c-language definition. Consider the following trivial example:

```c
u = floor(3*uniform());
a = shuffle([1:2],1);
switch (u)
{
    case 0:
        printf ("u is 0\n");
        break;
    case a:
        printf ("u is %g\n", a);
        break;
    default:
        printf ("u is %g\n", complement(0:2,[0,a]));
}
```

**Notes:**

In rlab3 the switch (var) and case (value) in c-language are expanded to include expressions as in switch (expr) and case (expr). The example above is thus possible where the variable `a` is used as a part of the case statement.

Obvious requirement for the choice of expression is that the logical statement `((u) == (expr))` can be evaluated.
1.4.5 Modifiers +=, -=, *= and /=

In rlab3, the modifiers extend upon the c-language definition. They are introduced as to minimize intermediate storage in element-wise operations on large data sets, e.g., images. Consider the following trivial example:

```
NITER = 10;
i=1;
x = uniform();
do
{
y = gaussian(1,100);
x+=y;
}
while( (i++) < NITER*1e4 );
```

Notes:
In the above example $x$ is initially a scalar. Thus in the first iteration the operation performed is $x = x + y$, where initial value of $x$ is discarded after addition. In the second and higher iterations storage for $x$ is large enough to store the results of computations, henceforth the true $x+ = y$ is calculated without storing the intermediate result.
1.4.6 function-static variables

In rlab3, the static(...) statement inside function-body indicates that the content of these variables will be preserved between function calls. This fully follows the c-language definition. Consider the following trivial example:

```plaintext
// some rfile containing functions definitions, perhaps a library?
//
static(i);
i = 0;

f = function()
{
    return i;
};

g = function()
{
    static(i);
    if (!exist(i))
    { i = -1; }
    i++;
    return i;
};

h = function()
{
    static(i);
    if (!exist(i))
    { i = 1; }
    i--;
    return i;
};

g2 = copy(g);
```

Notes:
In the above example there are two static variables i. The first is file-static which is available to all functions defined in the same rfile as a global variable (does not need to be specified in the function-body, see f()). The second is function-static, which is available only to function g(). For that reason, i inside g() and h() are completely different.

Please remember, the function-static preceeds the file-static declaration.
If one needs the second function doing the same thing, in the example above this is g2, but with its own set of static variables, then one needs to use copy function to duplicate the function, and not assignment operator.
1.4.7 EOF

In rlab3, the EOF token allows user to programmatically instruct the parser/interpreter to stop processing the current file and return the control either to the script that called the rfile or the prompt. One of the best usages of the EOF token is to avoid reloading the same libraries from the script that is being ran multiple times from the command prompt. Consider the following trivial example:

```c
// some rfile containing functions definitions, perhaps a library?

//
static(INIT);
if (exist(INIT))
{ EOF }

// function definitions follow

//
g = function()
{
  static(i);
  if (!exist(i))
  { i = -1; }
  i++;
  return i;
}

//
// end of function definitions

//
INIT = 1;
```

Note: In the above example static(INIT) tells the interpreter that INIT is (or is going to be) a file-static variable so it knows where to look for its value. Then, if it knows the value of INIT it exits the file, or otherwise processes the rest of the file. As the rest of the file includes defining INIT, on second execution of the same rfile the interpreter will exit the file at EOF token and not re-read function definitions again. The same can be done using the if-statement, but this may hinder the usability and maintainability of the code.

Note: The EOF token operates on parser level, so it has to be placed in the file of interest, and e.g., cannot be a result of function call.
1.4.8 script

In rlab3, the script environment allows placing script inside curly brackets for later execution using eval command. Consider a rfile containing the following lines:

```
static(a); /* file static variable */
instrument = script
{
  << ...
  f = function(s)
  {
    static(i); /* function static variable */
    if (exist(s))
      { i = s; }
    return a+i;
  }; ...
  a = function(x)
  {
    if (exist(x))
      { a = x; }
    return a;
  }; ...
  >>
};

x = eval(instrument);
x.a(1);

y = eval(instrument);
y.a()
```

The script environment is intended for manipulating the rlab code programmatically: It is easier to cut-and-paste chunk of code between the curly brackets, then to write it down as a string matrix in which every line is surrounded by the quotation marks and ends with "...".

Executing this rfile sets x and y to be lists <a; f >, where x.a and y.a are the functions that control the value of the file-static variable a that is common for both functions. On the other hand, the functions x.f and y.f each has its own file-static variable i, which controls the output together with the file-static a.
1.4.9 classdef

In rlab3, the classdef environment allows programmatic creation of class members similar to how it is done in C++. The new type of variable is a user class, which is a constructor that creates members, where the class-private variables are shared within all class members. The constructor may take arguments which may be used for instantiation of some of the class properties and methods. The class member properties are available through its public properties and methods. Consider a rfile containing the following lines:

```r
instrument = classdef(a)
{
  static(s_val); // for class-member private properties/methods
  public(r,s,q,arg1); // class-member public properties/methods
  private(r_val); // class private properties/methods

  // public method to control the class-private variable
  r = function(x)
  {
    if (exist(x))
    {
      r_val = x;
    }
    return r_val;
  }

  // class instantiation values are stored in class-public variable
  // and only then accessed inside class method
  arg1 = a;

  // public method to control the class-member private variable
  s = function(x)
  {
    printf('If you ever wondered arg1 = %s, while s_val = %s\n', arg1, s_val);
    if (exist(x))
    {
      s_val = x;
    }
    return s_val;
  }

  x_init = uniform();
  y_init = gaussian();

  x = instrument(x_init);
  y = instrument(y_init);
}
```

The example above demonstrates how the classdef environment can be used to create class members. Executing this rfile sets `x` and `y` as members of the class instrument, which in effect means that each appears as a list `≪ r; s; q; arg1 ≫`, where `arg1` is the property that contains class-member instantiation variable, `s` is the method that controls the class-member private property `s_val`, while `r` is the public method for accessing the class-private variable `r_val`. Note that class-member instantiation variables, here `a`, are not accessible directly in the class-methods but through public variables, here `arg1`. 
1.5. ON THE SHOULDERS OF

classdef
class1 (arg1, arg2, ...)
private(p1, ...)
member2
static(s1,...)
public(r1,...)
member1
static(s1,...)
public(r1,...)

Figure 1.1: Outline of classdef environment in rlab3 and hierarchy of the instantiation arguments arg1, ..., class-private (p1, ...), class-member public and static variables.

1.4.10 Future Developments

As I see it now, the work on development of rlab3 will continue, while the work on rlab2 will stop, until everything I want to have in rlab3 in regards to structures and constructs is there.

Once that task is completed, I am thinking of combining the two, so that user at the build time decides whether they want one or the other. Keep in mind that at that point the only difference between the two will be in parser treatment of the if-else statement, that is, the two will differ only in their scan.l and rlab_parser.y files and the supporting libraries.

With the release of rlab3 I will stop development work on the rlab2 scripted libraries. All the new libraries will be written using rlab3 syntax exclusively.

1.5 On the shoulders of

rlabplus contains the following libraries

• packages FFTPACK, by Paul Schwarztrauber, ARPACK by Lehoucq, Sorensen, and Yang (1997),
  CLAWPACK by R. J. LeVeque, Langseth, Berger, and Mitran (2003), TISEAN, time series analysis
  software by Hegger et al. (2000), The Gnu Linear Programming Kit, by Makhorin (2006), cURL
  by Stenberg (2009), and PGPLOT, plotting library, by Pearson June 1989;

while to the following libraries it is linked dynamically,

• The GNU Scientific Library, by Galassi et al. (2002);
• sparse packages SuperLU by Demmel, Gilbert, and Li (2003), UMFPACK by Davis (2005) and within
  SuiteSparse library;
• ngspice by NGspice team (2013);
• BLAS, LAPACK;

For plotting and data visualization it can use the following packages (that have to be installed on the
system):

• gnuplot by Williams and Kelley (2004), xmgrace by Turner, Grace Development Team, and Stam-

The rlab releases up to 2.1 are the result of hard work of the following individuals: Mike Brennan, Phillip
Musumeci and Brad Hards, T. S. Yang, Dave Raymond, Matt Wette, Fook Fah Yap, Chris Thewalt, J J
Green, Jeff Layton, Maurice LeBrun and Geoffrey Furnish, Scott Hunziker, Glen Greers, John Peterson,
Don Morris, Dave Beste, and finally the rlab’s creator Ian Searle.

The rlab2.2 through 2.5 are Marijan Koštrun’s idea, which were result of the rlabplus project. Contributions
to source code by Elmar Plischke, Tore Johansson and Ian Searle are acknowledged and appreciated.
Eventually, a need was felt to enrich the language with new programming constructs. This resulted in the
release of rlab3 scripting language.

1.6 Commercial development

See sub-project libgpbib.so on http://rlabplus.sourceforge.net for ideas.
Chapter 2

Getting \textit{rlabplus} to Run on a Linux System

2.1 Obtaining the Latest Source Code

\texttt{rlab2} and \texttt{rlab3} are available for download from \texttt{rlabplus.sourceforge.net} as a source code, or as a prebuilt self-installable tar-archive. As mentioned earlier, the two can coexist on the same system as one executable is \texttt{rlab2} and uses files with the extension \texttt{*.r}, while the other is \texttt{rlab3} which uses files with the extension \texttt{*.r3}.

2.2 Building \textit{rlabplus}

Building \texttt{rlab} from source code requires

I. \textsc{Tools:} gcc, g77 or gfortran.

II. \textsc{Mandatory Libraries:}

general blas lapack ncurses
hdf5 hdf5 hdf5-devel
readline libreadline readline-devel
hash libopenssl libopenssl-devel
gsl gsl-devel libgsl0
curl curl libcurl libcurl-devel

III. \textsc{Optional Libraries:}

num libarpack libglpk glpk-devel superlu
gphoto2 libgphoto2 libexif-devel libgphoto2-devel libusb-1.0-devel
MagickWand MagickWand-6.Q16 MagickCore-6.Q16
python python python-devel
jvm – needs java virtual machine location –
sparse suitesparse (on opensuse linux)

IV. OPTIONAL SOFTWARE:

plot gnuplot, open/libre office, xmgrace, libplplot
eda ngspice (built as shared object library, since ngspice-25, or as executable)
sys firefox with mozrepl package installed, aspell, ...

The tape archive (tar) downloaded from the sourceforge.net site contains the directory rlab-3.x.x-gcc, and the file install.txt which contains this section in an ASCII format.

Building rlab goes according to standard linux mantra.

```
> cd rlab-3.x.x-gcc
> make clean
> ./configure [ put your flags here, or use rconfigure\_* that has all the options ]
> make scanner
> make gc flibs clibs rlab
> sudo make install
```

Later on, if fortran, gc, or c-language libraries are modified, then one uses

```
> make gc    /* re-build the garbage collector */
> make flibs /* re-build fortran libraries */
> make clibs /* re-build c-libraries */
```

and then build and install the rlab executable with

```
> make rlab    /* re-build the rlab executable */
> sudo make install /* just as it names says */
```

When rlabplus is built from source and installed using make install then this also installes the include files necessary for development in the directory /usr/local/include/rlab.

Note

```
> make scanner
```

recreates rlab parser file if such got lost in transport or damaged.

Also note that make check no longer works.

To completely uninstall rlabplus use this:

```
> sudo make uninstall
```
2.3 gfortran vs. f77/g77, 32 vs. 64 bit

Makefile.in has been modified so as rlabplus should compile without problems on opensuse platform. User can communicate her or his fortran compiler through an environment variable $F77.

In opensuse, 32-bit installs in /usr/local/lib while 64-bit in /usr/local/lib64, so that both versions can coexist on the same system. The name of the directory in /usr/local/lib{64} carries the version of the gcc RLaB was built with, so multiple compiled instances can coexist, as well.

2.4 Examples and Tests

For testing and as templates please get the test package from sourceforge.net site and unpack it in your local directory $HOME/rlab. This will create a subdirectory $HOME/rlab/test, which will have a number of subdirectories containing scripts that test different features of rlab, e.g., ordinary differential equations, minimization, numerical integration, gnuplot interface, & c.

2.5 Default user paths

Say your name is Marion Doe, and your home directory is /home/mdoe, then the following two directories are in rlabplus path

/home/mdoe/

 rlab/

 rlab/lib.so/ scripted loaders and shared object libraries,

 rlab/lib.r/ user’s scripted libraries.

As an example of a shared object library and its loader an interested user may consult the sub-project libgpib.
Chapter 3

Getting rlab to work for you

3.1 Basic types of variables

To enter rlab environment on your linux workstation type rlab3 at the command prompt

```
earnonymous@localhost:~$ rlab3

Welcome to RLab3+rlabplus Rel. 3.0.5-gcc for Linux
RLab3 and rlabplus(C) 2004-2016 Marijan Kostrun, RLab(C) 1992-2001 Ian Searle
Please check http://rlabplus.sourceforge.net for the latest news, & c
RLab3 comes with ABSOLUTELY NO WARRANTY: for details type 'help warranty'
New users can type 'help INTRO' to get started
This is free software, and you are welcome to redistribute it under
certain conditions; type 'help conditions' for details

>>
```

To exit the environment, one can press Ctrl+D, or Ctrl+—, or type quit at the prompt.

The following is the list of user variables that can serve as input and output parameters for different solver functions

- real scalars

```
>> x=sqrt(3)
an =
1.73
>> type(x)
an =
‘real’
>>
```

- integer scalars
• real matrices

```matlab
>> x = rand(3,2)
ans =
     1     2
     3     4
     5     6
>> x[3;1]
ans = 5
```

• complex scalars and matrices.

```matlab
>> x = rand() + 1i*rand()
ans =
0.808542952 + 0.212228635i
```

• string scalars and matrices

```matlab
>> x = 'string scalars and matrices'
ans =
string scalars and matrices
>> strplt(x, ' ' ')
ans =
string scalars and matrices
```

• list, identical to shell associative arrays, or to python dictionaries, is a binary-tree type structure, which entries are accessible through their keys, the string scalars,

```matlab
>> options = <<
options.eabs = 1e-3
ans =
eabs
>> options.erel = 0.01
ans =
```
3.1. BASIC TYPES OF VARIABLES

```
>> eabs erel
```

Note that the list entries are sorted using the lexicographic ordering. E.g., a list with entries 1,2,10,21, will appear as

```
>> x=[1;2;10;21]
x =
   1   10   2   21
```

To use a content of a variable, say called “handle”, to access an entry in a list do,

```
>> options
ans =
eabs erel
>> handle = 'eabs'
ans =
eabs
>> options.handle
ans =
 1e-3
```

When defining matrix entries “,” is the column separator, e.g.,

```
>> x=[1,2,3,4]
x =
   1   2   3   4
```

while “;” is the row separator, e.g.,

```
>> x=[1;2;3;4]
x =
   1
   2
   3
   4
```

There are other storage types in rlab but these will be covered later in the manual.

To recall the result of last command line computation use variable name ans.
3.2 Matrix Optimization

The matrix optimization is implemented in almost all functions, which take scalar arguments. Examples of such functions are matrix addition and subtraction, and element division (“./”) and element multiplication (“.*”). Consider an example, in which we add to each row of the matrix \( x \) the row-vector \( y \),

\[
>> x = \text{rand}(4,2) \\
0.442631533 & 0.829417558 \\
0.133380664 & 0.685231976 \\
0.582815987 & 0.311300559 \\
0.725561778 & 0.705769397 \\
>> y = \text{rand}(1,2) \\
0.544930784 & 0.319334006 \\
>> x + y \\
0.987562317 & 1.14875156 \\
0.678311448 & 1.00456598 \\
1.12774677 & 0.630634565 \\
1.27049256 & 1.0251034 \\
>>
\]

The matrix optimization for a scalar operator, to which, say two, matrices are supplied as the arguments is defined as follows.

Let function \( f = f(x_1, x_2) \) be a scalar function of two scalar arguments, say \( x_1 \) and \( x_2 \). When it is supplied two matrix arguments instead, \( X_1 \) of size \( r_1 \)-by-\( c_1 \), and \( X_2 \) of size \( r_2 \)-by-\( c_2 \), then \( Y = f(X_1, X_2) \) becomes a matrix of size \( \max(r_1, r_2) \)-by-\( \max(c_1, c_2) \), which individual entries are

\[
Y[i; j] = f(X_1[\min(i, r_1); \min(j, c_1)], X_2[\min(i, r_2); \min(j, c_2)]),
\]

(3.1)

for \( i = 1, \ldots \max(r_1, r_2) \), and \( j = 1, \ldots \max(c_1, c_2) \). While this rule is can easily be generalized for a function with any number of arguments, \texttt{rlab} can handle at most two-dimensional arrays, that is, matrices.

The matrix-optimized binary operations, e.g., addition, subtraction, element multiply and division, have the following feature: If one of the argument is an empty matrix then the other matrix is returned unchanged.

\texttt{rlab} allows integer matrices. For binary operations addition, subtraction, and element multiply, if both are integer matrices then the result is an integer matrix, as well.
Part II

Solvers
Chapter 4

Controlling rlab from command line

4.1 Command Line Input

rlab2 and rlab3 support the following command line flags

-\texttt{d} Print disassembled program to terminal during execution.
-\texttt{l} Do not load default r-libraries prior to startup.
-\texttt{m} Do not print greeting message when entering \texttt{rlab}.
-\texttt{q} Do not load initialization file during startup.
-\texttt{i} Indicates \texttt{rlab} to stay in interactive mode after the scripts have finished executing.

-\texttt{e "cmds"} Tells \texttt{rlab} to execute the statements in string \texttt{cmds} before loading the script or reverting to the command prompt, but after the libraries from \texttt{rlib} have been loaded.

Typically, to execute a file, say \texttt{eg1\_rotateellipse\_r} from command line one would type

\begin{verbatim}
# rlab2 eg1\_rotateellipse\_r [press enter here]
#
\end{verbatim}

Using flag \texttt{-i} would force \texttt{rlab} to stay interactive, that is

\begin{verbatim}
# rlab2 -i eg1\_rotateellipse\_r [press enter here]
>>
\end{verbatim}

Use \texttt{"-\textendash"} at the end of the command line flags for \texttt{rlab}, follow it with list of r-scripts that ends with \texttt{"-\textendash"}, then put all the arguments that are to be passed to \texttt{rlab} environment:

\begin{verbatim}
# rlab2 -i -e \textquoteleft\textquoteleft{x=1; y=2}\textquoteright\ -- eg1.r eg2.r -e this is -e -f -h [press enter here]
>> isinteractive()
ans =
  1
>> argv()
\end{verbatim}
ans =
  -e this is -e -f -h
>> basename()
ans =
/usr/local/bin/rlab64-2.4.2-gcc4.8.1
>>

4.2 Accessing Command Line Options Inside rlab

The following functions are relevant for processing the command line parameters within rlab.

4.2.1 argv

Format: $y = argv()$

Result: $y$, string vector of input arguments following "-" with which the list of r-scripts to be executed ended on command line that was used to start rlab.

4.2.2 basename

Format: $y = basename()$

Result: $y$, string scalar, exact name of rlab executable on the system.

4.2.3 isinteractive

Format: $y = isinteractive()$

Result: $y$, integer scalar 0 or 1, provides type of environment in which the r-script is executed (within rlab environment this is always 1). If this is 0 the execution of the script has been invoked from command line, and unless the flag "-i" is set there, upon script completion the control will be returned to the process invoking it.
Chapter 5

Miscellaneous

5.1 Input/output/control functions

5.1.1 stat

Format: \( r = \text{stat}(fn) \),

Arguments:

1. \( fn \), string vector, filenames.

Result: \( r \), list.

Abstract: Wrapper for the system \texttt{stat()}. Retrieves all information about the file, which is returned as entries in the resulting list.

```plaintext
>> \text{stat("some_file_name")}
ans =
   atime   ctime    dev    file    gid
   inode   mode    mtime   nlinks  size
   uid
```

Note: \texttt{isfile()}, \texttt{isdir()}, \texttt{isblkspec()}, \texttt{ischarspec()} are all based on \texttt{stat()}. Entries \texttt{atime} - last access time, \texttt{ctime} - creation time, and \texttt{mtime} - mode modification time - are all in unix-epoch seconds: use \texttt{clock()} to get calendar time.

5.1.2 inputs

Format: \( s = \text{inputs}(\text{/timeout/}, \text{/echo_mode/}, \text{/limit/}) \),

Arguments:

1. \texttt{timeout}, positive real scalar, duration of time the function waits for user input. The default value is 5 (seconds).
2. \textit{echo\_mode}, integer scalar, accepts values 0 through 5, where 0 does not echo pressed key in the terminal, 1 does, 2 prints \texttt{“*”} instead, 3 prints between 1 and 4 \texttt{“*”} for each pressed key, 4 prints a random character, while 5 prints between 1 and 4 random characters in the terminal for each pressed key. The default value is 1.

3. \textit{limit}, integer scalar or array. If scalar, then it represents a lower bound on the ASCII codes of acceptable characters (say 31, if only printable characters are accepted). If array, then it represents the ASCII codes of acceptable characters. The default value is 1 (echos characters as they are pressed). The function stops collecting when \texttt{“\n”} is pressed, and does not return newline character with the collected input.

\textbf{Abstract:} The function waits for user input for the specified duration of time, during which it expects certain characters, and possibly echoes the input in the terminal.

\section*{5.1.3 prompt}
\textbf{Format:} \( s = \text{prompt}(p, d, c, \text{opts}) \),

\textbf{Arguments:}
1. \textit{p}, string, text that is printed as a prompt for user input.
2. \textit{d}, any scalar, default value which will be used if user does not make a choice (presses enter at the prompt).
3. \textit{c}, any array, provides choices from which user has to choose.
4. \textit{opts}, list with the following entries
   \begin{itemize}
   \item \textit{timeout}, \textit{echo}, \textit{limit}. These entries imply that \textit{inputs} will be used for collecting the user input, otherwise the function will wait indefinitely for the user input.
   \item \textit{input\_class}, what to expect for input: “int”, “real” or “string”,
   \item \textit{question\_mark}, string, printed after the text of the prompt (and possibly default value),
   \item \textit{clear\_screen}, integer scalar, 0 or 1, where 1 clears the screen once, while 0 clears the screen before each prompt. If the entry is not provided then the function does not clear screen at all.
   \item \textit{print\_choices}, string scalar, the text one wants to print informing user about the available choices. By default, the values from array \textit{c} are printed separateb by commas.
   \end{itemize}

\textbf{Abstract:} The function prompts user to provide certain values from certain choices, and if user just presses enter, it takes a default value.

\section*{5.1.4 randchar}
\textbf{Format:} \( sr = \text{randchar}(nr, nc, len, ctype) \),

\textbf{Arguments:} integer \textit{nr}, integer \textit{nc}, integer \textit{len}, string \textit{ctype},

\textbf{Result:} \textit{MDS} \( * sd \).
5.1. INPUT/OUTPUT/CONTROL FUNCTIONS

Abstract: This command creates a string matrix with \( nr \) rows and \( nc \) columns, where each entry is a string of length \( len \). The format \( ctype \) indicates what kind of characters will be included in creation of the strings. The possibilities for \( ctype \) are

- “alphanumeric”: characters chosen from 012..9, abcde..z, ABCDE..Z,
- “printable”: characters in ascii range 33:126 and 161:255,
- “alphabet”: characters abcde..z, ABCDE..Z,
- “numeric”: characters 0123..9.

This command is very useful if a random file names are needed, e.g., for temporary files.

5.1.5 spinner
Format: \texttt{voidspinner(void)}

Abstract: This function prints a member of the sequence “—”, “/”, “\" on the same spot on the screen following each call, and thus giving an impression that the computer is doing something. I use it to insert inside of a \texttt{for} loop to see how fast the code does each iteration.

5.1.6 smiley
Format: \texttt{voidsmiley(void)}

Abstract: This function prints a different member of a sequence of ‘smiley’ characters on the same spot on the screen following each call. It thus gives an impression that the computer is doing something. I use it to insert inside of a \texttt{for} loop to see how fast the code does each iteration.

5.1.7 getmatrix
Format: \texttt{m = getmatrix(filename/, ncr, ich/)},

Arguments: \texttt{char * filename}; Optional: \texttt{integer ncr, char * ich},

Result: \texttt{MDR * m} or \texttt{MDS * m},

Abstract: The function reads in the matrix from the data file. However, it does not forget to close the file it did read from (like I used to do). To justify its existence an optional parameter \( ncr \) is added. In this case the function reads the data from the file \texttt{filename}, where \( ncr \) rows in the data file are concatenated into a single row of a string matrix \( m \). When doing concatenation a line from the input file is skipped if \( ich \) is the string at the beginning of the line. Default value for \( ich \) is “#”. See also command \texttt{reads}.

5.1.8 hostname
Format: \texttt{sm = hostname()},

Arguments: see \texttt{man hostname} from \texttt{bash} prompt,

Result: \texttt{MDS * sm},

Abstract: Using this function one can find out the hostname of the machine on which work is performed. It is a pipe to the \texttt{hostname} command in shell. By default the flag -\( f \) is assumed (fully qualified domain name).
5.1.9 email
Format: \texttt{email(msgrecepient, msgsubject, msgbody)}
Arguments: \texttt{string msgrecepient}, \texttt{string msgsubject}, \texttt{MDR * or MDS * msgbody},
Abstract: This function sends an email to the \texttt{msgrecepient} with the subject line \texttt{msgsubject}, where the body of the message is contained in the matrix structure (real, string or complex) in \texttt{msgbody}. The command relies on default mailer agent on Linux, \texttt{/usr/bin/mail} program. I see it being used to report a result of some long unattended calculation in the background.

5.1.10 ls
Format: \texttt{sm = ls(/dir/)},
Arguments: \texttt{MDS * dir},
Result: \texttt{MDS * sm},
Abstract: This function is the port to the shell \texttt{ls} function, where \texttt{dir} is passed as the argument. Useful in when automatizing the manipulations with the files within the rlab environment.

5.1.11 mkdir
Format: \texttt{status = mkdir(dirlist)},
Arguments: \texttt{MDS * dirlist},
Result: \texttt{integer status}, or \texttt{string status}
Abstract: The function creates directories listed in the string array \texttt{dirlist}. If there were no error messages it returns \texttt{status} = 1. If there were any error messages than they are returned.

5.1.12 mv
Format: \texttt{mv(file, location, flags)},
Arguments: \texttt{string file}, \texttt{string location}, \texttt{string flags},
Result: \texttt{integer status}
Abstract: The function moves \texttt{file} to \texttt{location} where \texttt{flags} are passed to the shell command \texttt{move}. It returns status 1, if file move was successful, and zero if not.

5.1.13 pwd
Format: \texttt{sm = pwd()},
Result: \texttt{MDS * sm},
Abstract: This function is the port to the shell function \texttt{pwd}, it prints the working directory of the rlab.

5.1.14 touch
Format: \texttt{touch(filenames, params)}
Arguments: \texttt{MDS * filenames}, \texttt{char * params},
Abstract: The function changes the time attribute of the files the names of which are given in a column-vector filenames using the information in parameter. One is here recommended to check “man touch” on its machine. I use this function when doing the calculations which produce large amounts of data which does not need to be backed-up on an incremental device. Consider an example:

```bash
>> touch("timer.r", "-d '7 days ago'")
```

This will change the time attribute of the file “timer.r” from what ever it was previously to the date which is 7 days ago from today.

### 5.1.15 grep

**Format:** 

\[ sm = \text{grep}(textvar, pattern) \]

**Arguments:** 

MDS *textvar, char *pattern,

**Result:** 

MDS *sm,

**Abstract:** This function checks a column-vector of string entries for a pattern, and returns a string array of only those elements from the array which contain the pattern. Consider an example

```bash
>> x=randchar(2,1,16)
uRfT8l1Lmx0AxY3
AIWspCNxSHFYbhTR
>> grep(x, "yes")
[]
```

### 5.1.16 rlab_initstderr

**Format:** 

\[ sm = \text{rlab_initstderr}(titlestring) \]

**Arguments:** 

char *titlestring,

**Result:** 

MDS *sm,

**Abstract:** This function initializes a xterm window and adds to the rlab environment list-type variable rlab_config an additional entry rlab_config.stderr, which contains the pointer (that is, the name) of the newly created virtual terminal, something like /dev/pts/nn, where nn is some number. I find this function useful when using the rlab functions that provide run-time messages or warnings, to avoid the dump of the same onto main rlab window. Instead, the pointer to the xterm window can be sent as the parameter of the command, so that xterm gets all of them while keeping the main window neat. The parameter titlestring is printed on the top of the newly opened xterm window.

### 5.1.17 view

**Format:** 

\[ \text{view}(data, [editor]) \]

**Arguments:** 

MDS *data; Optional: MDS *editor,

**Result:** none,

**Abstract:** The function either opens a konsole and starts “vi” editor inside it on the content of the data string, or starts emacs, with the content of the variable data in it. Which of the two is given by the variable editor = vi, emacs. Default value is vi.
5.1.18 edit

Format: \( \text{editeddata} = \text{edit}(\text{data}, \text{editor}) \)

Arguments: \( \text{MDR \ast data; Optional: MDS \ast editor,} \)

Result: \( \text{MDR \ast editeddata} \).

Abstract: This function allows one to change the content of variable \( \text{data} \) in a simple fashion. By typing the command:

\[
>> \text{var} = \text{edit(var)}
\]

\text{rlab} starts the editor which opens with the content of the variable \( \text{var} \). Once the editing is finished (eg. !w, !q sequence in vi) the control is returned to \text{rlab}, where the variable \( \text{var} \), in this case, has the edited value. This command may be useful in fixing the glitches in bigger data arrays.

5.1.19 isfile

Format: \( \text{filestatus} = \text{isfile}(\text{filelist}) \),

Arguments: \( \text{filelist, string vector, names of the files} \)

Result: \( \text{filestatus, real vector} \).

Abstract: Tests each entry in \( \text{filelist} \) whether it is a regular file (\( \text{filestatus} = 1 \)) or not (\( \text{filestatus} = 0 \)).

5.1.20 isdir

Format: \( \text{filestatus} = \text{isdir}(\text{filelist}) \),

Arguments: \( \text{filelist, string vector, names of the files} \)

Result: \( \text{filestatus, real vector} \).

Abstract: Tests each entry in \( \text{filelist} \) whether it is a directory (\( \text{filestatus} = 1 \)) or not (\( \text{filestatus} = 0 \)).

5.1.21 islink

Format: \( \text{filestatus} = \text{islink}(\text{filelist}) \),

Arguments: \( \text{filelist, string vector, names of the files} \)

Result: \( \text{filestatus, real vector} \).

Abstract: Tests each entry in \( \text{filelist} \) whether it is a symbolic link (\( \text{filestatus} = 1 \)) or not (\( \text{filestatus} = 0 \)).

5.1.22 isblkspec

Format: \( \text{filestatus} = \text{isblk/spec}(\text{filelist}) \),

Arguments: \( \text{filelist, string vector, names of the files} \)

Result: \( \text{filestatus, real vector} \).

Abstract: Tests each entry in \( \text{filelist} \) whether it is a block special file (\( \text{filestatus} = 1 \)) or not (\( \text{filestatus} = 0 \)).
5.1.23  ischarspec

**Format:**  \( filestatus = ischarspec(filelist), \)

**Arguments:**  \( MDS * filelist, \)

**Result:**  \( MDR * filestatus. \)

**Abstract:**  Tests each entry in \( filelist \) whether it is a character special file (\( filestatus = 1 \)) or not (\( filestatus = 0 \)).
5.2 General purpose functions

location: $RD/rlib/libgeneral.r

5.2.1 names

Format: $s = names(p)$

Arguments:

1. $p$, string scalar, part of the name of the solver.

Result: $s$, string vector, entries in global symbol table which contain in their name the pattern $p$.

Abstract: Useful for locating the names of the solvers which partial name is known.

5.2.2 range

Format: $i = range(m)$

1. $m$, any scalar, vector, or matrix.

Result: $i$, real vector or 2-column matrix.

Abstract: If $m$ is vector then $i$ contains range of indices through which all entries in $m$ can be accessed:

\[
\begin{array}{c}
\text{>> x= rand(1,3)} \\
0.680477173 & 0.309193013 & 0.866111103 \\
\text{>> range(x)} \\
1 & 2 & 3
\end{array}
\]

If $m$ is matrix then $i$ is a two-column vector, where by going over the rows of $i$ all entries in matrix $m$ can be accessed:

\[
\begin{array}{c}
\text{>> x= rand(2,3)} \\
0.283073163 & 0.855798839 & 0.436473753 \\
0.391326915 & 0.136659487 & 0.380062419 \\
\text{>> range(x)} \\
1 & 1 \\
1 & 2 \\
1 & 3 \\
2 & 1 \\
2 & 2 \\
2 & 3
\end{array}
\]

5.2.3 last, lastr, lastc

Format: $x = last(m)$

1. $m$, any scalar, vector, or matrix.

Result: $x$, last entry (row-, or column-wise) of matrix $m$. 
5.3 File locking functions

location: $RD/rlib/liblock.r

5.3.1 getpid

Format: \( i = \text{getpid}() \)

Result: \( i \), double scalar, process id of the current \texttt{rlab} instance.

5.3.2 lock

Format: \( \text{lock}(fn) \)

Arguments:

1. \( fn \), string vector, filenames that user wants to “lock,” that is, inform other instances of \texttt{rlab} on this or other machines that the current instance of \texttt{rlab} would like to limit access to these files to the others.

Abstract: File locking is a two-step process in \texttt{rlab}. The function waits for an exclusive write-lock on a regular file \texttt{.lock.fn} using the system call to \texttt{fnctl}. After gaining the write-lock, the function returns to the caller.

5.3.3 unlock

Format: \( s = \text{unlock}(fn) \)

Arguments:

1. \( fn \), string vector, filenames that user wants to “unlock,” that is, inform other instances of \texttt{rlab} on this or other machines that the current instance of \texttt{rlab} would like to release its exclusive access to these files.

Result: \( s \), integer vector, 1 if unlock was successful, 0 if it was not.

Abstract: Checks if the current instance of \texttt{rlab} has write-lock to file \texttt{.lock.fn}. If it does, then it releases it and returns 1, otherwise it returns 0.
Example: The following code snippet shows how a file can be locked by an instance of rlab while the variable data is being written to it.

```
lock(dbname); // wait until have exclusive write-lock
open (dbname, 'a');
write (dbname, data); // append 'data' to 'dbname'
close (dbname);
unlock(dbname); // release exclusive write-lock for the others
```

Note: As a result of file locking operations the lock file .lock.fn is created in the directory in which the file fn is located. The lock file persists even after all the processes accessing it are completed.

5.4 Timer Related Functions

5.4.1 sleep

Format: sleep(s),

Arguments:
1. s, real scalar, or real row-vector of length 3, 6, or 8;

Abstract: Stops the execution of rlab code for the duration of time depending on s:
If s is a real scalar, than this is the amount of time rlab waits before resuming the execution of the code.
If s is 3-column vector, then this is understood as the absolute time in H:M:S format at which the code execution will resume, on the same day!
If s is a 6- or 8-column vector, then this is the local calendar time at which the code execution will resume.

5.4.2 tic

Format: tic(/i/),

Arguments:
1. i, integer vector with entries in range 1...32, the indices of the timers, if omitted default value of 1 is assumed;

Abstract: Resets the system timers which indices are given in i to zero.

5.4.3 toc

Format: x = toc(/i/),

Arguments:
1. i, integer vector with entries in range 1...32, the indices of the timers, if omitted default value of 1 is assumed;
Abstract: Reads the time passed on the system timers which indices are given in \( i \).

Note: The functions \texttt{tic} and \texttt{toc} with arguments are useful when timing the sub-processes within another timed process: E.g., consider accessing instruments through serial port (each given certain timeout) or the internet, at regular intervals.

5.5 Functions for Manipulating Dates

The functions in this group are partially written in C, and partially scripted. The scripted functions are based on \texttt{m*lab} collection \texttt{timeutil} by Peter John Acklam, which he posted on his web site, \texttt{http://home.online.no/ pjacklam/matlab/software/util/timeutil}.

I fixed some of the functions so they don’t fail for inconsistent arguments (e.g., the functions from \texttt{rlab} library will tell you that 15-15-nnnn is a non-existent date). \texttt{rlab} functions are located in \texttt{rlib/libtime.r}.

The functions support the following date formats:

- 1-column format (unix-epoch);
- 6-column format (year, month, day, hour, minute, second);
- 8-column format (year, month, day, hour, minute, second, dst, offset), as specified in C-language \texttt{<timer.h>}.

Most of the functions accept the argument in either of the three formats, and produce result in one of them.

5.5.1 seconds

Format: \( t = \text{seconds}(/tx/) \)

Arguments:

1. \( tx \), 1-, 6- or 8-column integer or double matrix of dates. If omitted the current time is used.

Result: \( t \), double.

Abstract: Converts time to unix-epoch, or gives unix-epoch of local time.

5.5.2 clock

Format: \( tm = \text{clock}(/tx/) \)

Arguments:

1. \( tx \), 1-, 6- or 8-column real matrix of dates.

Result: \( tm \), 8-column integer matrix.

Abstract: Converts time to 8-column format, or gives 8-column formatted local time.

5.5.3 etime

Format: \( dt = \text{etime}(tm1,tm2) \)

Arguments:

1. \( tm1, tm2 \), 1-, 6-, or 8-column real matrix of times.
Result: $dt$, double vector.

Abstract: Finds the number of elapsed seconds between times $tm1$ and $tm2$.

$$\text{etime}(tm1, tm2) = \text{seconds}(tm1) - \text{seconds}(tm2).$$ (5.1)

5.5.4 gmt ime

Format: $tm = \text{gmtime}(/t/)$

Arguments:

1. $t$, integer or real matrix containing times in 1-, 6- or 8-column format. If not provided, the current time is used.

Result: $tm$, 8-column integer matrix.

Abstract: Converts time to Universal Greenwich Meridian Time in 8-column format.

5.5.5 time2dstr

Format: $s = \text{time2dstr}(tm, /fmt/)$

Arguments:

1. $tm$, integer or real matrix containing times in 1-, 6- or 8-column format.

2. $fmt$, string, format in which the local time is to be displayed, see Table 5.1 ($"%c (%z/%Z)"$, default value).

Result: $s$, string matrix.

Abstract: Converts time to string $s$ according to provided format $fmt$, see Table 5.1 for acceptable values.

5.5.6 dstr2time

Format: $t = \text{dstr2time}(dstr, /fmt)$

Arguments:

1. $dstr$, string matrix, each entry contains full date;

2. $fmt$, string scalar, anticipated format of dates in $dstr$.

Result: $t$, integer matrix of times in 8-column format.

Abstract: If format $fmt$ is not provided then the function uses GNU coreutil-8.22 parse-datetime module to extract 8-column time from date string $dstr$.

If format $fmt$ is given, then it uses the c-language function $\text{strptime}$ to extract the 8-column date.

Please note: The function $\text{strptime}$ is volatile in that if the $dstr$ does not contain full time information the returned time is non-sensical.
5.5. FUNCTIONS FOR MANIPULATING DATES

```matlab
>> _today_stamp = time2dstr(seconds(), "%Y%m%d-%H%M%S")
ans =
20140226-102156

>> dstr2time(_today_stamp, "%Y%m%d-%H%M%S")
ans =
2014 2 26 10 21 56 0 -5

>> clock()
ans =
2014 3 20 9 3 23 1 -4

>> dstr2time("13 days ago")
ans =
2014 3 7 8 3 14 0 -5
```

5.5.7 timezone

**Format:** \( s = \text{timezone}(\text{tz}) \)

**Arguments:**

1. \( \text{tz} \), string, timezone.

**Result:** \( s \), string, timezone.

**Abstract:** Sets the timezone of the system through the environmental variable TZ. If the argument is omitted, then the function retrieves its current value.

5.5.8 datesinrange

**Format:** \( c = \text{datesinrange}(t, \text{range}) \)

**Arguments:**

1. \( t \), vector, time datum in one of the three supported formats.
2. \( \text{range} \), vector, offset days from the time datum \( t \)

**Result:** \( c \), integer matrix of 8-column times in specified range from the time datum \( t \)

**Abstract:** Finds dates offset from the known time point.

5.5.9 dayofweek, dayofmonth, dayofyear

**Format:** \( x = \text{dayofweek}(t) \)

**Arguments:**

1. \( t \), matrix, time data in one of the three supported formats.

**Result:** \( x \), column vector.
Abstract: Converts time data to the corresponding week day, in range 0:7, or day in the month, or day in the year. Use \texttt{ceil()} to get the ISO day of the week, or month, or year. The fractional part corresponds to the ratio of seconds passed in that day to the total number of seconds in the day (e.g., noon corresponds to a fractional value of 0.5 - half of a day).
If argument is omitted current time is used.

5.5.10 \texttt{isleapyear}

Arguments: Format: $i = \text{isleapyear}(t)$
1. $t$, vector, time datum in one of the three supported formats.

Result: $i$, vector or scalar, 1 or 0 depending whether the datum $t$ is leap or not.

5.5.11 \texttt{monthofyear}, \texttt{weekofyear}

Arguments: Format: $i = \text{isleapyear}(t)$
1. $t$, vector, time datum in one of the three supported formats.

Result: $i$, vector or scalar, returns 1 through 12 for month in datum $t$, or 1-53 for the week.

5.5.12 \texttt{daysinmonth}, \texttt{daysinyear}, \texttt{weeksinyear}

Arguments: Format: $i = \text{daysinmonth}(t)$
1. $t$, vector, time datum in one of the three supported formats.

Result: $i$, vector or scalar, returns number of days in the month or in the year given in the datum $t$, or a number of weeks.

5.5.13 \texttt{date2jd}, \texttt{date2mjd}, \texttt{jd2mjd}

Arguments: Format: $jd = \text{date2jd}(t), mjd = \text{date2mjd}(t), mjd = \text{jd2mjd}(jd)$
1. $t$, vector, time datum in one of the three supported formats.

Result: $i$, vector or scalar, returns the (modified) Julian day number of the given date (Gregorian calendar) plus a fractional part depending on the time of day.

Note: Start of the JD (Julian day) count is from 0 at 12 noon 1 January -4712 (4713 BCE), Julian proleptic calendar. Note that this day count conforms with the astronomical convention starting the day at noon, in contrast with the civil practice where the day starts with midnight.

Astronomers have used the Julian period to assign a unique number to every day since 1 January 4713 BCE. This is the so-called Julian Day (JD). JD 0 designates the 24 hours from noon UTC on 1 January 4713 BCE (Julian proleptic calendar) to noon UTC on 2 January 4713 BCE.

Note: BCE stands for Before Common Era, the opposite of Common Era, negative and positive dates with respect to time zero.
5.5.14 yearnum

Arguments: Format: \( y = yearnum(t) \)
1. \( t \), vector, time datum in one of the three supported formats.

Abstract: Returns the year of the datum \( t \), where the fractional part refers to the day within the year.

5.5.15 otherday

Format: \( y = otherday(t, d, k) \)

Arguments:
1. \( t \), vector, time datum in one of the three supported formats. If \( 't' \) is not provided use the current time.
2. \( d \), integer 1-7, for the day in the week.
3. \( k \), integer vector, for the range of days.

Result: \( y \), dates.

Abstract: If \( k = 0 \) and \( 't' \) is not given find the day \( 'd' \) in the current week, otherwise find the day \( 'd' \) in the week described by datum \( t \). If \( k < 0 \) then find the day \( 'd' \) in the \( k \)-th week before this one and opposite for \( k > 0 \), or with respect to the datum \( t \).

5.5.16 earliertime, latertime

Format: \( y = earliertime(d, t) \)

Arguments:
1. \( d \), integer 1-7, for the day in the week.
2. \( t \), vector, time datum in one of the three supported formats. If \( 't' \) is not provided use the current time.

Abstract: Finds the day \( d \) before or after datum \( t \).

5.5.17 days2hms, hms2days

Format: \( y = days2hms(d), d = hms2days(y) \)

Arguments:
1. \( d \), number of days.
\( y \), 3-column matrix \([h, m, s]\).

Result: \( y = [h, m, s] \), 3-column matrix.

Abstract: Converts days into a 3-column array, \([h, m, s]\), and vice versa, that is 3-column array, \([h, m, s]\), to number of days.
5.5.18 \quad \textbf{hms2sec, sec2hms}

\textbf{Format:} \( s = \text{hms2sec}(y), y = \text{sec2hms}(s) \)

\textbf{Arguments:}

1. 
   \( y \), 3-column matrix \([h, m, s]\).
   \( s \), vector, number of seconds.

\textbf{Result:} \( s \), vector.

\textbf{Abstract:} Converts time in format \([h, m, s]\) to number of seconds, and \textit{vice versa}.

5.5.19 \quad \textbf{jd2date, jd2jdate, jdate2jd, jd2mjd, mjd2jd}

\textbf{Format:} \( d = \text{jd2date}(jd), d = \text{jd2jdate}(jd), jd = \text{jdate2jd}(d), mjd = \text{jd2mjd}(d), jd = \text{mjd2jd}(mjd), d = \text{mjd2date}(mjd) \)

\textbf{Arguments:}

1. 
   \( jd \), vector, julian day date.

\textbf{Result:} \( d \), 6-column date matrix, \([Y, m, d, h, m, s]\).

\textbf{Abstract:} The function returns the Julian calendar date (year, month, day, hour, minute, and second) corresponding to the Julian day number \( jd \) with and without modification, and \textit{vice versa}.

Start of the JD (Julian day) count is from 0 at 12 noon 1 JAN \(-4712\) (4713 BCE), Julian proleptic calendar. Note that this day count conforms with the astronomical convention starting the day at noon, in contrast with the civil practice where the day starts with midnight.

Astronomers have used the Julian period to assign a unique number to every day since 1 January 4713 BCE. This is the so-called Julian Day (JD). JD 0 designates the 24 hours from noon UTC on 1 January 4713 BCE (Julian calendar) to noon UTC on 2 January 4713 BCE.

5.5.20 \quad \textbf{isdate, isjdate}

\textbf{Format:} \( i = \text{isdate}(ymd) \)

\textbf{Arguments:}

1. 
   \( ymd \), 3-column matrix, \( ymd = [Y, m, d] \), Gregorian or Julian date.

\textbf{Result:} \( i \), integer vector.

\textbf{Abstract:} Returns 1 if the entry is a proper date, 0 otherwise.
### 5.5. FUNCTIONS FOR MANIPULATING DATES

<table>
<thead>
<tr>
<th>fmt</th>
<th>meaning</th>
<th>example output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;%a&quot;</td>
<td>abbreviated weekday name</td>
<td>Mon, Tue, ...</td>
</tr>
<tr>
<td>&quot;%A&quot;</td>
<td>full weekday name</td>
<td>Monday, Tuesday, ...</td>
</tr>
<tr>
<td>&quot;%b&quot;</td>
<td>abbreviated month name</td>
<td>Jan, Feb, ...</td>
</tr>
<tr>
<td>&quot;%B&quot;</td>
<td>full month name</td>
<td>January, ...</td>
</tr>
<tr>
<td>&quot;%c&quot;</td>
<td>date and time</td>
<td>Sun Jan 6 15:41:05 2002</td>
</tr>
<tr>
<td>&quot;%d&quot;</td>
<td>day of month as a number</td>
<td>01 – 31</td>
</tr>
<tr>
<td>&quot;%H&quot;</td>
<td>hour (24-hour clock)</td>
<td>00 – 23</td>
</tr>
<tr>
<td>&quot;%I&quot;</td>
<td>hour (12-hour clock)</td>
<td>01 – 12</td>
</tr>
<tr>
<td>&quot;%j&quot;</td>
<td>day of year as a number</td>
<td>000 – 366</td>
</tr>
<tr>
<td>&quot;%m&quot;</td>
<td>month as a number</td>
<td>01 – 12</td>
</tr>
<tr>
<td>&quot;%M&quot;</td>
<td>minute as a number</td>
<td>00 – 59</td>
</tr>
<tr>
<td>&quot;%p&quot;</td>
<td>AM/PM designation</td>
<td>AM,PM</td>
</tr>
<tr>
<td>&quot;%S&quot;</td>
<td>seconds</td>
<td>00 – 61</td>
</tr>
<tr>
<td>&quot;%U&quot;</td>
<td>week number of the year, Sunday first day</td>
<td>00 – 53</td>
</tr>
<tr>
<td>&quot;%w&quot;</td>
<td>weekday as a number</td>
<td>0(Sunday) - 6</td>
</tr>
<tr>
<td>&quot;%W&quot;</td>
<td>week number of the year, Monday first day</td>
<td>00 – 53</td>
</tr>
<tr>
<td>&quot;%x&quot;</td>
<td>date</td>
<td>01/06/02</td>
</tr>
<tr>
<td>&quot;%X&quot;</td>
<td>time</td>
<td>15:48:01</td>
</tr>
<tr>
<td>&quot;%y&quot;</td>
<td>year without century</td>
<td>00 – 99</td>
</tr>
<tr>
<td>&quot;%Y&quot;</td>
<td>year with century</td>
<td>1900 –</td>
</tr>
</tbody>
</table>

Table 5.1: Acceptable formats for the function `time2dstr`.
5.6 Indices, row and column manipulations

5.6.1 unique
Format: \( s = \text{unique}(x) \)
Arguments:
1 \( x \), vector;

Result: \( s \), vector of the same type as \( x \) in which each entry in \( x \) appears exactly once.
Abstract: The function returns sorted unique elements in vector \( x \).

5.6.2 union
Format: \( s = \text{union}(x, y) \)
Arguments:
1 \( x \), vector;
2 \( y \), vector;

Result: \( s \), union of \( x \) and \( y \).
Abstract: The function returns sorted unique elements in union of vectors \( x \) and \( y \).

5.6.3 intersect
Format: \( s = \text{intersect}(x, y) \)
Arguments:
1 \( x \), vector;
2 \( y \), vector;

Result: \( s \), intersection of \( x \) and \( y \).
Abstract: The function returns sorted elements common for both vectors \( x \) and \( y \).

5.6.4 complement
Format: \( s = \text{complement}(x, y) \)
Arguments:
1 \( x \), vector;
2 \( y \), vector;

Result: \( s \), \( x \) and complement \( y \), that is, elements in \( x \) that are not present in \( y \).
Abstract: The function returns sorted elements that exist only in \( x \).
5.6.5  merge

Format: \( s = merge(x, y) \)

Arguments:

1. \( x \), real data matrix, with independent variable in the first column.
2. \( y \), real data matrix, with independent variable in the first column.

Result: \( s \), real matrix.

Abstract: Appends data from both matrices in new matrix, which first column is the union of the first columns of \( x \) and \( y \). Absent values are filled with NaN's.

Example: Consider merge of two data matrices \( d1 \) and \( d2 \).

```matlab
>> d1
ans =
8   152.31761
7   112.552089
1  -1.80540831
2    2.59983992
0    0.215016465
6   78.3555393
5    50.2339188
4    28.2814005
3    12.148733

>> d2
ans =
2    2.31714743
-4   68.5618686
6   78.5847951
4    28.5201186
-2   22.0350066
0    0.504355725

>> merge(d1,d2)
ans =
-4   NaN   68.5618686
-2   NaN   22.0350066
0    0.215016465  0.504355725
1  -1.80540831   NaN
2    2.59983992  2.31714743
3    12.148733   NaN
4    28.2814005  28.5201186
5    50.2339188   NaN
6   78.3555393  78.5847951
7   112.552089   NaN
8   152.31761   NaN
```

>>
5.6.6 **compact**

Format: \( s = \text{compact}(x, i, sx, wx) \),

Args:

1. \( x \), real data matrix;
2. \( i \), integer, index of a column in \( x \);
3. \( sx \), two-column matrix of indices in the form \( sx = [j_x, j_{\sigma_x}] \).
4. \( wx \), two-column matrix of indices in the form \( wx = [j_x, j_{w_x}] \).

**Result:** \( s \), real dense matrix.

**Abstract:** Compacts data matrix \( x \) with respect to the \( i \)-th column (if \( i \) is not given then \( i = 1 \) is assumed) in such a way that two or more rows in \( x \) which have the same \( i \)-th entry are replaced by their means unless the matrix \( sx \) xor \( wx \) is provided.

If user provides one of the two-column matrices \( sx \) or \( wx \), they contain per-row the index of the datum \( j_x \) and the index of its standard deviation/tolerance \( j_{\sigma_x} \) or its weight \( j_{w_x} \). Indices in the matrix \( sx \) or \( wx \) with \( i \) should cover the range of column indices of \( x \).

**Example:** In the following example data matrix \( x \) is compacted with respect to the first column. For this example \( x \) has been sorted with respect to the first column.


gg x
ans =
   -3    1.31583351    0.05
   -2    1.13021612    0.05
   -2    -0.348800041   0.05
    0    -0.318240244   0.05
    0    1.21816713    0.05
    0    0.196647169    0.05
    5    -0.168349856   0.05
    7    -0.489007578   0.05
   12    -0.35753957    0.05

>> compact(x,1,[2,3])
ans =
   -3    1.31583351    0.05
   -2    0.390708039  1.04582236
    0    0.264426571  0.83271165
    5    -0.168349856   0.05
    7    -0.489007578   0.05
   12    -0.35753957   0.05

**Note:** If one wants to calculate means and standard deviations from the rows of data without them, then the original matrix needs to be restacked so that each data column is followed by the same small constant standard deviation (as was done in the example above).

**Note:** In calculations using \( sx \) or \( wx \) matrices, incomplete entries (at least one value is NaN) are ignored.
5.6.7 rmrows

Format: \( s = \text{rmrows}(x, idx) \),

Arguments:

1. \( x \), dense matrix;
2. \( idx \), real vector of indices of the rows that are to be removed from \( x \).

Result: \( s \), dense real matrix.

Abstract: \textit{rmrows} removes rows the indices of which are given in the vector \( idx \). The function is user friendly, that is, it checks for the range and repetition in the array \( idx \) before removing the rows.
Chapter 6

Data Plotting and Export
CHAPTER 6. DATA PLOTTING AND EXPORT

6.1 Gnuplot

Gnuplot by [Williams and Kelley (2004)](#) is a popular plotting package available for all Un*x environments. The rlabplus library is located in ${RLAB}/rlib/libgnuplot.r, and loaded upon start of rlab. It allows user to access all of the Gnuplot functionality through,

1. Pipe: (default) rlab opens a Gnuplot session in the background, and through pipe writes all plot formatting commands;
2. File: rlab writes all plot formatting commands to an ascii file, which can then be loaded in a user initiated Gnuplot session by typing load 'filename' and further manipulated;
3. Any combination of the above two approaches. E.g., data could be written to a file, while through pipe a Gnuplot session could be opened, and then told to load the data from just created file (this concept was employed in the original rlab-library gnuplot.r).

Initialization/termination functions

6.1.1 gnustart

I Format: $I = \text{gnustart}(fn)$

1. $fn$, string, containing the filename to which the rlab will print all the commands and data. To produce a plot, one should start Gnuplot in a new terminal and type load 'fn' at the prompt.

Result: $I$, integer, the index of the Gnuplot device.

Abstract: Opens a file to which all the command and data are written following the execution of gnuplot or gnusplot command. A script should close the file using gnuclose(I) before loading it into Gnuplot.

II Format: $I = \text{gnustart}()$, $I = \text{gnustart}(/s0/,/fn/)$

Arguments:

1. $s0$, string, containing Gnuplot compatible expression, e.g., set term .

2. $fn$, string, contains the name of the file or a terminal to which any messages created by Gnuplot will be sent. When using Gnuplot via pipe this translates to

   gnuplot 1>'fn' 2>&1

   If omitted $fn$ is replaced by /dev/null,

   gnuplot 1>/dev/null 2>&1

Result: $I$, integer, the index of the newly opened Gnuplot window.

Abstract: Initializes a single Gnuplot device. The library default initialization string is

set term x11 title 'Gnuplot %i: rlabplus'

If this is not satisfactory, one needs to modify the line in file rlabplus_gnuplot.c,

#define GNUPLLOT_SET_TERM "set term x11"

to one's choosing and then rebuild rlab.
6.1. GNUPLOT

6.1.2 gnuclose

Format: gnuclose(I)

Arguments:
1. I, integer, index of an open Gnuplot window.

Abstract: Closes a single Gnuplot window I.

6.1.3 gnuwin

I Format: I = gnuwin()

Abstract: Obtain the index of a default Gnuplot device.

II Format: gnuwin(I)

Arguments:
1. I, integer, index of an open Gnuplot window.

Abstract: Set the index of a default Gnuplot device.

6.1.4 gnuwins

I Format: <<default; available; device >> = gnuwins()

Abstract: Obtains the information about open Gnuplot devices: default holds the index of the default Gnuplot device; available is an array containing the index of all open Gnuplot devices; while device contains the information about the device: “gnuplot” for Gnuplot session, or a “filename” if the output is redirected to a Gnuplot-compatible file.

II Format: gnuwins(N, fmt, stream)

Arguments:
1. N, integer, desired number of open Gnuplot devices (plotting windows);
2. fmt, string vector, containing the command to be passed to Gnuplot for each initialized device. This value can be changed through gnuresetcmd(i), where i is a new initialization/reset series of commands. The following are the predefined shortcuts: eps, eps_mono, ps, ps_mono, png and png.t.
3. stream, terminal or file to which any error messages produced by Gnuplot are sent. Useful in debugging the rlabplus communication with Gnuplot. E.g., one can use the output of stderr() so all messages are passed to error terminal.

Abstract: Manages the number of open Gnuplot windows. It first checks for the number of open windows, and then either closes or opens windows, so that the total number of available windows is N. If the existing windows are reused then they are reset. Use the result of gnuwins() to access particular window, i.e., gnuwins().available[1] for the first available window, and so on.

Note: Use gnuwins(...) to manage the number of open Gnuplot devices.
6.1.5 gnudefault

**Format:** gnudefault(name, value), gnudefault(name), gnudefault()

**Arguments:**

1. `name`, string, name of a global variable containing information pertaining to the current gnuplot installation:
   - `palette`, string, it may contain word “default” which tells gnuplot to use its built-in colors (what ever these are), or a file name that contains color specifications and their names. Currently, `palette` is set to `/usr/share/X11/rgb.txt`, which allows user with the littlest effort to use these colors.
   - `color`, list, by default its entries are the names of the colors as found in the file found in entry `palette`, `/usr/share/X11/rgb.txt`, which contains some ~ 600 colors. Each color entry is a list containing the following three entries: `space`, representing the color space in gnuplot nomenclature; `xyz`, containing the color coordinates in that space; and, `hex`, with the hexadecimal representation of the color.
   - `debug`, integer, 0 or 1, if set to 1 then the all the functions write to the console what are they doing. Default value is 0.
   - `info`, integer, 0 or 1, if set to 1 then only the functions gnuplot or gnusplot report, and then only if their output is redirected to a file. Default value is 1.
   - `reset_wins`, string, a command sent to the open gnuplot windows after each call to gnuwins. Default value is “reset;” .
   - `reset_term`, string, a command sent to an open gnuplot window after its input was temporary redirected to a file. Default value is “reset; set term x11; set output;” .
   - `output`, string, a setting for a type of terminal the output is temporary redirected to. Default value is “eps” .
   - `fontsize`, number, size of font used in the gnuplot terminal. Default value is 11. Keep in mind that for eps format the image gets reduced by a factor of two, and the same apply for the png format.
   - `font` and `font_symbol`, list, containing the fonts being used for respective choice of outputs. Its entries are currently supported types of terminals, “eps” and “png” . Default content is:
     - `font.eps` = “Times-Roman”
     - `font.png` = “times”
     - `font_symbol.eps` = “Symbol”
     - `font_symbol.png` = “”
   At the start of rlab the library initiates a search for the symbol font that can be used for png terminals. gnuplot can work with pfa and ttf fonts only, so to support symbols in your plots the font “system.ttf” has to be obtained and put in the following location.
   - `env`, list, entries are the names of the environment variables that need to be set for gnuplot to work properly. Currently assigned variables are:
     - `GDFONTPATH`, string, environemnt variable used for the gd-library which is used for png/jpeg/tiff format. Its default content is “/usr/share/fonts/truetype:/usr/share/ghostscript/fonts” . In one of this directories user should put the font “symbol.ttf” or change them to match the available linux configuration.
6.1. GNUPLOT

– **set_term**, list, which entries are the names of the terminals besides “x11” that are currently being supported by the library: *eps, eps mono, png, and png.t*. These are used to prepare the terminals to which the output will be temporarily redirected.

– **histmode**, integer 1 or 2, determines the way the GSL-histograms are plotted: 1, plotted using bars which stretch the width of the bins; 2, plotted using lines connecting the center points of the bins.

2 **value**, string or number, a value to be assigned to the global variable name.

If no **value** is given then **gnudefault(name)** returns the current value of the global variable name. **gnudefault()** returns the list of all global variables.

**Plotting**

I organized the library so that it resembles the rlab-library **pgplot** as much as possible:

1. There are many commands which for **pgplot** have “p” or “pg” prepended (e.g., plimits) which got prefix **gnu** in the rlabplus-library.

2. All commands apply to a default Gnuplot device. In the case when there are more devices then the one to which the commands are sent is set with **gnuwin()**.

3. The two library are distinct in that Gnuplot requires **gnulegend** and **gnufomat** to be executed before **gnuplot** or **gnusplot**, that is, before the datasets to be plotted are provided.

6.1.6 **gnutext**

**Format:** 

\[
\text{gnutext}(s, loc[, size, cs, opts])
\]

1 **s**, string, grafitti to be put on top of a graph;

2 **loc** = \([x, y]\), real vector, position of the grafitti;

3 **size**, scalar, relative size of the grafitti in terms of the default font size (11pt);

4 **cs**, string, coordinate system for **loc**. Allowed values are: “first”, to use \(x_1y_1\); “second” to use \(x_2y_2\); “graph”; “screen”, to use absolute coordinates where 0,0 is the bottom left, and 1,1 is the top right corner; and “character”, to use absolute coordinates in units of character width and height. Default value is “first”.

5 **opts**, string, may contain anything else one would want to add.

**Abstract:** 

Constructs a command

\[
\text{set label} \ 's' \ \text{at} \ 'cs' \ \text{loc[1],loc[2]} \ \text{font} \ \{\text{default font},(size*{default size})\} \ \{\text{opts}\}
\]

and executes it in a default gnuplot window.

**Note:** The size of the label is adjusted for saving a plot to a post script file. As is known, then **gnuplot** halves all the sizes of the plot. In order to have a font of size, say, 11 appearing as such on the plot the user size of the plot is multiplied by a factor of two - in other words the library sets the font size to, in this case, 22. A side effect is that the labels may look awkward when plotting on a screen, as they will appear twice as large then desired.
6.1.7 **gnutitle**  
**Format:** `gnutitle(s)`  
**Arguments:**  
1. `s`, string, the title of the plot  

**Abstract:** Sets the title of the graph on a default Gnuplot device.

6.1.8 **gnuxtics, gnux2tics, gnuytics, gnuy2tics, gnuztics**  
**Format:** `gnuxtics(v1/,v2,fmt/)`  
**Arguments:**  
1. `v1`, real vector or scalar, desired position of x-, y-tics (plot) or z-tics (splot). If vector than it represents the position of the tics, if scalar that it represents the interval between two tics;  
2. `v2`, scalar, number of minor tics to be placed between two tics;  
3. `fmt`, string, format for the tics with a default value of “%%g”  

**Abstract:** Regulate primary and secondary, major and minor x- and y- tics, and the primary major and minor z-tics.  

**Note:** Always set secondary axes after the primaries, because commands for primary axes reset the values for the secondary axes.

6.1.9 **gnulimits, gnulimits2**  
**Format:** `gnulimits(xlo,xhi,ylo,yhi,zlo,zhi), gnulimits2(xlo,xhi,ylo,yhi)`  
**Arguments:**  
1-2 `xlo,xhi`, real scalar, min and max values of the primary or secondary x-axis.  
3,4 `ylo,yhi`, real scalar, min and max values of the primary or secondary y-axis.  
5,6 `zlo,zhi`, real scalar, min and max values of the primary z-axis.  

**Abstract:** Set the limits of the plot.

6.1.10 **gnuscale, gnuscale2**  
**Format:** `gnuscale(sx, sy, sz), gnuscale2(sx, sy)`  
**Arguments:**  
1-3 `sx, sy, sz`, string, “lin” for linear axis, “log” for logarithmic.  

**Abstract:** Set the linear or logarithmic style for a primary or secondary axis.
6.1.11 gnuxlabel, gnuylabel, gnuzlabel

Format: gnuxlabel($L_1$, $L_2$), gnuylabel($L_1$, $L_2$), gnuzlabel($L_1$)

Arguments:

1. $L_1$, string, label for the primary x-, y-, or z-axis.
2. $L_2$, string, label for the secondary x- or y-axis.

Abstract: Set the axes labels.

6.1.12 gnulegend

Format: gnulimits($L$)

Arguments:

1. $L$, string vector, labels for each data set to follow in gnuplot().

Abstract: Set the names of the datasets for the plot.

6.1.13 gnuformat

Format: gnuformat($K$)

Arguments:

1. $K$, a string vector, a directions to Gnuplot what to do with each data set that follows in gnuplot(.
   or gnusplot(.

6.1.14 gnurect

Format: gnurect($loc_1$, $loc_2$, $opts$/)

1. $loc_1 = [x_1, y_1]$, two-column real matrix, lower-left corners of the rectangulars to be put on the active
   Gnuplot window;
2. $loc_2 = [x_2, y_2]$, two-column real matrix, upper-right corners;
3. $opts$, string vector, may contain anything else one would want to individual rectangulars being plotted.

Abstract: Executes a series of commands

\begin{verbatim}
set object #n rect from x1,y1 to x2,y2 /opts/
\end{verbatim}

in the default Gnuplot window. It uses internal counter \#n, which is updated with each call to gnurect,
and which is reset to zero with the command gnuwins.
6.1.15  gnuplot

Format:  gnuplot(data/, filename/, fmt//)

Arguments:

1  data:

   (a) real matrix, data to be plotted is in format \([x, y_1, y_2, \ldots]\);
   (b) string vector, then each string is either a filename of the data to be plotted (according to directions
       from the respective gnuformat) or a string to be passed directly to Gnuplot (where we assume
       that Gnuplot will know what to do with it, e.g., plot a functions);
   (c) list, then each entry in the list is assumed to be a data set to be plotted: if an entry is a matrix
       then gnuformat knows what to do with it (e.g., three column matrix containing the \([x, y, dy]\) has
       respective gnuformat of “with errorlines”, and so on); if an entry is a string then it is assumed a
       filename or it is passed to Gnuplot as a string.

2  filename, string, this assumes that Gnuplot is going to draw the plot into a file, instead of to an open
   x11 session. This is useful once visual appearance of the plot is satisfactory - it is time then to export
   the plot to a file for incorporation into a publication.

3  fmt, string, a Gnuplot compatible specification of the format the plot will be written to the filename
   with - basically, everything that follows set term command in Gnuplot. Default format is that given in
   GNU_DEFAULTS.epscolor, see below. These are also available if one chooses two special values, “color”
   or “mono” for fmt. If you have your special choice of values, these may be set in file libgnuplot.so.r,

1  GNU_DEFAULTS = <<>>;
2  GNU_DEFAULTS.debug    = 0;
3  GNU_DEFAULTS.resetcmd = "unset multiplot; clear;";  // gnucmds
4  GNU_DEFAULTS.resetterm = "set term x11; set output"; // after printing to a file, this is how
5   // the output is redirected to the window
6  GNU_DEFAULTS.font      = "Times-Roman";
7  GNU_DEFAULTS.fontsize  = 11;
8  GNU_DEFAULTS.epscolor  = "postscript eps enh color blacktext solid lw 2 " ...
9      + "\" + GNU_DEFAULTS.font  + "\" + text(2*GNU_DEFAULTS.fontsize, " %g")
10  GNU_DEFAULTS.epsmono  = "postscript eps enh mono blacktext solid lw 2 " ...
11      + "\" + GNU_DEFAULTS.font  + "\" + text(2*GNU_DEFAULTS.fontsize, " %g")

6.1.16  gnusplot

Format:  gnusplot(data/, filename/, fmt//)

Arguments:

1  data:

   (a) string vector, where each string is either a filename containing the data to be plotted (according
       to directions from the respective gnuformat) or an instruction telling Gnuplot what to do;
   (b) list, with entries 1,2,\ldots, where these are sublist which entries are:
1    4 1 2 3, real vector, real vector, real matrix, single set of data to be plotted;
6.1. GNUPLOT

  2 string, a filename containing data or an instruction to Gnuplot what to do;
  (c) list of the form << x, y, z >>, data set to be plotted.

  2 filename, string, tells Gnuplot to save the plot in a file in format that follows;
  3 fmt, string, format of the plot. See above for defaults.

Abstract: Creates a 3-D plot of a single or multiple data sets. Use gnuformat to specify whether particular
dataset will be plotted as a scatter plot (‘‘with points’’) or as a surface plot (‘‘with lines’’).

Low level functions

6.1.17 gnucmd
Format: gnucmd(cmd)
  1 cmd, string vector, a sequence of instructions to be sent to the default Gnuplot device.

6.1.18 gnuprint
Format: gnuprint(x), gnuprint(x,y,z)
Arguments:
  1 x, real matrix, the data to be sent to Gnuplot for 2-D plotting; or, real vector, x-coordinates of the
     points (scatter) or of the mesh (surface) for 3-D plotting;
  2 y, real vector, y-coordinates of the points (scatter) or of the mesh (surface) for 3-D plotting;
  3 z, real matrix, for surface plot its entries are z_{j,k} = z(x_j, y_k); or, real vector, for scatter plot its entries
     are z_i = z(x_i, y_i).

Abstract: Sends data to the default Gnuplot device. The format of the data determines whether it is a
surface or a scatter plot. See Gnuplot manual for more details.

Note: Sending really large data sets can choke Gnuplot (cause a segmentation fault). In that case one should
revert to the strategy used by the older version of the libgnuplot.so library, that is, write everything to a
Gnuplot-compatible file and then tell Gnuplot to load it.
Examples

Example: Consider a rlabor code that produces a simple plot shown in Fig. [6.1] on p. [79]

```r
//
// file: main1.r
//

rfile libgnuplot.so

rand ("normal",0,0.05);

t = [];
x = const.pi * [;0:2:1/32]';
y = sin(x);
z = cos(x);
for (i in 1:10)
{
  dy1 = rand(x);
dx1 = rand(x);
y1 = sin(x+dx1) + dy1;
s1 = 0.05*ones(x);
t = [t; [(x+dx1)/pi, y1]];
}
t = t[sort(t[;1]).ind;];

gnuwins (1);

gnuxlabel ("x / \{/Symbol p\}");
gnuylabel ("y");
gnulimits (0,2,-1.1,1.1);
gnuytics (1/4, 5);
gnuxtics (1/4, 5);
gnucmd ("unset grid; set grid xtics ytics;");
gnulegend ("y=sin(x)", "random noise added");
gnuformat ("with lines", "with points");
gnuplot ("<a=[x/pi,y];b=t[;1,2]>>");
```
Figure 6.1: An example of a single plot, of default size 1,1, imported without resizing into this tex document.
Example: Consider a R code that produces a simple plot shown in Fig. (6.1) on p. This time we create a Gnuplot file, which we then load inside the Gnuplot application by typing

```
load 'testfile.gnu'
```

at the Gnuplot prompt. See next page for how the file `testfile.gnu` looks like.

```r
// file: main7.r
//
// rng (1, "normal", [0,0.05]);

rng (1, "normal", [0,0.05]);

t = [];
x = const.pi * [;0:2:1/32]';
y = sin(x);
z = cos(x);
for (i in 1:10)
{
  dy1 = rand(x);
dx1 = rand(x);
y1 = sin(x+dx1) + dy1;
s1 = 0.05*ones(x);
t = [t; [(x+dx1)/pi, y1]];
}
t = t[sort(t[:,1]).ind;];

I = gnustart("./testfile.gnu");
gnuwin(I);
gnuxlabel ("x / \{/Symbol p\}");
gnuylabel ("y");
gnulimits (0,2,-1.1,1.1);
gnuytics (1/4, 5);
gnuxtics (1/4, 5);
gnucmd ("unset grid; set grid xtics ytics;");
gnulegend ("["y=sin(x)", "random noise added"]");
gnuformat ("["with lines", "with points"]");
gnuplot(<<a=[x/pi,y];b=t[:,1,2]>>);
gnuclose(I);
```
Listing of file testfile.gnu created by the rlab script from the previous page.

```plaintext
unset grid; set grid xtics ytics;
set xrange [0:2]; set yrange [-1.1:1.1];
set xlabel 'x / \{Symbol p\}'; unset x2label
set ylabel 'y'; unset y2label
set format x '%g'; set xtics 0.25;set mxtics 5;
set format y '%g'; set ytics 0.25;set mytics 5;
set nologscale xy;
set nologscale x2; set nologscale y2;
plot '-' title 'y=sin(x)' , '-' with points title 'random noise added'
  0  0
  0.03125 0.0980171
  0.0625  0.19509
  0.09375 0.290285
...
  1.96875 -0.0980171
  2  -2.44921e-16
  e
  -0.0301722 -0.183424
  -0.0265165  0.00249443
  -0.0233749 -0.147465
...
  2.01231  0.0139905
  2.02181  0.0574628
  e
unset xlabel
unset ylabel
set xtics;
set ytics;
```
Example: Consider a script that produces two plots in a single plot window, Fig. (6.9) on p. 112.

```plaintext
1 //
2 // file: main2.r
3 //
4 rng (1, "normal", [0,0.05]);
5 t = [];
6 x = const.pi * [;0:2:1/32]';
7 y = sin(x);
8 z = cos(x);
9 for (i in 1:10)
10 {
11   dy1 = rand(x);
12   dx1 = rand(x);
13   y1 = sin(x+dx1) + dy1;
14   s1 = 0.05*ones(x);
15   t = [t; [(x+dx1)/pi, y1]];
16 }
17 t = t[sort(t[:,1]).ind,];
18 gnuwins(1, "set term postscript enh color; set output 'gnuplot2.eps'");
19 gnucmd("set multiplot");
20 // plot no.1 in the same device
21 gnucmd("set origin 0,0.55");
22 gnucmd("set size 1,0.45");
23 gnuxlabel ("x / {/Symbol p}");
24 gnuylabel ("y");
25 gnulimits (0,2,-1.5,1.5);
26 gnuxtics (1/2,5);
27 gnuytics (1/2,5);
28 gnulegend ("y=sin(x)");
29 gnuformat ("with lines");
30 gnuplot (<<a=[x/pi,y]>>);
31 // plot no. 2 in the same device
32 gnucmd("set origin 0,0");
33 gnucmd("set size 1,0.45");
34 gnuxlabel ("x / {/Symbol p}");
35 gnuylabel ("y");
36 gnuxtics (1/2,5);
37 gnuytics (1/2,5);
38 gnulegend ("random noise");
39 gnuformat ("with points");
40 gnuplot (<<b=t[:,1]>>);
41 gnuclose(1);
```
Figure 6.2: Example of multiplot.
Example: Consider a script that produces two plots in a single plot window, Fig. 6.10 on p. 114.

```plaintext
rng (1, "normal", [0, 0.05]);
t = [];
x = const.pi * [;0:2:1/32]';
y = sin(x);
z = cos(x);
for (i in 1:10)
{
y1 = sin(x+dx1) + dy1;
s1 = 0.05*ones(x);
t = [t; [(x+dx1)/pi, y1]];
}
t = t[sort(t[;1]).ind;;];
gnuwins(1, "eps", stderr());
gnucmd ("set output 'gnuplot3.eps';");
gnucmd ("set size 2,2");

// big plot, takes the whole window
gnucmd("set multiplot");
gnucmd("set origin 0,0");
gnuxlabel ("{/=44 x} / {/Symbol=44 p}");
gnuxtics (0.5,5);
gnuytics (0.5,5);
gnucmd("unset grid");
gnulimits (0,2,-1.5,1.5);
gnulegend ( ["y=sin(x)","random noise"]);
gnufORMAT ( ["with lines","with points"]);
gnuplot ( <<a=[x/pi,y];b=t[;1,2]>> );

// small plot that is inserted on top of the big plot
gnucmd("set origin 1.08,1.08");
gnucmd("set size 0.76,0.76");
gnuxtics (0.5,5);
gnuytics (0.5,5);
gnucmd("clear");
gnulegend (["y=cos(x)"]);
gnuplot ( <<a=[x/pi,z]>> );
gnucmd("unset size; unset origin");
gnuclose(1);
```
Figure 6.3: Example of multiplot.
Example: Consider a script that creates a plot that is combination of plotting the data sets from files, here "tryme1.dat" and "tryme2.dat," then a data set created in rlab directly and after it has been filtered within Gnuplot, as shown in Fig. 6.11, p. 116.

```r
// file: main5.r: making of a strawberry/rhubarb pie

rng(1, "uniform", [0,1.5]);
t = rand (1000,1);

rng(2, "uniform", [0,1.11]);
t = [t, rand (1000,1)];

gnuwins(1, "set term postscript enh color; set output 'gnuplot5.eps'");
gnuxlabel ("Parameter {/Symbol strawberry} (strawberry)");
gnuylabel ("Parameter {/Symbol rhubarb} (rhubarb)");
gnulimits (0,1.5,0,1.5);
gnuytics (0.1, 2);
gnuxtics (0.1, 2);
gnucmd("unset grid; set grid xtics ytics;");
gnuformat ( [ ...
  "using ($2):($1) with lines", ...
  "using ($2):($1) with lines", ...
  "with lines", ...
  "with points", ...
  "using 1:($1+$2<1.11 ? $2 : 1/0) with points ", ...
] );
gnulegend ( [ ...
  "file tryme1.dat", ...
  "file tryme2.dat", ...
  "expression y = 1.11-x evaluated inside Gnuplot", ...
  "random points generated inside rlab", ...
  "same points but filtered inside Gnuplot" ...
] );
data = <<<>
  data.[1] = "tryme1.dat"; // file containing data to be plotted
  data.[2] = "tryme2.dat"; // the same
  data.[3] = "1.11-x"; // gnuplot expression we want plotted
  data.[4] = t; // some rlab generated data
  data.[5] = t; // some rlab generated data to be filtered by gnuplot
  gnuplot(data);
  gnuclose(1);
```
Figure 6.4: Example of single plot that combines data provided in separate files, an expression computed by Gnuplot, then data provided by *rlab*, and the same data filtered by Gnuplot.
Example: Consider a script that creates a two-surface plot in Fig. 6.12, p. 118.

```r
// file: main6.r

x = [0:1:1/16]';
y = [0:1:1/16]';
z = zeros(length(x),length(y));
for (i in 1:length(x))
{
  for (j in 1:length(y))
  { z[i,j] = x[i].^2 + y[j].^2; }
}

gnuwins(1, "set term postscript enh color; set output 'gnuplot6.eps'");
gnuxlabel("x");
gnylabel("y");
gnuzlabel("Function \( F(x,y) \) = \sin(\pi x)\sin(\pi y)\ by gnuplot");
gnulimits(0,1,0,1,-1,2);
gnuytics(1/4, 5);
gnuxtics(1/4, 5);
gnuztics(1/2, 5);
gnulegend( ["x^2+y^2 calculated inside rlab", ...
            "\sin(\pi x)\sin(\pi y)\ by gnuplot" ... ] );
gnuformat("with lines");
data = <<>>;

// first data set: from rlab
data[1].x = x;
data[1].y = y;
data[1].z = z;

// second data set: a function calculated by gnuplot
data[2] = "\sin(\pi x)\sin(\pi y)";
gnusplot(data);
gnuclose(1);
```
Function $\Phi = \Phi(x, y)$ calculated inside rlab

$\sin(\pi x) \cdot \sin(\pi y)$ by gnuplot

Figure 6.5: Example of surface (3-D) plot.
6.2 Pgplot

Long time ago, rlab used to rely on fortran pgplot library by [Pearson June 1989] for plotting. Then, it was shifted to PLplot by [LeBrun and Furnish 2014], which the author has difficulties to compile for some platforms. Then, it was decided to go back to pgplot, as it was distributed with rlab, so there was full control over the source code.

The reason why one would choose pgplot over gnuplot is the speed. E.g., if one wants to follow results of computation in-vivo, so to speak, then communication with gnuplot through pipe is too slow. test/pgplot/2D/ directory contains an example that directly compares the speed of two interfaces, which gives that pgplot at least 7 times as fast as gnuplot for small data sets. For large and huge datasets it is probably even higher.

The library ${RLAB}/rlib/pgplot.r3 is work in progress started with rlab creator Ian Searle. Since the version described here, however, the approach has changed in that from rlab user perspective commands for configuring plots resemble gnuplot interface to rlab. Basically, for most plots if one takes working gnuplot script and replace all gnu’s with pl’s or pg’s this should create reasonably similar plot.

Currently, pgplot facility for rlab supports 2-dimensional plots such as scatter or histograms, and also includes contours and color maps.

The rlabplus access to pgplot comprise of scripted library ${RLAB}/rlib/pgplot.r3, and set of low-level functions written in c-language that are part of rlab, has user chosen to build it with --with-pgplot flag.
Initialization/termination functions

6.2.1 plwins

I Format: \( \langle\langle \text{act}; \text{dev}; \text{win} >\rangle = \text{plwins}() \)

Abstract: Obtains the information about open pgplot streams: \( \text{act} \) holds the index of the active (default) stream; \( \text{win} \) is an array containing the index of all open streams; while \( \text{dev} \) lists the devices of each stream (file, or terminal).

II Format: \( \text{plwins}(N, s, [xwid, ywid]) \)

Arguments:

1. \( N \), integer, desired number of open plplot streams (plotting windows);
2. \( s \), type of stream, typically “xwin”.
3. \( [xwid, ywid] \), two-column real matrix, contains x- and y-width (pixel count) of the plotting windows for each stream.

Abstract: Manages the number of open streams, and brings them to “N” per user request. If streams already exist, then repeated call with this function doesn’t do anything (so repeated execution of the same script does not mess-up subsequent user manipulations of plot windows, e.g., resizing or repositioning).

6.2.2 plwin

I Format: \( I = \text{plwin}() \)

Abstract: Obtain the index of active (or default) stream.

II Format: \( \text{plwin}(I) \)

Arguments:

1. \( I \), integer, index of active stream.

Abstract: Sets the index of an active stream. If stream has been used previously (repeated execution of the same script) then clears its previous content.

Note: It is good practice to start any plotting module with \( \text{plwin}(J) \), with \( J \) being desired stream. Conversely, \( \text{plwins}(N) \) should be put at the beginning of the main script (that calls all the other modules).

Note: Unlike plplot, in pgplot it is not possible to position plot windows on the desktop. First time they are created, they are put horizontally in the middle and vertically at the top.
Plotting

6.2.3  pltext

**Format:** \texttt{pltext(s, loc, scale, opts)}

1. \( s \), string, graffitti to be put on top of a graph;
2. \( loc = [x, y] \), real vector, position of the graffitti in graph coordinates;
3. \( scale \), scalar, relative scale of the graffitti in terms of the default font scale;
4. \( opts \), list with entries \(<incl, angle; just; color>\), where \( incl/angle \) gives the inclination or angle of the text either in terms of angle or direction vector \([dx, dy]\); \( just \) is justification which can be either string “r” for right, “l” for left, or “c” for center, or number in range 0 to 1 where 0 stands for left, 0.5 for center, or 1 for right; \( color \) is an integer in range 1 to 15 for a color from the pallete (don’t use 0, this is background color, sort of kind of invisible ink).

**Abstract:** Writes graffitti on top of graph according to specifications by user on default plplot stream.

6.2.4  pltitle

**Format:** \texttt{pltitle(s, scale)}

**Arguments:**

1. \( s \), string, the title of the plot
2. \( scale \), scalar, relative scale of the title text in terms of the default font scale;

**Abstract:** Writes the title of the graph on default plplot stream.

6.2.5  plxtics, plx2tics, plytics, ply2tics

**Format:** \texttt{plxtics(v1/.,v2,.scale/)}

**Arguments:**

1. \( v1 \), real scalar, increment in coordinate in combination with \texttt{plimits} at which a tick is put.
2. \( v2 \), scalar, number of minor tics to be placed between two major tics;
3. \( scale \), scalar, relative scale of the tic text in terms of the default font scale;

**Abstract:** Regulate primary and secondary axes, major and minor x- and y- tics, and their relative scale.

6.2.6  pllimits, pllimits2

**Format:** \texttt{pllimits(xlo, xhi, ylo, yhi, zlo, zhi)},

**Arguments:**

1.2. \( xlo, xhi \), real scalar, min and max values of the primary or secondary x-axis.
3.4. \( ylo, yhi \), real scalar, min and max values of the primary or secondary y-axis.
5.6 \textit{zlo, zhi,} real scalar, min and max values of the primary z-axis.

**Abstract:** Set the limits of the plot.

**Note:** Entries can be left empty, then script figures values from the data.

### 6.2.7 plscale, plscale2

**Format:** \texttt{plscale(scalex, scaley)}

**Arguments:**

1. \textit{scalex}, string “lin” or “log”.
2. \textit{scaley}, string “lin” or “log”.

**Abstract:** Sets the grid of active plot window to linear or logarithmic.

### 6.2.8 plxgrid, plxgrid2, plygrid, plygrid2

**Format:** \texttt{plxgrid(stylemaj, stylemin)}

**Arguments:**

1. \textit{stylemaj}, string, gnuplot-compliant string that describes the “major” grid line, e.g., “[with lines] lt 1 lw 2 lc rgb ’brown’”.
2. \textit{stylemin}, string, gnuplot-compliant string that describes the “minor” grid line.

**Abstract:** Sets the grid properties of the active stream. If no arguments are provided then resets grids to their default values.

**Note:** Use this function after \texttt{plscale()}.

### 6.2.9 plxlabel, plx2label, plylabel, ply2label

**Format:** \texttt{plxlabel(s, specs)}

**Arguments:**

1. \textit{s}, string, label for the primary and secondary x- and y-, or primary z-axis.
2. \textit{specs}, up to 4-dim positive real vector, where specs[1] is the scale of the label, specs[2] is the vertical (horizontal) offset of the label with respect to the axis;

**Abstract:** Set the axes labels.

### 6.2.10 plegend

**Format:** \texttt{plimits(L)}

**Arguments:**

1. \textit{L}, string vector, labels for each data set to follow in \texttt{gnuplot( ... )}.

**Abstract:** Set the names of the datasets for the plot.
6.2.11 plformat

Format: plformat(K)

Arguments:

1. K, string vector, where each entry is a gnuplot-compliant directions how to plot the dataset.

6.2.12 plot

Format: plot(data/,filename/)

Arguments:

1. data:

   (a) real matrix, data to be plotted is in format \([x, y_1, y_2, \ldots]\);

   (b) list, then each entry in the list is assumed to be a data set to be plotted following a prescription in plformat.

2. filename, string, this assumes that pgplot will first draw the plot in a desktop window, then save it to the file. The filename has to be pgplot compliant, e.g., for landscape color post-script “./somefilename.eps/CPS” and so forth.
Example: Consider a rlab code that produces a simple plot shown in Fig. (6.6) on p. 97.

```c
// Illustration of logarithmic scale and two y-axes
// ---
if (length(plwins().win)<2)
    { plwins (2,["/XWIN", "pgplot_eg4.eps/CPS"]); }

// Set up data for log plot
N = 100;
exp_fmax=5;
f0 = 1.0; // corner frequency
freq = 10 .^ freql;
ampl = 20.0 .* log10( 1.0 ./ sqrt(1.0 + (freq ./ f0 ).^2) );
phase = -rad * atan2(freq,f0);

//
//
//
plwin (1,[0.125,0.85,0.15,0.9]);
plimits (1e-2,1000,-60,0);
plimits2 (,,-90,0);
plytics (10,2);
pscale ("log");
plytics2 (10,5);
pscale2 ("log");
ply2tics (10,5);
plegend ("Amplitude", "Phase", 1, "brit");
pxxlabel ("Frequency (Hz)");
pxylabel ("Amplitude (dB)",[1,4.0]);
ply2label ("Phase shift (deg)",[1,4.5]);
//plxgrid ("lt 2 lw 5 lc rgb grey","lt 3 lw 1 lc rgb grey");
//plygrid ("lt 2 lw 5 lc rgb grey","lt 3 lw 1 lc rgb grey");
pformat ([... "with lines lt 1 lw 5 lc rgb blue using 1:2 axes x1y1", ... "with lines lt 1 lw 5 lc rgb brown using 1:3 axes x1y2", ... ]); ptext("-20 dB/decade", [50, -30], 1.25, <<color=10;angle=-50>>); plot([freq,ampl,phase]);
```
plwin (2,[0.125,0.85,0.15,0.9]);
colors("red");
"NOTE: When plotting to a file don't forget 'plclose()' so that\n"?
"NOTE: PGPLOT writes the image to the file and close it!\n"?
colors();

plimits (1e-2,1000,-60,0);
plimits2 (,,90,0);
plscale ("log");
plscale2 ("log");
ply2tics (10,5);
plegend (["Amplitude", "Phase"], 1, "bli");
plxlabel ("Frequency (Hz)"");
plylabel ("Amplitude (dB)",[1,3.5]);
ply2label ("Phase shift (deg)",[1,4]);
plxgrid ("lt 4 lw 1 lc rgb grey");
plygrid ("lt 4 lw 1 lc rgb grey");
plformat ([... "with lines lt 1 lw 5 lc rgb blue axes x1y1", ... "with lines lt 1 lw 5 lc rgb brown axes x1y2", ... ]);
pltext("-20 dB/decade", [0.53, 0.5], 1.25, <<color="rgb magenta";angle=-50;cs="n">>);
plot( <<a1=[freq,ampl];a2=[freq,phase]>> );
plclose();
Figure 6.6: Pgpplot demonstration of plotting two datasets each with its own vertical scale, in multicolors and with some text on top for good measure. Please note: the plot was saved as a landscape embedded postscript (CPS in pgplot parlance) and then converted to pdf using utility eps2pdf, and rotated by 270° for this page.
Example: Consider a rlab code that produces a simple plot shown in Fig. (6.7) on p. 100.

```rlab
if (!exist(NITER))
  { NITER = 1; }

pm3d_format = [...] "pm3d grey 0,1 cbrange [-1:1]", ...
  "pm3d map cbrange [-1:1]", ...
  "pm3d grad 8,4 cbrange [-1:1]", ...
  "cont lev disc 0.25,0.75 lt 1 lw 1 lc rgb magenta", ...
  "cont lev incr -1,0.5,1 cbrange [-1:1] lt 1 lw 1 lc grey 2,3 labels '%.1f' ts 0.75", ...
  "cont lev incr -1,0.5,1 lt 1 lw 1 lc map cbrange [-1:1]", ...
[];

xl = 10;
x = [0:xl:1/32]';
y1 = sin(pi * (x + 0.1*(uniform()-0.5)));
y2 = cos(pi * (x + 0.1*(uniform()-0.5)));

// for greymap
y = [-1:1:1/32]';
z = zeros(length(x),length(y));
for (i in 1:length(x))
{
  for (j in 1:length(y))
  {
    z[i;j] = sin(0.25 *pi .* x[i] .* y[j]);
  }
}
doclose=0;

np = range(pm3d_format);
for (i in 1:NITER)
{
  //xwid = 7 + 4 * uniform();
  //ywid = 5 + 2 * uniform();
  xwid = 8;
  ywid = 6;
  // idx_p = sample(np,1);
  idx_p = 2;
```
47 _pm3d_1 = pm3d_format[idx_p];
48 tic();
49 //if (uniform()>0.5)
50 if (1)
51 {
52     plwins(1,"/XWIN",[xwid,ywid]);
53 }
54 else
55 {
56     plclose ();
57     plwins(1,"pgplot_eg10_"+num2str(idx_p,"%.0f")+.eps/CPS",[xwid,ywid]);
58     doclose=1;
59 }
60 }
61 plwin (1,[0.125,0.75,0.15,0.9]);
62 plimits (0,xl,-1,1);
63 plytics (1,2);
64 plytics (0.5,5);
65 plztics (0.1);
66 plxlabel ("my x-axis", [1.75,2]);
67 plylabel ("my y-axis", [1.75,2]);
68 plegend (["Magnitude"], 1.0, "trib");//top-right-inside-with bounding box
69 plformat (_pm3d_1);
70 plot (<<x=x;y=y;z=z>>);
71 toc()?
72 }
73 if (doclose)
74 {
75     plclose ();
76     doclose = 0;
77 }
78 else
79 {
80     //      sleep(1);
81 }
82 }
Figure 6.7: Another Pgplot demonstration of plotting a 3-d data set through denoted contour lines (top) or through color map gnuplot-style (pgplot library was enhanced with the gnuplot-like pm3d curves, but this will be published elsewhere). Please note: the plot was saved as a landscape embedded post script (CPS in pgplot parlance) and then converted to pdf using utility \texttt{eps2pdf}, and rotated by 270° for this page.
6.3 Plplot

PLplot by LeBrun and Furnish (2014) is a cross-platform software package for creating scientific plots whose (UTF-8) plot symbols and text are limited in practice only by what Unicode-aware system fonts are installed on a user’s computer. The PLplot software, which is primarily licensed under the LGPL, has a clean architecture that is organized as a core C library, separate language bindings for that library, and separate device drivers that are dynamically loaded by the core library which control how the plots are presented in noninteractive and interactive plotting contexts.

The reason why one would choose plplot over gnuplot is the speed. E.g., if one wants to follow results of computation in-vivo, so to speak, then communication with gnuplot through pipe is too slow. test/plplot/2D/ directory contains an example that directly compares the speed of two interfaces, which gives that plplot at least 7 times as fast as gnuplot for small data sets. For large and huge datasets it is probably even higher.

The library ${\texttt{RLAB}}$/rlib/plplot.r is work in progress started with rlab creator Ian Searle. Since the version described here, however, the approach has changed in that from rlab user perspective commands for configuring plots resemble gnuplot interface to rlab. Basically, for most plots if one takes working gnuplot script and replace all gnu’s with pl’s this should create reasonably similar plot, with that difference that gnucmd does not have an equivalent.

The rlabplus access to plplot comprise of scripted library ${\texttt{RLAB}}$/rlib/plplot.r, and set of low-level functions written in c-language that are part of rlab, has user chosen to build it with --with-plplot flag.

Note: The scripted library plplot.r utilizes cells, thus it won’t work on installations of rlab prior to 2.5.x.x.

### Initialization/termination functions

#### 6.3.1 plwins

I **Format:** $<< \texttt{default}; \texttt{available}; \texttt{device} >> = \texttt{plwins}()$

**Abstract:** Obtains the information about open plplot streams: default holds the index of the default plplot stream; available is an array containing the index of all open plplot streams; while device contains the information about the stream.

II **Format:** $\texttt{plwins}(N, s, [xwid, ywid], [xoff, yoff])$

**Arguments:**

1 $N$, integer, desired number of open plplot streams (plotting windows);
2 $s$, type of stream, typically “xwin”.
3 $[xwid, ywid]$, two-column real matrix, contains x- and y-width (pixel count) of the plotting windows for each stream.
4 $[xoff, yoff]$, two-column real matrix, contains x- and y-offsets (pixel count) of the plotting windows for each stream.

**Abstract:** Manages the number of open plplot streams, and brings them to “N” per user request. If streams already exist, then repeated call with this function doesn’t do anything (so repeated execution of the same script does not mess-up subsequent user manipulations of plot windows, e.g., resizing or repositioning).
6.3.2  plwin

I Format: \( I = \text{plwin()} \)

Abstract: Obtain the index of a default plplot stream.

II Format: plwin(I)

Arguments:

1 \( I \), integer, index of an open plplot stream.

Abstract: Sets the index of a default plplot stream. If stream has been used previously (repeated execution of the same script) then clears its previous content.

Note: It is good practice to start any plplot module with \( \text{plwin}(J) \), with \( J \) being desired stream. Conversely, \( \text{plwins}(N) \) should be put at the beginning of the main script (that calls all the other modules).

Plotting

6.3.3  pltext

Format: \( \text{pltext}(s, \text{loc}, \text{size}, \text{opts}) \)

1 \( s \), string, graffitti to be put on top of a graph;

2 \( \text{loc} = [x, y] \), real vector, position of the graffitti in graph coordinates;

3 \( \text{size} \), scalar, relative size of the graffitti in terms of the default font size;

4 \( \text{opts} \), list with entries \( \ll \text{incl}; \text{just}; \text{color} \gg \), where \( \text{incl} \) is inclination of the text in terms of \( [dx, dy] \) from the center of the text; \( \text{just} \) is justification which can be either string “r” for right, “l” for left, or “c” for center, or number in range 0, for right, 0.5 for center, or 1 for left; \( \text{color} \) is an integer in range 1 to 15 for a color from plplot’s color0 pallete (don’t use 0, this is background color, sort of kind of invisible ink).

Abstract: Writes graffitti on top of graph according to specifications by user on default plplot stream.

6.3.4  pltitle

Format: \( \text{gnutitle}(s, \text{size}) \)

Arguments:

1 \( s \), string, the title of the plot

2 \( \text{size} \), scalar, relative size of the title text in terms of the default font size;

Abstract: Writes the title of the graph on default plplot stream.
6.3.5 plxtics, plx2tics, plytics, ply2tics, plztics

**Format:** `gnuxtics(v1, v2, size)`

**Arguments:**

1. `v1`, real scalar, desired number of major x-, y-tics (plot) or z-tics (splot).
2. `v2`, scalar, number of minor tics to be placed between two major tics;
3. `size`, scalar, relative size of the tic text in terms of the default font size;

**Abstract:** Regulate primary and secondary, major and minor x- and y- tics, and the primary major and minor z-tics, and their character size.

6.3.6 plimits, plimits2

**Format:** `plimits(xlo, xhi, ylo, yhi, zlo, zhi)`

**Arguments:**

1. `xlo, xhi`, real scalar, min and max values of the primary or secondary x-axis.
2. `ylo, yhi`, real scalar, min and max values of the primary or secondary y-axis.
3. `zlo, zhi`, real scalar, min and max values of the primary z-axis.

**Abstract:** Set the limits of the plot.

**Note:** Entries can be left empty, then script figures values from the data.

6.3.7 plsca, plsca2

**Format:** `plsca(scalex, scaley)`

**Arguments:**

1. `scalex`, string “lin” or “log”. Resets the x-axis style to default value “bcnst” and then set logarithmic scale if necessary.
2. `scaley`, string “lin” or “log”. Resets the y-axis style to default value “bcnst” and then set logarithmic scale if necessary.

**Abstract:** Sets the default linear or logarithmic grid.

6.3.8 plgrid, plgrid2

**Format:** `plgrid(xspec, yspec)`

**Arguments:**

1. `xspec`, if real scalar then 0,1,2 for no grid, grid with major x-tics, or grid with major and minor x-tics;
   if string then full description of x-axis style in plplot parlance, default value is “bcnst”.

2. `yspec`,...
2 \textit{yspec}, if real scalar then 0,1,2 for no grid, grid with major y-tics, or grid with major and minor y-tics; if string then full description of y-axis style in plplot parlance, default value is “bnstu”.

\textbf{Abstract:} Sets the grid properties of the default stream. If no arguments are provided then resets grids to their default values.

\textbf{Note:} Use this function after \texttt{plscale()}.  

6.3.9 \texttt{plxlabel, plx2label, plylabel, ply2label, plzlabel}  

\textbf{Format:} \texttt{plxlabel(s, size)}  

\textbf{Arguments:}  

1 \textit{s}, string, label for the primary and secondary x- and y-, or primary z-axis.  

2 \textit{size}, scalar, relative size of the tic text in terms of the default font size;  

\textbf{Abstract:} Set the axes labels.

6.3.10 \texttt{gnulegend}  

\textbf{Format:} \texttt{gnulimits(L)}  

\textbf{Arguments:}  

1 \textit{L}, string vector, labels for each data set to follow in \texttt{gnuplot(...)}.  

\textbf{Abstract:} Set the names of the datasets for the plot.

6.3.11 \texttt{gnuformat}  

\textbf{Format:} \texttt{gnuformat(K)}  

\textbf{Arguments:}  

1 \textit{K}, a string vector, a directions to Gnuplot what to do with each data set that follows in \texttt{gnuplot(...)} or \texttt{gnusplot(...)}.  

6.3.12 \texttt{gnuplot}  

\textbf{Format:} \texttt{gnuplot(data/, filename/, fmt/\})  

\textbf{Arguments:}  

1 \textit{data}:  

(a) real matrix, data to be plotted is in format \([x, y_1, y_2, \ldots]\);  

(b) string vector, then each string is either a filename of the data to be plotted (according to directions from the respective \texttt{gnuformat}) or a string to be passed directly to Gnuplot (where we assume that Gnuplot will know what to do with it, e.g., plot a functions);
(c) list, then each entry in the list is assumed to be a data set to be plotted: if an entry is a matrix then gnuformat knows what to do with it (e.g., three column matrix containing the \([x, y, dy]\) has respective gnuformat of “with errorlines”, and so on); if an entry is a string then it is assumed a filename or it is passed to Gnuplot as a string.

2 \textit{filename}, string, this assumes that Gnuplot is going to draw the plot into a file, instead of to an open x11 session. This is useful once visual appearance of the plot is satisfactory - it is time then to export the plot to a file for incorporation into a publication.

3 \textit{fmt}, string, a Gnuplot compatible specification of the format the plot will be written to the \textit{filename} with - basically, everything that follows \texttt{set term} command in Gnuplot. Default format is that given in GNU_DEFAULTS.epscolor, see below. These are also available if one chooses two special values, “color” or “mono” for \textit{fmt}. If you have your special choice of values, these may be set in file \texttt{libgnuplot.so.r},

\begin{verbatim}
1 GNU_DEFAULTS = "";
2 GNU_DEFAULTS.debug = 0;
3 GNU_DEFAULTS.resetcmd = "unset multiplot; clear;";  // gnuwins
4 GNU_DEFAULTS.resetterm = "set term x11; set output";  // after printing to a file, this is how
5 // the output is redirected to the window
6 GNU_DEFAULTS.font = "Times-Roman";
7 GNU_DEFAULTS.fontsize = 11;
8 GNU_DEFAULTS.epscolor = "postscript eps enh color blacktext solid lw 2 " ...
9 + \\
10 + GNU_DEFAULTS.font + \\
11 + \\
12 + GNU_DEFAULTS.fontsize, " %g";)
13 GNU_DEFAULTS.epsmono = "postscript eps enh mono blacktext solid lw 2 " ...
14 + \\
15 + GNU_DEFAULTS.font + \\
16 + \\
17 + GNU_DEFAULTS.fontsize, " %g";)
\end{verbatim}

6.3.13 gnusplot

\textbf{Format:} \texttt{gnusplot(data/,filename/,fmt//)}

\textbf{Arguments:}

1 \textit{data}:

(a) string vector, where each string is either a filename containing the data to be plotted (according to directions from the respective gnuformat) or an instruction telling Gnuplot what to do;

(b) list, with entries 1,2,…, where these are sublists which entries are:

1 \(x, y, z\), real vector, real vector, real matrix, single set of data to be plotted;

2 string, a filename containing data or an instruction to Gnuplot what to do;

(c) list of the form \texttt{<<x, y, z>>}, data set to be plotted.

2 \textit{filename}, string, tells Gnuplot to save the plot in a file in format that follows;

3 \textit{fmt}, string, format of the plot. See above for defaults.

\textbf{Abstract:} Creates a 3-D plot of a single or multiple data sets. Use gnuformat to specify whether particular dataset will be plotted as a scatter plot (“\texttt{with points}”) or as a surface plot (“\texttt{with lines}”).
Low level functions

6.3.14  **gnucmd**

**Format:**  `gnucmd(cmd)`

1. `cmd`, string vector, a sequence of instructions to be sent to the default Gnuplot device.

6.3.15  **gnuprint**

**Format:**  `gnuprint(x)`, `gnuprint(x, y, z)`

**Arguments:**

1. `x`, real matrix, the data to be sent to Gnuplot for 2-D plotting; or, real vector, x-coordinates of the points (scatter) or of the mesh (surface) for 3-D plotting;
2. `y`, real vector, y-coordinates of the points (scatter) or of the mesh (surface) for 3-D plotting;
3. `z`, real matrix, for surface plot its entries are \( z_{j,k} = z(x_j, y_k) \); or, real vector, for scatter plot its entries are \( z_i = z(x_i, y_i) \).

**Abstract:**  Sends data to the default Gnuplot device. The format of the data determines whether it is a surface or a scatter plot. See Gnuplot manual for more details.

**Note:**  Sending really large data sets can choke Gnuplot (cause a segmentation fault). In that case one should revert to the strategy used by the older version of the libgnuplot.so library, that is, write everything to a Gnuplot-compatible file and then tell Gnuplot to load it.
Example: Consider a rlab code that produces a simple plot shown in Fig. (6.1) on p. 79.

```r
//
// file: main1.r
//

rfile libgnuplot.so

rand ("normal",0,0.05);

x = x = const.pi * [0:2:1/32];
y = sin(x);
z = cos(x);
for (i in 1:10)
{
  dy1 = rand(x);
dx1 = rand(x);
y1 = sin(x+dx1) + dy1;
s1 = 0.05*ones(x);
t = [t; [(x+dx1)/pi, y1]];
}
t = t[sort(t[;1]).ind;];
gnuwins (1);

gnuxlabel ("x / /Symbol p");
gnuylabel ("y");
gnuunits (0,2,-1.1,1.1);
gnuytics (1/4, 5);
gnuxtics (1/4, 5);
gnuflags ("unset grid; set grid xtics ytics; ");
gnulegend (["y=sin(x)", "random noise added"]);
gnuformat (["with lines", "with points"]);
gnuplot (<<a=[x/pi,y];b=t[;1,2]>>);
```
Figure 6.8: An example of single plplot plot, and how pen width affect the outcome (top panel has pen width 1, bottom pen width 8). While 1 is good for plotting on the computer monitor, when preparing plots for publication greater widths work better.
Example: Consider a rlab code that produces a simple plot shown in Fig. (6.1) on p. 79. This time we create a Gnuplot file, which we then load inside the Gnuplot application by typing

```
load 'testfile.gnu'
```

at the Gnuplot prompt. See next page for how the file testfile.gnu looks like.

```
//
// file: main7.r
//
rng (1, "normal", [0,0.05]);

t = [];
x = const.pi * [;0:2:1/32]';
y = sin(x);
z = cos(x);
for (i in 1:10)
{
  dy1 = rand(x);
dx1 = rand(x);
y1 = sin(x+dx1) + dy1;
s1 = 0.05*ones(x);
t = [t; [(x+dx1)/pi, y1]];
}
t = t[sort(t[;1]).ind;];
I = gnustart("./testfile.gnu");
gnuwin(I);
gnuxlabel("x / {/Symbol p}");
gnuylabel("y");
gnulimits (0,2,-1.1,1.1); 
gnuytics (1/4, 5); 
gnuxtics (1/4, 5); 
gnucmd ("unset grid; set grid xtics ytics;"); 
gnulegend ("y=sin(x)", "random noise added");
gnuformat ("with lines", "with points");
gnuplot(<a=[x/pi,y];b=t[1,2]>);
gnuclose(I);
```
Listing of file `testfile.gnu` created by the `rlab` script from the previous page.

```
unset grid; set grid xtics ytics;
set xrange [0:2]; set yrange [-1.1:1.1];
set xlabel 'x / {/Symbol p}'; unset x2label
set ylabel 'y'; unset y2label
set format x '%g'; set xtics 0.25; set mxtics 5;
set format y '%g'; set ytics 0.25; set mytics 5;
set nologscale xy;
set nologscale x2; set nologscale y2;
plot '-' title 'y=sin(x)', '-' with points title 'random noise added'
  0  0
  0.03125  0.0980171
  0.0625   0.19509
  0.09375  0.290285
...
  1.96875 -0.0980171
  2  -2.4492e-16
  e
-0.0301722 -0.183424
-0.0265165  0.00249443
-0.0233749 -0.147465
...
  2.01231  0.0139905
  2.02181  0.0574628
  e
unset xlabel
unset ylabel
set xtics;
set ytics;
```
Example: Consider a script that produces a two plots in a single plot window, Fig. (6.9) on p. 112.

```plaintext
//
// file: main2.r
//
rng (1, "normal", [0,0.05]);
t = [];
x = const.pi * [;0:2:1/32]';
y = sin(x);
z = cos(x);
for (i in 1:10) {
  dy1 = rand(x);
dx1 = rand(x);
y1 = sin(x+dx1) + dy1;
s1 = 0.05*ones(x);
t = [t; [(x+dx1)/pi, y1]];
}
t = t[sort(t[:1]).ind;];

gnuwins(1, "set term postscript enh color; set output 'gnuplot2.eps'");
gnucmd("set multiplot");
// plot no.1 in the same device
 gnucmd("set origin 0,0.55");
 gnucmd("set size 1,0.45");
 gnuxlabel ("x / {/Symbol p}");
 gnuylabel ("y");
 gnulimits (0,2,-1.5,1.5);
 gnuxtics (1/2,5);
 gnuytics (1/2,5);
 gnulegend ("y=sin(x)");
 gnuformat ("with lines");
 gnuplot (<<a=[x/pi,y]>>);
// plot no. 2 in the same device
 gnucmd("set origin 0,0");
 gnucmd("set size 1,0.45");
 gnuxtics (1/2,5);
 gnuxlabel ("x / {/Symbol p}" );
 gnuylabel ("y");
 gnulegend ("random noise");
 gnuformat ("with points");
 gnuplot (<<b=t[:1,2]>>);
 gnuclose(1);
```
Figure 6.9: Example of multiplot.
Example: Consider a script that produces two plots in a single plot window, Fig. (6.10) on p. 114.

```plaintext
rng (1, "normal", [0,0.05]);
t = []; x = const.pi * [;0:2:1/32]'; y = sin(x); z = cos(x);
for (i in 1:10)
    dy1 = rand(x);
dx1 = rand(x);
y1 = sin(x+dx1) + dy1;
s1 = 0.05*ones(x);
t = [t; [(x+dx1)/pi, y1]];
} t = t[sort(t[;1]).ind;];
gnuwins(1, "eps", stderr());
gnucmd ("set output 'gnuplot3.eps';"); gnucmd ("set size 2,2");
// big plot, takes the whole window
gnucmd("set multiplot"); gnucmd("set origin 0,0");
gnxlabel ("{/=44 x} / {/Symbol=44 p}""); gnuxtics (0.5,5);
gnuytics (0.5,5); gnucmd("unset grid");
gnulimits (0,2,-1.5,1.5); gnulegend ("[y=sin(x)","random noise"]"); gnufont ("["with lines","with points"]");
gnuplot (<<a=[x/pi,y];b=t[;1,2]>>);
// small plot that is inserted on top of the big plot
gnucmd("set origin 1.08,1.08");
gnucmd("set size 0.76,0.76");
gnuxtics (0.5,5); gnuytics (0.5,5);
gnucmd("clear"); gnulegend ("[y=cos(x)]");
gnuplot (<<a=[x/pi,z]>>);
gnucmd("unset size; unset origin");
gnuclose(1);
```
Figure 6.10: Example of multiplot.
Example: Consider a script that creates a plot that is combination of plotting the data sets from files, here “tryme1.dat” and “tryme2.dat,” then a data set created in rlab directly and after it has been filtered within Gnuplot, as shown in Fig. [6.11], p. 116.

// file: main5.r: making of a strawberry/rhubarb pie
rng(1, "uniform", [0,1.5]);
t = rand (1000,1);

rng(2, "uniform", [0,1.11]);
t = [t, rand (1000,1)];

gnuwins(1, "set term postscript enh color; set output 'gnuplot5.eps' ");
gnuxlabel ("Parameter {/Symbol strawberry} (strawberry)"");
gnuylabel ("Parameter {/Symbol rhubarb} (rhubarb)"");
gnulimits (0,1.5,0,1.5);
gnuytics (0.1, 2);
gnuxtics (0.1, 2);
gnucmd("unset grid; set grid xtics ytics; ");
gnuformat ( [ ...
   "using ($2):($1) with lines", ... 
   "using ($2):($1) with lines", ... 
   "with lines", ... 
   "with points", ... 
   "using 1:($1+$2<1.11 ? $2 : 1/0) with points 
   ]);
gnulegend ( [ ...
   "file tryme1.dat", ... 
   "file tryme2.dat", ... 
   "expression y = 1.11-x evaluated inside Gnuplot", ... 
   "random points generated inside rlab", ... 
   "same points but filtered inside Gnuplot" ... 
 ]);
data = <<< >>;
data.[1] = "tryme1.dat"; // file containing data to be plotted 
data.[2] = "tryme2.dat"; // the same 
data.[3] = "1.11-x"; // gnuplot expression we want plotted 
data.[4] = t; // some rlab generated data 
data.[5] = t; // some rlab generated data to be filtered by gnuplot 
gnuplot(data);
gnuclose(1);
Figure 6.11: Example of single plot that combines data provided in separate files, an expression computed by Gnuplot, then data provided by rlab, and the same data filtered by Gnuplot.
Example: Consider a script that creates a two-surface plot in Fig. [6.12], p. [118]

```plaintext
1 //
2 // file: main6.r
3 //
4
5 x = [0:1:1/16];
6 y = [0:1:1/16];
7 z = zeros(length(x),length(y));
8 for (i in 1:length(x))
9 {
10   for (j in 1:length(y))
11     { z[i;j] = x[i].^2 + y[j].^2; }
12 }
13
14 gnuwins(1, "set term postscript enh color; set output 'gnuplot6.eps'");
15 gnuxlabel ("x");
16 gnuylabel ("y");
17 gnuzlabel ("Function \( /Symbol F = /Symbol F(x,y) \)";
18 gnulimits (0,1,0,1,-1,2);
19 gnuytics (1/4, 5);
20 gnuxtics (1/4, 5);
21 gnutzics (1/2, 5);
22 gnulegend ( ["x^2+y^2 calculated inside rlab", ...
23     "sin(pi*x)*sin(pi*y) by gnuplot" ...
24     ] );
25 gnucmd ("set view 53,13 ");
26 gnucmd ("set grid ");
27 gnuformat ("with lines");
28 data = <<>>;
29 // first data set: from rlab
30 data[1].x = x;
31 data[1].y = y;
32 data[1].z = z;
33 // second data set: a function calculated by gnuplot
34 data[2] = "sin(pi*x)*sin(pi*y)";
35 gnusplot(data);
36 gnuclose(1);
```
Figure 6.12: Example of surface (3-D) plot.
6.4 Xfig

Xfig is an interactive drawing tool which runs under X Window System Version 11 Release 4 (X11R4) or later, on most UNIX-compatible platforms. It is freeware.

In xfig, figures may be drawn using objects such as circles, boxes, lines, spline curves, text, etc. It is also possible to import images in formats such as GIF, JPEG, EPSF (PostScript), etc. Those objects can be created, deleted, moved or modified. Attributes such as colors or line styles can be selected in various ways. For text, 35 fonts are available. Text can also include Latin-1 characters such as "a umlaut" or "c cedilla".

The library is located in rclass/xfig.r3 in the installation directory.

The library is written in script language, and is loaded upon starting rlab. Unlike other plotting libraries this library utilizes new feature of the rlab3, that is, availability of the classes.

This version of the library supports 2-dimensional plots.

Creation of an XFig object

6.4.1 xfig

Format: \( r = xfig() \)

Result: Xfig-class type object. Contains properties and methods.

Abstract:

Public methods that set basic properties of the XFig object

6.4.2 comment

Format: \( \text{comment}(s) \)

Arguments:

1 \( s \), string;

Abstract: Public method of an Xfig object which sets the property of the current Xfig object. The methods in this group set the properties that go in the header of the Xfig file (once the file is created). \text{comment} sets the text that is written after mandatory the "#FIG 3.2", which is the first line in the Xfig-file header.

6.4.3 justification

Format: \( \text{justification}(s) \)

Arguments:

1 \( s \), string, "center" or "left";

Abstract: Public method of an Xfig object which sets the property of the current Xfig object. The methods in this group set the properties that go in the header of the Xfig file (once the file is created).
6.4.4 orientation
Format: orientation(s)
Arguments:
   1 s, string, “landscape” or “portrait”;

6.4.5 units
Format: units(s)
Arguments:
   1 s, string, “metres” or “inches”;

6.4.6 papersize
Format: papersize(s)
Arguments:

6.4.7 resolution
Format: resolution(s)
Arguments:
   1 s, integer greater than 1;

Abstract: Public method of an Xfig object which sets the property of the current Xfig object. The methods in this group set the properties that go in the header of the Xfig file (once the file is created).

Public methods that set drawing-related properties of the XFig object

6.4.8 color
Format: idx = color(s, /, rgb/)
Arguments:
   1 s, lower-case string, name of the color;
   2 rgb = [r, g, b], optional 3-dimensional vector, where each entry is in range 0-255. If provided it represents the RGB values of the color which name is given in s.

Result: idx, integer, index of the color.
Abstract: Public method of an Xfig object which creates new color and adds it to the Xfig palette. If the color has already been defined, that is, if the color can be located there by its name then it returns its position.
6.4. world

**Format:** world\((\text{world}\_xy\_0, \text{xfig}\_xy\_0, \text{xfig}\_xy\_dir, xy\_scale/, bbox/)\)

**Arguments:**

1. \(\text{world}\_xy\_0\), vector \([x, y]\), gives the origin of world coordinate system;
2. \(\text{xfig}\_xy\_0\), vector \([x, y]\), gives the position of the world origin on the Xfig plane;
3. \(\text{xfig}\_xy\_dir\), mapping between relative change in world coordinates and in the Xfig plane for each direction. Default value is \([1, -1]\) because the Xfig coordinate system is identical to the image coordinate system where \((0, 0)\) is in the top left corner, and positive directions are to the right for x-coordinate, and down for y-coordinate;
4. \(xy\_scale\), scalar or vector \([s_x, s_y]\), scaling from the world coordinates to the Xfig coordinates;
5. \(bbox\), vector \([w, h]\) describing the width and height of the bounding box using the world coordinates.

**Abstract:** Public method of an Xfig object that provides mapping between the world coordinates and the Xfig coordinates. Bounding box is used if the world data to be plotted on the Xfig plot is to be limited to the box.

6.4.10 label

**Format:** label\((s, xy/, params/)\)

**Arguments:**

1. \(s\), string, text to be put on the Xfig plane;
2. \(xy\), vector \([x, y]\), or matrix \([x_1, y_1; x_2, y_2; \ldots]\) of the coordinates to which the same string \(s\) will be put.
3. \(params\), list, provides details of how the string will be put. If \(params\) contains entry \(\text{world\_coord} = 1\), then it will be assumed that \(xy\) are the world coordinates, and using the world properties these will be then converted to Xfig coordinates.

Strings which world coordinates lie outside the bounding box will be ignored.

**Abstract:** Public method of an Xfig object which puts text at a specified position on the Xfig plane.

6.4.11 ellipse

**Format:** ellipse\((xy, radii, angle/, params/)\)

**Arguments:**

1. \(xy\), vector \([x, y]\), coordinates of the center of the ellipse;
2. \(radii\), scalar or vector \([r_x, r_y]\), the radii of the ellipse (or if scalar, it is then a circle);
3. \(angle\), scalar, value in degrees of the angle between the Xfig x-axis and the x-axis of the ellipse in counter-clockwise direction.
4. \(params\), list, provides details of how the ellipse will be drawn. Its entries are
CHAPTER 6. DATA PLOTTING AND EXPORT

- **line_style**, string, one of 7 styles, “default”, “solid”, “dash”, “dot”, “dash dot”, “dash dot dot”, or “dash dot dot dot”. One should rather use description like “” for default, “_” (underscore) for solid, “-” (minus) for dash, “.” for dot, “-..”, “-..”, and “...”.

- **thickness**, number, in 1/80inch units.

- **pen_color**, string, can be Xfig default colors, or user-defined colors.

- **fill_color**, string.

- **depth**, integer, between 1 and 999, the lower goes on top of the higher.

- **pen_style**, string.

- **area_fill**, if fill_color is provided then this default to 20, meaning that the fill_color is used as is (otherwise 0 to 20 are shades between black and the fill_color, while 21 through 40 are the shades between the fill_color and white.

- **style_val**

- **dir**, integer 0 or 1, direction of the ellipse clockwise, or counterclockwise;

- **start**, real vector [x,y], starting point of the ellipse;

- **end**, real vector [x,y], ending point of the ellipse;

- **world_coord**, integer 0 or 1, whether to use world coordinates rather then Xfig-coordinates for plotting the object.

Public method of an Xfig object which draws an ellipse at a specified position on the Xfig plane.

### 6.4.12 arc

**Format:** arc(xy_c,xy/.,params/)

**Arguments:**

1. **xy_c**, vector [x,y], coordinates of the center of the arc;
2. **xy**, stacked matrix of three vectors [x1,y1;x2,y2;x3,y3], points of the arc;
3. **params**, list, provides details of how the ellipse will be drawn.

Public method of an Xfig object which draws an ellipse at a specified position on the Xfig plane.

### 6.4.13 line

**Format:** line(xy/.,params/)

**Arguments:**

1. **xy**, matrix [x1,y1; x2,y2, ...], of points;
2. **params**, list, provides details of how the line will be drawn.

Public method of an Xfig object which draws a polyline over specified positions on the Xfig plane.
6.4. XFIG

6.4.14 picture

Format: picture(fn, xy,0, xy,d, orient/, params/)

Arguments:

1. *fn*, string, filename of the picture that will be imported into the Xfig plot;

2. *xy_0*, vector [x, y] in Xfig coordinates, where the lower left corner of the picture will go (were the picture imported at the orientation of 0-degrees);

3. *xy_d*, vector [dx, dy], the size of the picture in Xfig coordinates (were the picture imported at the orientation of 0-degrees);

4. *orient*, scalar, 0, 90, 180 or 270, rotation of the picture with respect to its lower left corner (which is fixed point of this rotation).

5. *params*, list, provides details of how the picture frame will be drawn.

Public method of an Xfig object which creates a box into which the external picture file is imported.

6.4.15 export

Format: line(xfig_fn/, format/)

Arguments:

1. *xfig_fn*, string, name to which xfig object will be written using Xfig format;

2. *format*, string, output graphics language with which the utility fig2dev will be called to produce graphics. Some of the supported values are (man fig2dev is your friend): box, cgm, epic, eepic, eepicemu, emf, eps, gbx (Gerber beta driver), gif, ibmgl, jpeg, latex, map (HTML image map), mf (MetaFont), mp (Meta-Post), mmp (Multi-MetaPost), pcx, pdf, pdftex, pdftex_t, pdftex_p, pic, pictex, png, ppm, ps, pstricks, pstricks_p, pstricks, ptricks, ptricks_p, ptk (Perl/tk), shape (LaTeX shaped paragraphs), sld (AutoCad slide format), svg (beta driver), textyl, tiff, tk (tcl/tk), tpic, xbm and xpm.

Public method of an Xfig object which exports the xfig object to a Xfig-formatted file, and possible converts it to other graphics formats.
Example: Consider script that creates a rudimentary Xfig drawing, cf. Fig. (6.13).

```plaintext
fig1 = xfig();
fig1.color("user32", [12, 24, 48]);
fig1.label("Graffiti #2", [2, 5], <<font_flag="post";font="courier";font_size=36;... font_color=xfig_colors().name[1+floor(31*uniform())]>>);
fig1.ellipse([2, 2], [1, 3], 60, <<thickness=3;pen_color=xfig_colors().name[1+floor(31*uniform())]>>);
fig1.ellipse([5, 1], 2, <<thickness=12>>);
fig1.line([1.2, 5.5; 1.4, 6], <<thickness=6>>);
fig1.label("1", [1.2, 2.5; 1.4, 3], <<font_flag="post";font="zapf dingbats";font_size=6>>);
fig1.box([2, 2; 4, 4], <<pen_color="gold";fill_color="white">>);
// specify lower left corner of the picture and its size,
// then rotate it by in 90deg increments:
fig1.picture("./randomnoise.jpg", [0, 8.5], [3, 2], 0);
fig1.picture("./randomnoise.jpg", [0, 8.5], [3, 2], 90);
fig1.picture("./randomnoise.jpg", [0, 8.5], [3, 2], 180);
fig1.picture("./randomnoise.jpg", [0, 8.5], [3, 2], 270);
fig1.line([8, 0; 7, 2], <<forward_arrow=xfig_arrow("tri","fil",4,100,180);thickness=4>>);
// draw flying rock
theta = 45; // deg
t = [0:3.5:1/16];
v_0 = 10; // m/s
x = v_0 * cos(theta * deg) .* t;
y = -0.5 * mks.g * t.^2 + v_0 * sin(theta * deg) .* t;
pos_xy = [x, y];
fig1.world([0,-30], [5,7], [1,1], 0.1, [20,60]);
fig1.line(pos_xy, <<thickness=6;world_coord=1;forward_arrow=xfig_arrow("tri","fil",4,100,180)>>);
fig1.draw_scale([8,7.5], 1, 10, <<thickness=3;font_size=36;unit_name="m">>);
// do the dump
fig1.export("xfig1.fig","pdf");
```
Figure 6.13: Xfig file created by the script from the previous page has been imported in the Xfig, then exported as an jpeg file to produce this image.
Example: Consider script that creates sequence of 6 overlapping circles Escher-style, cf. Fig. (6.14).

```plaintext
N = 6;
escher_colors = ["yellow", "orange", "red", "violet", "blue", "green"]; //
// fig1 = xfig();
fig1.color("orange",[255,165,0]); // source: http://www.rapidtables.com
fig1.color("violet",[238,130,238]);

// plot N circles starting from
c_0 = [5,3.5];
R = 1.25;
r = 1;
fi = 90 * (uniform() - 0.5) + 270 - [0:(N-1)]'./N * 360;
center = c_0 + R*[cos(fi*deg), sin(fi*deg)];

for (i in 1:(N-1))
{
    fig1.ellipse(center[i],r,<<thickness=3;pen_color=escher_colors[i];... fill_color=escher_colors[i];depth=53-i>>);
}

// two arcs (semi-circles) comprise N-th circle - each has its own depth
// arc 1: toward blue
a1_1 = center[N] + r *[cos(fi[N]*deg), sin(fi[N]*deg)];
a1_2 = center[N] + r *[cos((fi[N]-90)*deg), sin((fi[N]-90)*deg)];
a1_3 = center[N] + r *[cos((fi[N]-180)*deg), sin((fi[N]-180)*deg)];
a1_xy = [a1_1; a1_2; a1_3];
fig1.arc(center[N],a1_xy,<<pen_color=escher_colors[N];fill_color=escher_colors[N];... depth=53-N>>);

// arc 2: toward yellow
a2_1 = center[N] + r *[cos((fi[N]-180)*deg), sin((fi[N]-180)*deg)];
a2_2 = center[N] + r *[cos((fi[N]-270)*deg), sin((fi[N]-270)*deg)];
a2_3 = center[N] + r *[cos((fi[N])*deg), sin((fi[N])*deg)];
a2_xy = [a2_1; a2_2; a2_3];
fig1.arc(center[N],a2_xy,<<pen_color=escher_colors[N];fill_color=escher_colors[N];... depth=53>>);

// do the dump
fig1.comment("rlabplus was here on " + timestamp() + " brought by " + getenv("USER"));
fig1.export("xfig2.fig", "pdf");
```
Figure 6.14: Six overlapping solid circles Escher-style that were created using the script from the previous page, that was first created for Xfig, then converted to “pdf” using fig2dev utility (that comes with Xfig).
6.5 Xmgrace

Xmgrace is an open-source software package for producing publication quality 2-D graphs and other data analysis. The library libxmgrace provides functions for exporting data from rlab into xmgrace. The library is written in script language, and is loaded upon starting rlab.

The functions in the library are organized around the internal structure of the xmgrace, where many datasets enter a single graph, and a page can have more than one graph:

- Functions operating on a page: xmgpaper, xmgfilename,
- Functions operation on a graph: xmgtitle, xmgsubtitle, xmgxylabs, xmgxyscale, xmgxticks, xmgyticks, xmgrange, xmglegend, xmgxgrid, xmgygrid, xmggraphsize,
- Functions operating on a dataset: xmgdataset, xmgstyle,
- Associative and executive functions:
  - xmggraph, with which a dataset is associated with a graph,
  - xmgpage, with which a graph is associated with a page,
  - xmgprint, executive command to print a page.

The logic of the commands follows the same principles as with the pgplot and gnuplot packages: a set of formatting commands is prepares the data, which is following the executive command xmgprint written to a file. The file can later be loaded into xmgrace and further manipulated, and eventually printed on a printer or to a file in a number of different supported formats (eps and png probably being the most often used, former for \TeX\ documents, while the latter for Microsoft© or Open Office© documents and presentations).

Service Functions for the whole xmgrace spreadsheet

6.5.1 xmgcolormap

Format: xmgcolormap(i/,color,rgb/)

Arguments:

1. \( i \), an integer, index of a color;
2. \( \text{color} \), a string, name of the color;
3. \( \text{rgb} = [r, g, b] \), an array that provides RGB code of the color \( \text{color} \).

Abstract: xmgcolormap allows user to create a new color to be used either in further script or within xmgrace once the file created by the script is opened, or to retrieve information about already defined color. If used as xmgcolormap\((i)\) then it retrieves the information about the color \( i \). By default xmgrace has 16 predefined colors, for \( i = 0, \ldots, 15 \). Consider an example, where we retrieve the information about the color “1”.

```
>> xmgcolormap(1)
    color  rgb
>> xmgcolormap(1).color
    black
>> xmgcolormap(1).rgb
    0  0  0
```
If used with all parameters set, than it allows a definition of a new color, as in following example, where we change the shade of “gray”, color number 7 from \( rgb = [220, 220, 220] \) to \([160, 160, 160]\). The entries in \( rgb \) field are the red, green and blue component in the color, and run in the range from 0 to 255.

\[
\text{>> xmgcolormap(7, ‘‘gray’’, [160,160,160])}
1
\text{>>}
\]

Note: to reset to default values, one has to reload libxmgrace. Probably the easiest way to do that is to put

```
rfile libxmgrace
```

before doing any xmgrace related commands.

6.5.2 xmgtext

Format: \textit{xmgtext}(s, loc/, size, options, I/)

Arguments:

1. \textit{s}, a string, the text that is to be put on the paper;
2. \textit{loc} = \([x, y]\), a real vector, coordinates in world (\(I\) is given) or view coordinates (no \(I\)) where the text is to be put;
3. \textit{size}, a scalar, the size of the text in xmgrace units;
4. \textit{options} = \([\text{color}/, \text{rotation}, \text{font}, \text{justification}/]\), a real vector, contains the information about the color, angle of rotation, used font and the justification of the text.
5. \textit{I}, an integer, index of the graph.

Abstract: This function prints a string \textit{stext} at a location \textit{loc} on the page. If graph number \(I\) is given, than it is assumed that the location \textit{loc} is in “world” coordinates (in number range of the data plotted on the graph), otherwise it is assumed that the location is in “view” coordinates, where the paper is located in \([0,1]\) range in each direction. The parameter \textit{size} is the size of the text, while \textit{options} is a matrix with at most four elements: color, rotation (of the text), font, and justification. If less than four parameters are given than the missing values are filled with defaults (\( \text{rot} = 0, \text{font} = 0, \text{justification} = 0 \)).

Note: The data associated with this function are non-permanent, that is, after each \textit{xmgprint} the content of text field is erased.

6.5.3 xmgline

Format: \textit{xmgline}(loc/, width, options, arrowopts, I/)

Arguments:

1. \textit{loc} = \([x1, y1, x2, y2]\), a real vector, endpoints of the line in world (\(I\) is given) or view (no \(I\)) coordinates;
CHAPTER 6. DATA PLOTTING AND EXPORT

2. width, a scalar, the width of the line;

3. options = [color/, linestyle/], a real vector, the color and the style of the line;

4. arrowopts = [arrow, type, length, layout], a real vector, options for attaching the arrows to the line.

5. I, an integer, index of a graph.

Abstract: This function draws a line the endpoints coordinates of which are given in loc on the page. If graph number I is given, than it is assumed that the location loc is in “world” coordinates (in number range of the data plotted on the graph), otherwise it is assumed that the location is in “view” coordinates, where the paper is located in [0,1] range in each direction. The parameter width is the width of the line, while options is a matrix with at most two elements: color and linestyle. If less than two parameters are given than the missing values are filled with defaults (color = 0, linestyle = 1). If one wants to have arrows at the end of the line, then arrowopts = [arrow, type, length, layout] needs to be given, where arrow = 0, 1, 2, 3 gives the locations of the arrow: none, start, end, both ends; type = 0, 1, 2 is for line, filled or opaque arrow head, length is its length, and layout = [r1, r2] are the d/L and I/L form factors of the arrow. Check xmgrace manual or wait till I get to it.

Note: The data associated with this function are non-permanent, that is, after each xmgprint the content of line field is erased.

Functions operating on a page

6.5.4 xmgpaper

Format: xmgpaper(papersize, orientation, P)

Arguments:

1. papersize, a string, admissible values are “letter” or “A4”;

2. orientation, a string, admissible values are “landscape” or “portrait”;

3. P, an integer, index of a page.

Abstract: xmgpaper sets the size and the orientation of a page P.

6.5.5 xmgfilename

Format: xmgfilename(filename, P)

Arguments:

1. filename, a string, name of the file which will contain the xmgrace spreadsheet;

2. P, an integer, index of a page.

Abstract: xmgfilename specifies the name of the file to which a xmgrace the page P will be saved after the executive command xmgprint(P).
Functions operating on a graph

6.5.6  xmgtitle

Format: xmgtitle(title, size, N)

Arguments:
1  title, a string, title of the graph;
2  size, a scalar, size of the font used in printing the title;
3  N, an integer, index of the graph.

Abstract: xmgtitle assigns the “title” to the graph \( N \) of the desired size.

6.5.7  xmgsubtitle

Format: xmgsubtitle(subtitle, size, N)

Arguments:
1  subtitle, a string, subtitle of the graph;
2  size, a scalar, font size used for subtitle;
3  N, an integer, index of the graph.

Abstract: xmgsubtitle assigns the “subtitle” to the graph \( N \) of the desired size.

6.5.8  xmgxlabelas

Format: xmgxlabelas(labels, labelsize, N)

Arguments:
1  labels = [labelx, labely], a string vector, labels for x- and y-axis;
2  size, a scalar, font size used for labels;
3  N, an integer, index of the graph.

Abstract: xmgxlabelas assigns the labels to the axes of the graph \( N \).

6.5.9  xmgxyscale

Format: xmgxyscale(scales, N)

Arguments:
1  scales = [scalex, scaley], where scalex, y = “normal” or “log”, the type of the axis;
2  N, an integer, index of the graph.

Abstract: xmgxyscale sets the normal (linear) or logarithmic scaling for each axis of the graph \( N \). The default value for each scale is “normal”.
6.5.10 xmgxticks, xmgyticks

**Format:** xmgxticks(majmin, size, N), xmgyticks(majmin, size, N)

**Arguments:**

1. `majmin` = `[major, minor]`, a vector of size two, position of the major and minor ticks on a given axis;
2. `size`, a scalar, font size used for tick labels;
3. `N`, an integer, index of the graph.

**Abstract:** `xmgxticks`, `xmgyticks` assign the positions at which major ticks are placed (parameter `major`), the frequency of minor ticks (parameter `minor`, how many minor ticks come between two consecutive major ones), and the font size for the major tick labels. Default value for `majmin` is `majmin = [0, 0]`, that is, no ticks.

6.5.11 xmgxtickspec, xmgytickspec

**Format:** xmgxticks(tic/, label, /N), xmgytics(tick/, label, /N)

**Arguments:**

1. `tic`, real vector, position of the ticks on a given axis;
2. `label`, real or string vector, label associated with particular tick;
2,3. `N`, integer, index of the graph.

**Abstract:** Specifies the special ticks to be put on a graph `N`. The ticks can be numeric only, in which case it suffices to specify their position with `tick` vector, or can be numeric or textual labels, in which case the vector `label` specifies what will be written.

The command is useful for graphs where one of the axes is categorical rather than numerical.

6.5.12 xmgrange

**Format:** xmgrange(xyminmax, N)

**Arguments:**

1. `xyminmax` = `[minx, maxx, miny, maxy]`, a real vector of size four, the range for each of the axis;
2. `N`, an integer, index of the graph.

**Abstract:** `xmgrange` assigns the minimum and maximum value for each of the axes of the graph `N`. If the command is not used in preparation of the page, than the minimum and maximum values extracted from the datasets are used.
6.5.13 \texttt{xmglegend}

\textbf{Format}: \texttt{xmglegend(loc, size, /legopts, /N)}

\textbf{Arguments}:

1. \texttt{loc} = \([\text{locx}, \text{locy}]\), real vector of size two, location of the legend box;
2. \texttt{size}, scalar, font size used for tick labels;
3. \texttt{lopts}, list, additional options for user to fine tune the appearance of the legend box, \texttt{lopts =<< color; length; vgap; hgap >>}, for the color of the legend text (default 1, black), length of the line (default 4), vertical gap between two legend entries (default 2), in an entry the horizontal gap between the line/symbol and the text (default 1).
4. \texttt{N}, integer, index of the graph.

\textbf{Abstract}: \texttt{xmglegend} puts a legend on the graph \texttt{N}, at location \([\text{locx}, \text{locy}]\) in, so called, view coordinates, where the paper size is in range \([0, 1]\) for both coordinate directions. In order to properly place your legend, one has to be aware of the size of the graph, which is by default \([\text{xmin}, \text{xmax}, \text{ymin}, \text{ymax}] = [0, 1, 0, 1]\). \texttt{size} determines the size of the font used in preparing the legend, and the default value is \texttt{legsize} = 1. Check the Table 6.2 for more details on how to write a legend using the \texttt{xmgrace} text formatting commands. By default, the legend is not associated with the graph.

\textbf{Note}: A previously defined legend for a graph is erased by sending an empty \texttt{xmglegend} command, i.e., \texttt{xmglegend(., N)}.

6.5.14 \texttt{xmgxgrid}, \texttt{xmgygrid}

\textbf{Format}: \texttt{xmgxgrid(gridmaj, gridmin, N)}, \texttt{xmgygrid(gridmaj, gridmin, N)}

\textbf{Arguments}:

1. \texttt{gridmaj}, \texttt{gridmin} = \([\text{color}, \text{width}, \text{style}]\), a real vector of size three, information about the type of the line used for plotting the grid;
2. \texttt{N}, an integer, index of the graph.

\textbf{Abstract}: \texttt{xmgxgrid}, \texttt{xmgygrid} setup the grid for the graph \texttt{N}. Parameter \texttt{color} goes in the range 0…15, where 1 corresponds to black, and 7 to gray (I use these two for black and white publication plots). The parameter \texttt{width} determines the width of the line (1 is reasonable for auxiliary lines, 3 is a typically width of a line connecting the data points), and \texttt{style} is an integer in range 0…11 (where 1 is solid line, 2 is short dash, etc.). By default a grid is not defined for either axis.

\textbf{Example}: To have a grid consisting of gray solid for major ticks and short-dashed line for minor ticks use the following,

```
1 // graph #0
2 xmgxgrid( [7,1,1], [7,1,2], 0 );
3 xmgygrid( [7,1,1], [7,1,2], 0 );
```

Chapter 6. Data Plotting and Export

Note: To remove previously defined x-grid from the graph N, use command `xmgxgrid(N)`.

6.5.15 xmggraphsize

Format: `xmggraphsize(size, N)`

Arguments:

1. `size = [x_{left}, x_{right}, y_{bottom}, y_{top}]`, a real vector of size four, the corners of the graph in world coordinates;

2. `N`, an integer, index of the graph.

Abstract: `xmggraphsize` allows user to set the position of the graph `N` in the world coordinates `x, y ∈ [0, 1]`, so that x-position of the graph takes `[x_{left}, x_{right}]` in the x-direction and `[y_{bottom}, y_{top}]` in the y-direction.

Example: Setting two graphs on top of each other, say `graph 1` on top of `graph 0`, where both graphs have the same size (with some vertical space between the two), is done as follows,

```plaintext
1. xmggraphsize([0,1,0,0.45], 0); // put graph 0 on the bottom of the page
2. xmggraphsize([0,1,0.55,1], 1); // put graph 1 on top of the page
3. ...
4. xmpage([0,1], 0); // associate graphs 0,1 to page 0
5. xmpprint( 0 ); // print page 0 using info above for each graph
```

Functions operating on a dataset

6.5.16 xmgdataset

Format: `xmgdataset(data, plottype, name, I)`

Arguments:

1. `data`, a real matrix, dataset to be plotted;

2. `plottype`, a string, the type of the plot;

3. `I`, an integer, index of the dataset.

Abstract: `xmgdataset` associates with dataset `I`, a number matrix of actual data to be plotted column-wise, the format of which is given by `plottype`. Here, `plottype` can be any of supported data formats, shown in Table 6.1, together with the number of columns they have to have. The function makes the match between the number of columns, as prescribed by `plottype`, to that found in `dataset`, and complains if they don’t match. The parameter “name” is the name under which the dataset will appear if the legend is plotted, as well.

6.5.17 xmgstyle

Format: `xmgstyle(ls, style, I)`

Arguments:
Table 6.1: Types of plots and the number of columns in the respective dataset supported in libxmgrace toolkit.

<table>
<thead>
<tr>
<th>plottype</th>
<th>No. of columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;xy&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;xydx&quot;</td>
<td>3</td>
</tr>
<tr>
<td>&quot;xydy&quot;</td>
<td>3</td>
</tr>
<tr>
<td>&quot;xydxdx&quot;</td>
<td>4</td>
</tr>
<tr>
<td>&quot;xydydy&quot;</td>
<td>4</td>
</tr>
<tr>
<td>&quot;xydxdy&quot;</td>
<td>4</td>
</tr>
<tr>
<td>&quot;xydxdxdydy&quot;</td>
<td>6</td>
</tr>
<tr>
<td>&quot;bar&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;bardy&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;bardydy&quot;</td>
<td>3</td>
</tr>
<tr>
<td>&quot;xyhilo&quot;</td>
<td>3</td>
</tr>
</tbody>
</table>

1. `ls`, a string, admissible values are “line”, “symbol” or “both”;

2. `style = [shape, color, linewidth]` for `ls = “line”`,
   `style = [shape, color, size, linewidth]` for `ls = “symbol”`,
   `style = [shape, color, linewidth, 0; shape, color, size, linewidth]` for `ls = “both”`;

3. `I`, an integer, index of the dataset.

Abstract: `xmgstyle` determines how the dataset `I` will be presented on a graph. The parameter `shape` determines, in the case of “line”, what type of line will be used: 0, for none, 1 for solid, 2 for short dash, etc. The parameter `color` is as before, 0 for white, 1 for black, .., 7 for gray, etc. The parameter `linewidth` should be between 2 and 3 for publication-quality plots (small graphs, need thick lines). Symbols have two parameters, one is the size of the symbol (typically less than unity), the other is the linewidth of the line used in drawing the symbol (typically between one and two).

Note: When using “both” the parameters for the line and symbol are both passed, where line parameters are stacked on top of the symbol parameters.

Assignment and executive functions

6.5.18 `xmggraph`

Format: `xmggraph(datasets, N)`

Arguments:

1. `datasets = [I_1, ...]`, a real vector, indices of datasets to be plotted;

2. `N`, an integer, index of the graph.

Abstract: `xmggraph` associates `datasets` with graph `N`. 
6.5.19  **xmgpage**

**Format:**  `xmgpage(graphs, P)`

**Arguments:**
1. `graphs = [N1, ...]`, a real vector, indices of graphs to be plotted;
2. `P`, an integer, index of the page.

**Abstract:**  `xmgpage` associates `graphs` with the page `P`.

6.5.20  **xmgprint**

I  **Format:**  `xmgprint(P)`

**Arguments:**
1. `P`, an integer, index of the page.

**Abstract:**  `xmgprint` prints the page `P` to the file the name of which is previously set by `xmgfilename`.

II  **Format:**  `xmgprint(X)`

**Arguments:**
1. `X`, a real matrix, `X = [x, y_1, y_2, ...]`, or a list, `X =<< 1 = [x_1, y_1]; 2 = [x_2, y_2]; ... >>`, of two-column real matrices.

**Abstract:**  A quick and dirty version of the executive command of the `libxmgrace` toolkit. It creates a file that contains a graph number 0, and two-column datasets contained in `X`. 
Example: Producing an xmgrace compatible file from within rlab. Short script that produces Fig. (6.15).

```plaintext
rfile libxmgrace
format(9,9);
c = sqrt(2/pi);

// range of g
gmin = -0.670513; // this value was found by mathematica, see variational/poly.nb
gmax = 0;
ng = 200;

// book keeping
x = [];

// do the thing
for (i in 0:ng)
{
g = gmin + (gmax-gmin)/ng*i;
p = [1,0,0,0,-1,-c*g];
r1 = polyroots( p ).roots;
r2 = sort( real(r1) ).val;
x = [x; [g, r2[4,5];]];
}

// xmg: paper P=0
xmgfilename("variation.gr",0);
xmpaper ("letter", "landscape", 0);

// xmg: graph N=0
xmtitle ("Variational Calculation of the BEC size", 1.3, 0);
xmgxlabel( [ "parameter \qg\", "size of the condensate \qs\"] , 1.3, 0);
xmgrange ( [-1,0,0,1.1],0);
xmgxticks ( [ 0.2, 1 ] , 1.3, 0);
xmgyticks ( [ 0.2, 1 ] , 1.3, 0);
xmglegend ( [0.85, 0.5], 1, 0);

// xmg: datasets I = 0,1
xmgdataset( x[:,1,2], "xy", "local minimum", 0);
xmgs( "line", [1,1,3], 0);
xmgdataset( x[:,1,3], "xy", "local maximum", 1);
xmgs( "line", [5,1,3], 1);

// xmg: print
xmggraph ( [0,1], 0);
xmpage ( 0, 0 );
xmpprint ( 0 );
```
Figure 6.15: An example of figure created using the library \textit{libxmgrace}. The created file “variation.gr” was opened in \textit{xmgrace} and from there printed as an \textit{eps} file for this manual.
### 6.5. XMGRACE

Table 6.2: Some basic typesetting codes for `xmgrace` when used from the `xmgrace.r` or `xmgrace.r` toolkits

<table>
<thead>
<tr>
<th>Control code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\f{x}</td>
<td>switch to font named &quot;x&quot;</td>
</tr>
<tr>
<td>\f{n}</td>
<td>switch to font number n</td>
</tr>
<tr>
<td>\f{}</td>
<td>return to original font</td>
</tr>
<tr>
<td>\R{x}</td>
<td>switch to color named &quot;x&quot;</td>
</tr>
<tr>
<td>\R{n}</td>
<td>switch to color number n</td>
</tr>
<tr>
<td>\R{}</td>
<td>return to original color</td>
</tr>
<tr>
<td>#{x}</td>
<td>treat &quot;x&quot; (must be of even length) as list of hexadecimal char codes</td>
</tr>
<tr>
<td>\t{xx xy yx yy}</td>
<td>apply transformation matrix</td>
</tr>
<tr>
<td>\t{}</td>
<td>reset transformation matrix</td>
</tr>
<tr>
<td>\z{x}</td>
<td>zoom x times</td>
</tr>
<tr>
<td>\z{}</td>
<td>return to original zoom</td>
</tr>
<tr>
<td>\r{x}</td>
<td>rotate by x degrees</td>
</tr>
<tr>
<td>\l{x}</td>
<td>slant by factor x</td>
</tr>
<tr>
<td>\v{x}</td>
<td>shift vertically by x</td>
</tr>
<tr>
<td>\v{}</td>
<td>return to unshifted baseline</td>
</tr>
<tr>
<td>\W{x}</td>
<td>shift baseline by x</td>
</tr>
<tr>
<td>\W{}</td>
<td>reset baseline</td>
</tr>
<tr>
<td>\h{x}</td>
<td>horizontal shift by x</td>
</tr>
<tr>
<td>\n</td>
<td>new line</td>
</tr>
<tr>
<td>\u</td>
<td>begin underline</td>
</tr>
<tr>
<td>\U</td>
<td>stop underline</td>
</tr>
<tr>
<td>\o</td>
<td>begin overline</td>
</tr>
<tr>
<td>\O</td>
<td>stop overline</td>
</tr>
<tr>
<td>\Fk</td>
<td>enable kerning</td>
</tr>
<tr>
<td>\FK</td>
<td>disable kerning</td>
</tr>
<tr>
<td>\Fl</td>
<td>enable ligatures</td>
</tr>
<tr>
<td>\FL</td>
<td>disable ligatures</td>
</tr>
<tr>
<td>\m{n}</td>
<td>mark current position as n</td>
</tr>
<tr>
<td>\M{n}</td>
<td>return to saved position n</td>
</tr>
<tr>
<td>\dL</td>
<td>LtoR substring direction</td>
</tr>
<tr>
<td>\dr</td>
<td>RtoL substring direction</td>
</tr>
<tr>
<td>\dL</td>
<td>LtoR text advancing</td>
</tr>
<tr>
<td>\dR</td>
<td>RtoL text advancing</td>
</tr>
<tr>
<td>\x</td>
<td>switch to Symbol font (same as \f{Symbol})</td>
</tr>
<tr>
<td>+</td>
<td>increase size (same as \z{1.19} ; 1.19 = sqrt(sqrt(2)))</td>
</tr>
<tr>
<td>-</td>
<td>decrease size (same as \z{0.84} ; 0.84 = 1/sqrt(sqrt(2)))</td>
</tr>
<tr>
<td>\s</td>
<td>begin subscripting (same as \v{-0.4}\z{0.71})</td>
</tr>
<tr>
<td>\S</td>
<td>begin superscripting (same as \v{0.6}\z{0.71})</td>
</tr>
<tr>
<td>\t{xx xy yx yy}</td>
<td>same as \t{xx xy yx yy}</td>
</tr>
<tr>
<td>\z{x}</td>
<td>absolute zoom x times (same as \z{}z{x})</td>
</tr>
<tr>
<td>\q</td>
<td>make font oblique (same as \l{0.25})</td>
</tr>
<tr>
<td>\Q</td>
<td>undo oblique (same as \l{-0.25})</td>
</tr>
<tr>
<td>\N</td>
<td>return to normal style (same as \v{}\t{})</td>
</tr>
<tr>
<td>\print \</td>
<td>print \</td>
</tr>
<tr>
<td>\n</td>
<td>switch to font number n (0-9) (deprecated)</td>
</tr>
<tr>
<td>\c</td>
<td>begin using upper 128 characters of set (deprecated)</td>
</tr>
<tr>
<td>\C</td>
<td>stop using upper 128 characters of set (deprecated)</td>
</tr>
</tbody>
</table>
6.6 OpenOffice.org: toolbox/libooo.r

The library libooo.r provides functions for exporting the rlab data to an OpenOffice.org compatible spreadsheet (.sxc) or word (.sxw) document. It is a transliteration of a perl package ooolib-0.1.5 by [Colton 2004], written in scripting language. The library is located in toolbox subdirectory of the rlab source tree.

The library is limited to operating on one Open/Libre Office document at a time. It implements the following steps. First, one initializes the library. Then proceeds with initialization of the document variables, and setting them to appropriate values. In the end, the information is written to a file.

Initialization/resetting of the library is done by loading it in the memory,

```r
>> rfile libooo
```

For the remaining operations, a list of functions `ooo` is available.

6.6.1 ooo

1. Format: `ooo.init(type/,opt/)`
   
   Arguments:
   
   1. `type`, string, allowed values “sxc” for spreadsheet, or “sxw” for text document;
   2. `opt`, string, allowed value is “debug” for functions to write to the terminal what they should be doing;

   Abstract: Initializes the values of the internal rlab variables relevant for a creation of an OpenOffice.org document.

2. Format: `ooo.set(name/,val1//,val2/)`
   
   Arguments:
   
   1. `name`, string, refers to the option in the OpenOffice.org document that is being set;
   2. `val1`, `val2`, string or scalar, a value assigned to the option `name`;

   Abstract: sets the variables used in creating and processing documents. The following is the list of allowed values for their `name` and the values they require.

   - “title”, requires `val1` string, default value “rlabplus” ;
   - “author”, requires `val1` string, default value “Anonymous” ;
   - “subject”, requires `val1` string, default value “No particular subject” ;
   - “comments”, requires `val1` string, default value “OOO file generated by libooo/rlabplus”
   - “builddir”, requires `val1` string, default value “/ooo.tmp” in the current path;
   - “keyword”, requires `val1` string, a single word that is to be added to the keywords for the document.
   This may be used multiple times to add multiple keywords to the document.
   - “meta[1-4]-name”, requires `val1` string, containing the title of a user defined variable.
   - “meta[1-4]-value”, requires `val1` string, the value of a user defined variable.
6.6. OpenOffice.org: Toolbox/libooo.r

- “cell-loc”, requires val1 string, containing x and y coordinates in a spreadsheet in text format, say “A1” or such, or val1 and val2, both scalars, containing x and y, or just val1 as a two component vector [x, y]. It sets the coordinates of a cell to which the next oooData command will print.
- “cell-left”, moves the writing cell one to the left.
- “cell-right”, moves the writing cell one to the right.
- “cell-up”, moves the writing cell up one.
- “cell-down”, moves the writing cell down one.
- “cell-auto”, requires val1 and val2, which are automatic increments of the cell location following oooData.
- “justify”, requires val1 string to be “right”, “left”, “center”, or “block”. The next text will then be justified accordingly.
- “bold”, requires val1 string to be “on” or “off”.
- “italic”, requires val1 string to be “on” or “off”.
- “underline”, requires val1 string to be “on” or “off”.
- “text-color”, requires val1 string to be six hex digits in the form RRGGBB where RR is the red, GG is the green, and BB is the blue for the color of the text. You may also select the following options “red”, “green”, “blue”, “black”, “white”, and “default”.
- “text-bgcolor”, requires val1 string same as “text-color” except it is instead applied to the background or highlighting of the text.
- “text-size”, requires val1 string that must be a number for the size of the font. The following text will be written in this size. The exception is the headings.

3. Format: ooo.data(style, text)

Arguments:
1. style, string;
2. val, string or scalar;

Abstract: The function enters the data into the OpenOffice.org document. The following is the list of allowed values for style and a value it requires:

- “h”, val1 contains the heading of the document;
- “h[1-9]”, for headings level 1-9, val1 contains the heading of the document;
- “default”, sets style to “standard” or “default”, val1 contains the text of the document;
- “cell-text”, val1 contains a text;
- “cell-float”, val1 contains a number;
- “cell-formula”, val1 contains a textual representation of a formula;
– “cell-skip”, just runs the cell-auto increment without writing anything to the cell.

3. **Format**: \( id = \text{ooo.special}(\text{style/}, val1, val2/) \)

   **Arguments**:

   1,2 \textit{style}, string, allowed values are “pagebreak” or “list-ordered”, the latter requires \textit{val1} to be the first list item. In subsequent calls use \textit{val1} to pass the value \textit{id} returned from the first call, and \textit{val2} as the appropriate list entry.

   **Abstract**: Creates special types of data for documents.

4. **Format**: \( \text{ooo.generate}(filename) \)

   **Arguments**:

   – \textit{filename}, string, name used to write a generated document to the file system
Example: Create a sxc calc file with 25 cells and a formula.

```plaintext
//
// file: eg_calc1.r
//
// based on calc-example1 from perl package ooolib:
// This program creates a SXC Calc file with 25 cells and a formula.
// by Joseph Colton

rfile libooo

Set variables
ooo.init("sxc");
ooo.set ("builddir", ".\tmp");  // Directory to create document in
ooo.set ("title", "Calc 25 Cells");  // Title of document
ooo.set ("subject", "Calc Example");  // Subject of document
ooo.set ("comments", "Who needs comments?");  // Comments for document
ooo.set ("keyword", "keyword1");  // Add a keyword for the document
ooo.set ("keyword", "keyword2");  // Add a keyword for the document
ooo.set ("author", "libooo example");  // Set the author of the document

// User defined variables
ooo.set ("meta1-name", "Programmers");  // 1-4 user defined variables
ooo.set ("meta1-value", "Joseph Colton and Marijan Kostrun");  // 1-4 user defined variables

// Write the spreadsheet data
// It is in x, y cords.
for (x in 1:5)
{
  for(y in 1:5)
  {
    ooo.set ("cell-loc", x, y);
    ooo.data ("cell-float", x*y);
  }
}

ooo.set ("cell-loc", 1, 6);
ooo.data("cell-formula", "=SUM(B2:C2)");

ooo.generate("calc-example1_rlab.sxc");  // Create the document
```
Part III

Builtin-type functions
Chapter 7

General Purpose Functions

7.1 Miscellaneous mathematical and other functions

7.1.1 iseed

Format: \( i = \text{iseed}() \),

Result: \( i \), integer.

Abstract: Reads an unsigned integer from the local source of entropy, “/dev/urandom”. It is intended for use in conjunction with \textit{irand()} function to initialize random number generator.

7.1.2 hash

Format: \( h_\text{s} = \text{hash}(f_\text{n}, s/, nl/) \),

Arguments:

1. \( f_\text{n} \), string, name of the hash function used. Available are “md2”, “md5”, “sha”, “sha1”, “dss”, “dss1” and “ripemd160” hash function, all from the SSL library;
2. \( s \), string matrix;
3. \( nl \), string, sequence of characters representing a new line (typically “\n” for Linux or “\r\n” for MS windows).

Result: \( h_\text{s} \), string of length of respective hash function.

Abstract: Finds the hash of a given string matrix where the columns of the matrix are concatenated row-wise while between the rows a newline string is inserted.

7.1.3 crc32

Format: \( h_\text{s} = \text{crc32}(s/, nl, is/) \),

Arguments:

1. \( s \), string matrix;
2 *nl*, string, sequence of characters representing a new line (typically “\n” for linux, or “\r\n” for MS windows).

3. *is*, integer, seed for the crc-32 calculator. The default value is '0xFFFFFFFF'.

Result: *hs*, crc-32 hash of the string matrix *s*.

### 7.1.4 showkey

**Format:** \( i = \text{showkey}(/\text{dwait}/) \),

**Arguments:**

1. *dwait*, real scalar, duration of time the function will wait for user to press a key on keyboard. If 0 or not given then the function seats there until a key is pressed.

Result: *i*, integer scalar, ASCII code of the key being pressed, or 0 if none was pressed and the function timed out.

### 7.1.5 floor, ceil, round

**Format:** \( a = \text{floor}(x/, \text{opts}/) \),

**Arguments:**

1. *x*, real or complex matrix;
2. *opts*, optional list with entries *bin*, *offset*, *base*, *clamp*, *mesh*, where

\[
a = \text{offset} + \text{bin} \times f((x - \text{offset})/\text{bin}), \tag{7.1}
\]

with \( f = \text{ceil}, \text{floor} \) the standard math functions. In the absence of *opts*, the values *offset*=0 and *bin*=1 are used. In conjunction with these two options, one can also use *clamp* = [min_val, max_val], which clamps the output to the specified interval.

If instead *base* > 0 is provided, then the above formula changes to

\[
a = \text{base}^{\lceil \log(x)/\log(\text{base}) \rceil}, \tag{7.2}
\]

that is, the function finds the closest power of *base* that brackets the argument from below (floor) or above (ceil).

With *mesh* = \([x_1, x_2, \ldots, x_N]\) provided, for each entry in *a* the interval \([x_i, x_{i+1}]\) that brackets it is found, and then one of the bounds is chosen, \(x_i\) for *floor*, or \(x_{i+1}\) for *ceil*. If entry in *a* is outside the mesh then value NaN is returned.

Result: *a*, real or complex matrix.

### 7.1.6 ifelse

**Format:** \( a = \text{ifelse}(\text{cond, iftrue, iffalse}) \),

**Arguments:**

1. *cond*, a dense matrix of 0’s or 1’s, representing the condition;
2.3 *iftrue*, *iffalse*, two dense matrices of the same type (double, complex or string).

**Result:** a, a matrix of the same type as *iftrue* and *iffalse*.

**Abstract:** *ifelse*, also known as a spreadsheet “If”, is a function which in the output combines entries from *iftrue* and *iffalse* depending on the value of *cond*. Its exact definition is

\[
\text{ifelse}(\text{cond}, \text{iftrue}, \text{iffalse})[i; j] = \begin{cases} 
\text{iftrue}[i; j] & \text{for } \text{cond}[i; j] == 1, \\
\text{iffalse}[i; j] & \text{for } \text{cond}[i; j] == 0. 
\end{cases}
\] (7.3)

Here, for a given matrix A of size \(nr\)-by-\(nc\), barred indices are given as follows

\[
\bar{i} = \min(i, nr), \\
\bar{j} = \min(j, nc). 
\] (7.4)

This flexible definition allows a replacement array to have a single element, be a row or column vector. The only requirement *ifelse* has on its arguments that either both arrays *iftrue* and *iffalse* are either both numeric (real or complex) or both strings.

**7.1.7 swap**

**Format:** \(\text{swap}(a, b)\),

**Arguments:**

1. \(a\), rlab variable;
2. \(b\), rlab variable;

**Result:** The function swaps content of two variables.
7.1.8  flipud, fliplr

Format: flipud(x),
Arguments:
  1  x, dense matrix;
Result: Matrix x is manipulated in place; row of matrix are flipped up-down, or columns left-right.

7.1.9  shiftl, shiftr, shiftd, shiftu

Format: shiftu(x/,n/),
Arguments:
  1  x, dense matrix;
  2  n, positive integer scalar (if omitted default value n = 1 is used);
Result: Matrix x is manipulated in place; it is shifted up or down by n rows, or the columns are shifted left or right by n columns.

7.1.10  rot90

Format: rot90(x/,n/),
Arguments:
  1  x, dense matrix;
  2  n, positive integer scalar (if omitted default value n = 1 is used);
Result: Matrix x is manipulated in place; the matrix is rotated by 90° in counter-clockwise direction n-times.
Polynomial toolkit

The rlab uses the same way of a representing a polynomial as m*lab, that is, a polynomial is represented by a row-vector:

\[ P_n(x) \equiv [a_1, a_2, \ldots, a_{n+1}] \equiv \sum_{i=1}^{n+1} a_i x^{n+1-i}. \]  

(7.5)

7.1.11 polyval

**Format:** \( y = polyval(x, a) \),

**Arguments:**

1. \( x \), matrix;
2. \( a \), row-vector, coefficients of a polynomial, see Eq. (7.5).

**Result:** \( y \), matrix of size \( x \), containing the values of the polynomial for the respective entries from matrix \( x \).

7.1.12 polyder

**Format:** \( b = polyder(a, n) \),

**Arguments:**

1. \( a \), row-vector, real or complex.
2. \( n \), non-negative integer. If omitted \( n = 1 \) is assumed.

**Result:** \( b \), row vector, real or complex.

**Abstract:** Finds the \( n \)-th derivative of the polynomial \( a \).

7.1.13 polyint

**Format:** \( b = polyint(a) \),

**Arguments:**

1. \( a \), row-vector.

**Result:** \( b \), row-vector.

**Abstract:** Finds the integral of a polynomial \( a \) with respect to the argument \( x \). Newly emerged coefficient in \( b \) is set to 0.

7.1.14 conv

**Format:** \( c = conv(a, b) \),

**Arguments:**

1. \( a \), row-vector.
Result: \( c \), row-vector.

Abstract: Finds a convolution (product) of two polynomials \( a \) and \( b \). It removes any leading zeros before returning the resulting polynomial.

### 7.1.15 deconv

**Format:** \( c = \text{deconv}(a, b) \),

**Arguments:**

1. \( a \), row vector.
2. \( b \), row vector.

**Result:** \( c = << \text{quotient}; \text{residue} >> \), a list containing the quotient \( q \) and the residue \( r \) of deconvolution, where \( a = \text{conv}(b, q) + r \).

### 7.1.16 polyroots

**Format:** \( y = \text{polyroots}(a/\text{options}/) \),

**Arguments:**

1. \( a \), a dense row vector.
2. \( \text{options} \), list, its entries are:
   - \( \text{eps} \), positive scalar, desired precision of the root;
   - \( \text{dmin}, \text{dmax} \), positive scalar, minimal and maximal real number;
   - \( \text{maxit} \), integer, maximal number of iterations to reach the precision \( \text{eps} \).

**Result:** \( y = << \text{roots}; \text{radius}; \text{success} >> \), a list containing the roots of the polynomial and their numerical properties, where \( \text{roots} \) is a complex row-vector containing the roots of the polynomial, \( \text{radius} \) is a real row-vector containing the absolute error of a particular root. The array \( \text{success} \) contains information whether each of the root calculations was successful (1), or not (0).

### 7.1.17 poly

**Format:** \( a = \text{poly}(r) \),

**Arguments:**

1. \( r \), column-vector, roots of a polynomial \( a \).

**Result:** \( a \), row-vector representing the polynomial \( a \) in standard form.
7.1. MISCELLANEOUS MATHEMATICAL AND OTHER FUNCTIONS

7.1.18 nseries

Format: \( a = \text{nseries}(f, /p/, x, n, /options/) \),

Arguments:

1,2 \( f = \text{function}(x, /p/) \) and its parameter entity \( p \);

3 \( x \), scalar, abscissa at which the numerical series expansion of the function \( f \) is to be calculated;

4 \( n \), integer, order of expansion;

5 \( \text{options} = << \text{eps}; r >> \), list, containing the numerical error in the computation of coefficients and the radius of convergence of the requested series expansion.

Result: \( a \), row-vector, polynomial expansion of degree \( n \) of function \( f \) at \( x \).

Note: \( \text{nseries} \) does poorly on a series expansion of polynomials.
Example: *ifelse* function in rlab2.

```
>> x = [0,1,0]
   0 1 0
>> a = ["a", "a", "a"]
a a a
>> b = ["b", "b","b"]
b b b
>> ifelse(x,a,b)
b a b
>> ifelse(x,[a;a],[b;b])
b a b
b a b
>> ifelse(x,[a;b],[b;a])
b a b
a b a
>>
>> x=[0;1]
   0
   1
>> ifelse(x,a,b)
b b b
a a a
>>
```
Example: Polynomial toolkit functions in action.

```matlab
>> a=[1,2,1]
     1 2 1
>> polyint(a)
     0.333 1 1 0
>> polyder(a)
     2 2
>> x=[1;2;3]
     1
     2
     3
>> polyval(x,a)
     4
     9
     16
>> conv(a,a)
     1 4 6 4 1
>> a=[1,2,1]
     1 2 1
>> a = conv(a,a)
     1 4 6 4 1
>> b = [1,3,1]
     1 3 1
>> c = deconv(a,b)
     quotient    residue
>> c.quotient
     1 1 2
>> c.residue
     -3 -1
>>
```
Example: Root finding and subsequent test of the roots.

```matlab
>> a
1 4 6 4 1
>> x=polyroots(a)
-1 + 0i
-1 - 0.000168i
-1 + 0.000168i
-1 + 0i
>> imag(x)==0
1
0
0
1
>> polyval(x,a)
3.33066907e-16 + 0i
7.77156117e-16 + 3.8967486e-19i
7.77156117e-16 - 3.8967486e-19i
5.55111512e-16 + 0i
```
Exponential functions

7.1.19 mexp

Format:

I \( y = mexp(A/,.opts/) \),
II \( y = mexp(A,t/,opts/) \),
III \( y = mexp(A,t,w/,opts/) \).

Arguments:

1 \( A \), matrix, dense square.
2 \( t \), scalar, \( t > 0 \).
3 \( w \), vector.
4 \( opts \), list of options for the solver. Its entries are
   – \( padec\_degree \), integer, used for cases I and II, degree of Padé approximation to \( \exp(At) \). Default value 6.
   – \( krylov\_dim \), integer, used in III. If \( krylov\_dim = -1 \) then Chebyshev method is used (default), else for \( 0 < krylov\_dim < \dim A \) the krylov projection method is used.

Result: \( y \), matrix, (if \( t \) is not given then \( t \equiv 1 \) is assumed),
\[ y = \exp(At), \] (7.6)

or vector, where
\[ y = \exp(At)w. \] (7.7)


7.1.20 mpow

Format: \( y = mpow(A,j) \),

Arguments:

1 \( A \), matrix, dense square.
2 \( j \), integer, \( j \geq 0 \).

Result: \( y \), matrix, dense square, where
\[ y = A^j. \] (7.8)

Abstract: Finds matrix exponentiation by an integer. Uses some tricks so that total number of matrix multiplications is \( \leq 2 \cdot \log_2(j) \), while storing internally two matrices of the same size as \( A \). The code is illustrated in script test/mdi/eg_mpow1.r.
7.1.21 \texttt{cosh}

\textbf{Format:} \( y = \cosh(x) \)

\textbf{Arguments:}

1. \( x \), a dense numeric matrix.

\textbf{Result:} \( y \), a dense numeric matrix of the size of \( x \), where

\[
 y[i; j] = \frac{1}{2} (\exp(x[i; j]) + \exp(-x[i; j])).
\]  

\hspace{0.5cm} (7.9)

7.1.22 \texttt{sinh}

\textbf{Format:} \( y = \sinh(data) \)

\textbf{Arguments:}

1. \( x \), a dense numeric matrix.

\textbf{Result:} \( y \), a dense numeric matrix of the size of \( x \), where

\[
 y = \frac{1}{2} (\exp(x) - \exp(-x)) .
\]  

\hspace{0.5cm} (7.10)

7.1.23 \texttt{tanh}

\textbf{Format:} \( y = \tanh(data) \)

\textbf{Arguments:}

1. \( x \), a dense numeric matrix.

\textbf{Result:} \( y \), a dense numeric matrix of the size of \( x \), where

\[
 y = \begin{pmatrix}
 \exp(x) - \exp(-x) \\
 \exp(x) + \exp(x)
\end{pmatrix} .
\]  

\hspace{0.5cm} (7.11)

7.1.24 \texttt{acosh}

\textbf{Format:} \( y = \text{acosh}(data) \)

\textbf{Arguments:}

1. \( x \), a dense numeric matrix.

\textbf{Result:} \( y \), a dense numeric matrix of the size of \( x \), where

\[
 y = \ln \left( x + \sqrt{x^2 - 1} \right) .
\]  

\hspace{0.5cm} (7.12)
7.1. MISCELANEOUS MATHEMATICAL AND OTHER FUNCTIONS

7.1.25 asinh

Format: $y = \text{asinh}(data)$

Arguments:

1. $x$, a dense numeric matrix.

Result: $y$, a dense numeric matrix of the size of $x$, where

$$ y = \ln \left( x + \sqrt{x^2 + 1} \right). $$

(7.13)

7.1.26 atanh

Format: $y = \text{atanh}(data)$

Arguments: $MDR \times data$ or $MDC \times data$,

1. $x$, a dense numeric matrix.

Result: $y$, a dense numeric matrix of the size of $x$, where

$$ y = \frac{1}{2} \ln \left( \frac{1 + x}{1 - x} \right). $$

(7.14)
7.2 Miscellaneous byte-wise functions

7.2.1 bytejoin

**Format:** \( y = \text{bytejoin}(\text{data, is\_big\_end, stride}) \)

**Arguments:**

1. \( \text{data} \), array double or int, of byte values, that is, in range 0..255;
2. \( \text{is\_big\_end} \), 0 or 1, is the data in big endian or little endian (default) order;
3. \( \text{stride} \), integer or string. If integer, then this is how many bytes are combined into single datum. If string, than the following values are accepted “float”, “float32”, “single”, “real*4” for 4-byte floating point number; “double”, “float64”, “real*8” for 8-byte double precision number; “int”, “int32\_t”, “int32”, “integer*4” for 4-byte integer; “unsigned int”, “uint32\_t”, “uint32” for 4-byte unsigned integer; “unsigned short int”, “uint16\_t”, “uint16” for 2-byte unsigned integer; “short int”, “int16\_t”, “int16” for 2-byte signed integer. Based on the type, stride is computed and the signedness: e.g., 2-byte signed integer -1 (=0xffff) will be converted to 4-byte signed integer -1 (=0xffffffff), rather than to 65535 (=0x0000ffff).

**Result:** \( y \), array of numerical values converted from their byte-wise storage.

**Abstract:** Converts stream of bytes to their numerical value (signed and unsigned integer, float or double). Useful when numerical data is transferred byte-wise rather than as strings.
7.3 String functions

Since version 2.4.2.2, rlab string functions are fully integrated with the software library Pattern Matching Ver. 1.1 by Dmitry A. Kazakov [1996]. For readers convenience, the web pages of the software library are attached to this manual in the Appendix, Sec. 17.2, pp. 445.

In rlab, we identify two types of strings: plain, which are just what their name implies, a collection of characters in double quotation marks; and pattern, which is Pattern Matching-compliant string. To signal to rlab that the string is a pattern, it needs to start with single quotation mark, as in the following example:

```
csp = "'BLANK";
```

rlab passes patterns directly to software library after removing the leading single quotation mark. In this particular case “BLANK” is a built-in pattern that matches any continuous sequence of blank spaces and tabulators. How to construct patterns past the first single quotation mark is described in the Appendix, Sec. 17.2, pp. 445. Here we provide few characteristic examples:

```plaintext
note = "''/*' *(END/|%): '*/'"; // c-lang type comments
csp = "'(BLANK|','|':')"; // blank space/tab/comma/column separated values
comment = "'(('#'|'%'|'//')"; // hash or percent character, or two slashes in a row
```

Because of rlab convention with leading single quotation mark as indicator for the pattern, this is how one passes single quotation mark as a part of the pattern; then the pattern has to be enclosed in double quotation marks.

```
>> x = "'"'"'"
ans
   '"
```

7.3.1 ascii

**Format:** \( x = \text{ascii}(s) \)

**Arguments:**

1. \( s \), a string.

**Result:** \( x \), an integer row-vector.

**Abstract:** ascii writes the ASCII codes of the characters in the string entry \( s \).

7.3.2 char

**Format:** \( sx = \text{char}(idx) \)

**Arguments:**

1. \( idx \), a matrix with integer entries.

**Result:** \( s \), a string column-vector.
Abstract: *char* creates a single-column string matrix, consisting of the strings the ascii codes of which are given as the rows of integer matrix *x*.

**Note:** Replaces *ascii()* that used to be in $RD/rlib$.

\[
\begin{array}{l}
\texttt{>> x = ascii("alphabet")}
\texttt{>> char(x)}
\texttt{alphabet}
\end{array}
\]

### 7.3.3 blank

**Format:** \(sx = \text{blank}(nr, nc, fc)\)

**Arguments:**

1. \(nr\), an integer;
2. \(nc\), an integer;
3. \(fc\), a fill-in string.

**Result:** \(sx\), a string matrix of size \(nr\)-by-\(nc\) containing \(fc\) for all entries.

### 7.3.4 num2str,text

**Format:** \(sx = \text{num2str}(x, fmt, sep, ci)\)

**Arguments:**

1. \(x\), real or complex matrix;
2. \(fmt\), string matrix, c-compatible format;
3. \(sep\), string, separator between the column entries;
4. \(ci\), string, symbol for complex unity.

**Result:** \(sx\), string matrix.

**Abstract:** Converts the content of the dense real or complex matrix \(x\) to a string matrix according to the format \(fmt\) and separator \(sep\). If \(sep\) is not given then \(sx\) is a matrix of the same size as \(x\). If \(sep\) is given, then \(sx\) is a string column-vector, where the row-wise entries are separated by \(sep\).
By using “+” prepended to the format string, the numerical entries will be printed with “+” if positive, or with “-” if negative.

If \( x \) is complex then \( ci \) determines the format for writing the complex unity (default value is “+1i*”). The sign of the complex part can be either prepended (use “+” as the first character of \( ci \)) or appended to the complex unity.

### 7.3.5 strlen

**Format:** \( x = strlen(sx) \)

**Arguments:**

1. \( sx \), string matrix;

**Result:** If \( sx \) is a string, then \( x \) returns its length. Otherwise it returns \(-1\).

### 7.3.6 lstrip

**Format:** \( x = lstrip(s, l) \)

1. \( s \), string matrix.

2. \( l \), string scalar, either plain or pattern.

**Result:** Removes all occurrences of pattern \( l \) as long as they match the beginning of the string \( s \).
7.3.7 rstrip

Format: \( x = \text{rstrip}(s, r) \)

1. \( s \), string matrix.
2. \( r \), string scalar, either plain or pattern.

Result: Removes all occurrences of pattern \( r \) as long as they match the end of the string \( s \).

7.3.8 strip

Format: \( x = \text{rstrip}(s, l, r) \)

1. \( s \), string matrix.
2. \( l \), string scalar, either plain or pattern.
3. \( r \), string scalar, either plain or pattern.

Result: Removes all occurrences of patterns \( l \) and \( r \) as long as they match the beginning or the end of the string.

```
>> x="aa12345bb"
ans =
aa12345bb
>> strip(x,"a","b")
ans =
12345
```

7.3.9 strtod

Format: \( x = \text{strtod}(s/, \text{opts/}) \),

Arguments:

1. \( s \), string column-vector (e.g., such that came out of \text{reads}), or string scalar.
2. \( \text{opts} \), list, \( \text{opts} = \ll \text{skiprows}; \text{csp}; \text{min\_len}; \text{comment}; \text{note}; \text{use\_cols}; \text{use\_rows}; \text{join\_rows}; \text{join\_csp}; \text{lstrip}; \text{start}; \text{stop} \gg \),
   where the options are
   - \text{skiprows}, integer, number of rows of \( s \) to skip before trying numeric conversion;
   - \text{csp}, string pattern, delimiter between numeric data columns in single entry of \( s \);
   - \text{min\_len}, integer, minimum length of string entry of \( s \) to try numeric conversion on;
   - \text{comment}, string pattern, start of the string, the end of which is the end of string (or end of line), which is removed prior to numeric conversion;
   - \text{note}, string pattern, description of substring that is removed from a single entry of \( s \) prior to numeric conversion. Please note, \text{comment} is a \text{note} which stretches to the end of the string entry;
   - \text{use\_cols}, integer array, indices of columns in converted data that are to be returned as the result of conversion;
• **use_rows**, integer array, indices of rows in converted data that are to be returned as the result of conversion;
• **join_rows**, integer, number of consecutive (entries) rows in \( s \) that are to be concatenated prior to conversion;
• **join_csp**, string, additional column separator inserted between joined rows (default is blank space);
• **lstrip**, string pattern, to be removed from the string if the string starts with it;
• **start**, string pattern, identifier in entries in \( s \), presence of which initiates conversion starting from the entry after this one;
• **stop**, string pattern, identifier in entries in \( s \), presence of which terminates the conversion. The conversion may start again if another **start** pattern is found.

**Result:** \( x \), real matrix, result of conversion of string column vector \( s \) to numeric values.

**Abstract:** Converts column vector of strings to numeric values provided the specification in entries of the list **opts**.

**Example:** Consider string column-vector, where the c-language type comments mark the debugging messages from weather station (number of successful pings from 60 attempts), and which are irrelevant for the weather data

\[
s = [ ... \\
  '\# weather log file by mj@pitagora started on 20150522-000002'; ... \\
  '\# hour, min, sec, rel.humidity (%), deg Celsius, rain gauge count'; ... \\
  '\00,01,02/*60/60*/,53,15.3,0''; ... \\
  '\00,06,02/*60/60*/,52,15.4,0''; ... \\
  '\00,11,01/*60/60*/,51,15.6,0''; ... \\
  '\00,16,02/*60/60*/,53,15.5,0''; ... \\
  '\00,21,01/*60/60*/,51,15.6,0''; ... \\
  '\00,26,02/*60/60*/,55,15.4,0''; ... \\
  '\00,31,01/*60/60*/,54,15.2,0'' ];
\]

This code converts it to numbers

```plaintext
1 opts = <<>>;
2 opts.csp = ","; // comma is data field separator
3 opts.comment = "#'|(BLANK'#')"; // comment start with hash, or (space or tab)-then-hash
4 opts.note = "'*/' *(END/\%): '*/"; // pattern description of the c-language comment
5 opts.min_len=1; // ignore empty lines, just in case
6 y = strtod(sx, opts);
```

**Example:** Consider a case of string, where the delimiter (space) is also what the string starts with,

\[
x=[ ... \\
  "1 -1.5625 1.3964 -1.5566 1.1010 -0.8128 0.5643"; ... \\
  " 1 -1.5625 1.3964 -1.5566 1.1010 -0.8128 0.5643" ];
\]
This code converts it to numbers

```c
opts = <<>>;
opts.csp = "\'BLANK"; // space-or-tab as column separator
opts.lstrip = "\'BLANK"; // remove leading space-or-tab's
y2 = strtod(x, opts)
```

**Example:** Consider a case of string, where one datum stretches over two lines, with some text on the sides,

```c
x=[ ... 
"VECTOR_1 -1.5625 1.3964 -1.5566 1.1010 -0.8128 0.5643"; ... 
"CLASS C2", ... 
"VECTOR_2 -1.8170 1.7104 -1.4251 0.7118 -0.3490 0.2799"; ... 
"CLASS C3" ];
```

This code converts it to numbers

```c
opts = <<>>;
opts.csp = "\'BLANK";
opts.note = ["VECTOR_", "CLASS C"];
opts.join_rows=2;
opts.join_csp= " ";
y2 = strtod(x, opts)
```

Interested reader will find more examples in test directory, in subdirectory `strings`.

### 7.3.10 strindex

**Format:** \( r = \text{strindex}(s,p) \)

**Arguments:**

1. \( s \), string matrix;
2. \( p \), string scalar, either plain or pattern.

**Result:** \( r \), integer matrix.

**Abstract:** Locates first left position of pattern \( p \) in every entry of string matrix \( s \) and returns its position, or 0 if \( p \) was not found. It is equivalent to *gawk* command `index`.

Consider an example:

```c
>> s = randchar(3,3,15)
1NxTBf6IjKL41U Zg0p4SSQdVoOMWF L16D1Mg1l3mKSkp
98XDMouIkhdlq82o Knc204e1EQoMBqQ LcBS14BwnMtqTnj
vyc3HSD9dlc5nm 28J1c5bSG6fqANC wG03n6We7MUNRg8
>> strindex(s, "03")
0 0 0
0 0 0
0 0 3
```
7.3. STRING FUNCTIONS

7.3.11 strindexr

Format: \( r = \text{strindexr}(s, p) \)

Arguments:

1. \( s \), string matrix;
2. \( p \), string scalar, either plain or pattern.

Result: \( r \), integer array.

Abstract: Locates first left position of pattern \( p \) in \( s \) and returns indices corresponding to it. The function is useful if the pattern is pattern matching-compliant of indeterminate length, because then it can be combined with \( \text{substr} \) to get the exact text of the matched pattern.

7.3.12 grep

Format: \( r = \text{grep}(s, p) \)

Arguments:

1. \( s \), string column vector;
2. \( p \), string scalar, either plain or pattern.

Result: \( r \), string array.

Abstract: Copies in \( r \) entries from \( s \) that contain pattern \( p \).

7.3.13 findvec

Format: \( r = \text{findvec}(x, s) \)

Arguments:

1. \( x \), real vector;
2. \( s \), real vector.

Result: \( r \), real vector.

Abstract: Does for real vectors what \( \text{findstr} \) does for strings: It finds all positions of an array \( s \) within a bigger array \( x \). If it does not find array \( s \), then it returns an empty array.

7.3.14 substr

Format: \( s = \text{substr}(x, r) \)

Arguments:

1. \( x \), string matrix;
2. \( r \), integer vector.
Result: $s$, string matrix.

Abstract: For each entry in $x$ creates a substring from the characters which indices are given in $r$. The array of indices need not be sorted, and can extend beyond the range of entry in $x$. It is equivalent to the 	exttt{gawk} command 	exttt{substr}:

```
>> x=randchar(1,1,15)
zyeK2Af5CAe7u6I
>> substr(x,[1,3,5,7,8])
ze2f5
```

Note: To create a substring extending to the end of the string one just needs to set the range beyond the end of the string.

```
>> substr(x,[7:100])
f5CAe7u6I
```

### 7.3.15 gsub

Format: $x = \texttt{gsub}(r,s,t)$  

Format: $x = \texttt{gsub}(s,t)$

Arguments:

1. $r$, string, plain;
2. $s$, string, plain or pattern;
3. $t$, string matrix.

Result: $x = <<<\text{count;\ string }>>>$, list, with entries:

- $\text{string}$, string matrix, result of substitution of $s$ by $r$ for each entry in $t$;
- $\text{count}$, real matrix, count of how many substitutions were performed in each entry in $t$.

Abstract: In each entry of string matrix $t$ it replaces every occurrence of pattern $s$ with the plain string $r$. If plain string $r$ is absent, then the pattern $s$ is removed from entries of $t$.

```
>> x=randchar(1,1,16)
zyeK2Af5CAe7u6IG
>> x
zyeK2Af5CAe7u6IG
>> sx = gsub("_YEAH_", "ye", x+x)
count  string
>> sx.string
z_YEAH_K2Af5CAe7u6IGz_YEAH_K2Af5CAe7u6IG
>> sx.count
2
>>
```
7.3. STRING FUNCTIONS

7.3.16 tolower

Format: \( s = \text{tolower}(x) \)

Arguments:

1. \( x \), string matrix.

Result: \( s \), string matrix.

Abstract: Converts all uppercase characters to lowercase in \( x \).

7.3.17 toupper

Format: \( s = \text{toupper}(x) \)

Arguments:

1. \( x \), string matrix.

Result: \( s \), string matrix.

Abstract: Converts all lowercase characters to uppercase in \( x \).
170

CHAPTER 7. GENERAL PURPOSE FUNCTIONS

7.3.18

unpack

Format: x = unpack(s)
Arguments:
1 s, string scalar, contains json-formatted data.
Result: x, extracted data.
Abstract: Converts json-formatted string to rlab-data structure.
Consider json-string that was received from the Mathworks web site www.thingspeak.com in response
to curl-backed query,
>> s = readm(‘‘https://api.thingspeak.com/channels.json?api\_key=AAAAAAAAAAAAAB’’,
<<CURLOPT\_HTTPGET=1>>);
>> s?
{"id":217701,"name":"ISS","description":"Data from the International Space Station:
temperature (deg C), relative humidity (%) and altitude (km).",
"latitude":"55.8806","longitude":"38.1105",
"created_at":"2016-09-26T19:45:58Z","elevation":"","last_entry_id":532,
"public_flag":true,"url":"","ranking":50,"metadata":"","license_id":0,
"tags":[],"api_keys":[{"api_key":"WRVV37N67MUMP2W9","write_flag":true},
{"api_key":"4CMDON5DGZGVHEUA","write_flag":false}]}
>> y = unpack(s);
ans =
api_keys
created_at
description
elevation
id
last_entry_id
latitude
license_id
longitude
metadata
name
public_flag
ranking
tags
url
>> y.api_keys[2].api_key
ans =
4CMDOF5DGZGVHEUA
In the above example, y is a list, which entries are string or double scalars except for the entry api keys,
which is a cell: It has as entries two lists <<api key; write flag>>.
Note: : the keys in the list are lexicographycally sorted irrespectively of the order in which they appear in
json-string.


Example: Here, we count occurrence of individual characters in the English translation of the GNU Public License (files COPYING and eg_string2.r in directory test/strings). The xmgr compliant file the script creates was imported in xmGrace and from there exported as an embedded Postscript file, given on p.172.

```c
// file: eg_string2.r

//
filename="./COPYING"

y=toupper(reads(filename));
alphabet = create_charactertable();
cntalpha = zeros(alphabet.nr,1);

for(i in 1:y.nr)
{
    for(j in 1:alphabet.nr)
    {
        dy = y[i];
        dummys = gsub("", alphabet[j], dy );
        cntalpha[j] = cntalpha[j] + dummys.count;
        y[i] = dummys.string;
        if(length(y[i])==0) { break; }
    }
}

idx = sort( cntalpha ).ind;
cntalpha = cntalpha[idx];

// paper, P=0
xmgfilename("gpl_count.gr",0);
xmpaper ("letter", "landscape", 0);

// graph, N=0
xmgtitle ("Character Count in General Public License (GPL)", 1.25, 0);
xmgrange ([-0.5,25.5,0,2000],0);
xmgylabels("Character of English Alphabet", "Occurence", 1.25, 0);
xmgxticks ( [1,0], 1.25, 0);
xmgyticks ( [250,4], 1.25, 0);
xmgxtickspec(0:length(alphabet)-1, alphabet[idx], 0);

// dataset, I=0
xmgdataset (cntalpha, "bar", ",", 0);
xmgstyle ("line", [0,0,0], 0);

// print all
xmggraph (0,0);
xmgscale (0,0);
```
Figure 7.1: The count of characters in English alphabet in the English translation of the General Public License (GPL).
7.4 Terminal control functions

The functions in this library use system-wide libtermcap.so library to give user control over appearance of terminal.

7.4.1 clrscr
Format: clrscr()
Abstract: Clears the current rlabplus screen.

7.4.2 mvcrsr
Format: mvcrsr(row,col)
Arguments:
1 row, integer, row on the screen where the cursor is to be positioned. Top of the screen is 1;
2 col, integer, column on the screen where the cursor is to be positioned. Top of the screen is 1;

Abstract: Positions the cursor at the location (row, col) of the screen. The top left corner is at position (1,1).

7.4.3 clrpos
Format: clrpos(/rows,cols/)
Arguments:
1 rows, integer scalar or range, indices of the rows on the screen;
2 cols, integer scalar or range, indices of the columns on the screen;

Abstract: Clears the given range of rows, or in a given range of rows clears the columns with indices in cols. If no argument is given that the line is cleared from the current position of the cursor. The clearing is done either by using the native termcap code for “clear line,” or by filling the range with “spaces.”

7.4.4 colors
Format: colors(/fg/,/bg/)
Arguments:
1 fg, string, name of the foreground (text) color. The name of the color may be “default”, “red”, “green”, “yellow” or “brown”, “blue”, “magenta”, “cyan”, “grey” or “white”;
2 bg, string, name of the background color;

Abstract: Puts the rlabplus terminal in color mode with specified background and foreground color.

Note: colors() resets the color scheme to default.
Example: Consider code that puts random characters in random colors in random order on the current 
rlabplus screen. See rlab/test/termcap for full code.

```plaintext
1 // colors, "yellow" = "brown", and "grey" = "white"
2 cl = ["default", "red", "green", "yellow", "blue", ...
3     "magenta", "cyan", "grey"];
4
5 // terminal size
6 mx = strtod( getenv("COLUMNS") );
7 my = strtod( getenv("LINES") );
8
9 // prepare rng no. 1 for uniform distribution in [0,8]
10 rng(1, "uniform", 0, 8);
11
12 // prepare drng no. 1 to generate capital letters of the alphabet
13 drng(1, char([65:90]'), ones(26,1));
14
15 // figure out all the (y,x) for the terminal
16 ixy = [];
17 for (i in 2:my)
18 { ixy = [ixy; [1:mx]', i*ones(mx,1)]; }
19
20 // randomize order of all (y,x)
21 ixy = shuffle(ixy);
22
23 // write characters
24 clrscr(); // clear screen
25 mvcrsr(1,1); // put cursor at home position at (1,1)
26 clrpos (1); // clear first line
27 printf(" Fill the screen with random characters in random order");
28 sleep(0.5); // wait
29 for (i in 1:ixy.nr)
30 {
31    i1 = int(rand()) + 1; // choose random integer in 1..8 range
32    mvcrsr (ixy[i;2], ixy[i;1]); // position cursor
33    colors ( cl[i1] ); // set color of text
34    printf ( drand(1) ); // print random character from drng
35    sleep (0.001); // wait a little
36 }
37 colors(); // reset terminal to defaults
```
7.5 Read/Write Operations

Changes from rlab ver.1

Universal Resource Locator

For reading and writing data rlab uses their universal resource locator (URL), in the form

```
protocol://address
```

rlab can handle the following protocols,

I. **FILE** (regular text and binary files):
   
   - `protocol` = "file" or can be omitted, while `address` is the path to the file location on the accessible file system, e.g., "file://somefile.someext" or "somefile.someext".
   
   Read and write operations of ascii-files are facilitated through commands `readm`, `reads` and `writem`, while of binary files through `read`, and `write`.

II. **HDF5**:
   
   For accessing files that use Hierarchical Data Format 5 (HDF5) version 1.8.0, and later, (i), `protocol` has to be set to "hdf5" or "h5", while `address` contains the path to the file location on the accessible file system; or (ii), `protocol` can be omitted while `address` ends with "h5" (file extension), e.g., "h5://somefile.someext" or "hdf5://somefile.someext" or "somefile.h5".
   
   Given the richness of data types in HDF5, in rlab only data sets containing, so called, atomic data types are supported (real-valued and string scalars, vectors and matrices). Additionally, the following three, so called, compound types are supported: 1. complex-valued scalars, vectors and matrices; 2. real- and 3. complex- valued sparse matrices.
   
   Read and write operations are facilitated through commands `readm`, `read`, `write` and `writem`.

III. **World Wide Web**:
   
   For accessing content of web and ftp sites, `protocol` and `address` correspond to what is known as the URL of the object, be it a web page, or a file. The protocols "http", "https", and "ftp" are supported, e.g., "http://www.gnu.org".
   
   Read and write operations are facilitated through commands `readm`, and `writem`.

IV. **Socketry**:
   
   For accessing Un*x type sockets, `protocol` has to be set to "tcp", while `address` contains a valid internet address (v4) and a port number separated by the column ":", e.g., tcp://127.0.0.1:4242.
   
   Read and write operations are facilitated through commands `readm`, and `writem`.
   
   This is an experimentally implemented feature, which is in part based on netcat by [Giacobbi 2004](#).

V. **Serial Port**:
   
   For accessing serial port on Un*x type systems, `protocol` has to be set to "serial", while `address` contains a valid absolute path to the serial block device, e.g., “serial://dev/ttyS0”.
   
   Read and write operations are facilitated through commands `readm`, and `writem`.

For URL’s open through the command `open` rlab provides an internal book-keeping: the URL’s are kept open until closed through `close` or until user exits rlab.

**Note**: Some files can be accessed or created without `open/close`, that is by calling the functions `readm`, `writem`, `read`, or `write` right away. If that is the case, then rlab temporary opens/creates a file, and upon
completion of the operation it closes the file. For that reason, for example, a repeated execution of the following two scripts

```plaintext
// Script No. 1
fn = 'somefile.txt';
open ('somefile.txt', 'r');
x = readm(fn);  
```

and

```plaintext
// Script No. 2
fn = 'somefile.txt';
x = readm(fn);  
```

will produce different result.

Upon first execution, the script No.1 will open the file and read its content into x. In the second execution, `open` will be ignored, and x will become an empty variable because the end of file has been reached during the first `read`. Script No. 2 will at each execution produce the same result: As the file was not previously open, `rlab` will temporary open the file, read its content and then close it.

For the script No. 1 to produce the same result as the script No. 2 a line with `close (fn);` has to be added at the end.
7.5. READ/WRITE OPERATIONS

7.5.1 open

I. FILE

**Format:** `open(url, mode, options)`

**Arguments:**

1. `url`, string, file name;
2. `mode`, string, access mode: “r” for read, “w” for write, “a” for append;
3. `options`, optional list with entries `eol`, `csp`, `format`, `buffer_size`, where
   * `eol`, string for end-of-line sequence;
   * `csp`, string for column separator;
   * `format`, string matrix containing format (using matrix optimization) in which data matrix `x` will be written;
   * `buffer_size`, integer, size of the buffer.

II. HDF5

**Format:** `open(url, mode)`

**Arguments:**

1. `url`, string, universal resource locator in the standard format “`protocol://address`”, where `protocol` is `h5`, or `hdf5`, while `address` is the file name. Protocol can be omitted but then the file name has to end with “.h5”;
2. `mode`, string, access mode: “r” for read, “w” for write, “a” for append;

III. World Wide Web

**Format:** `open(url, curlopts)`

**Arguments:**

1. `url`, string, universal resource locator that includes protocol and the address of the resource in the standard format “`protocol://address`”. Protocols `http`, `https` and `ftp` are supported, while `address` is the WWW address of the resource;
2. `curlopt`, list, contains entries which provide necessary parameters for the connection, according to `cURL` by [Stenberg 2009](http://curl.haxx.se/libcurl/c), so called “easy” interface. The entries in the parameter list which are used to specify the connection have the verbatim names as specified in the API for the library (cf. `http://curl.haxx.se/libcurl/c`). We just list them here: `CURLOPT_URL; CURLOPT_PROXY; CURLOPT_PROXYTYPE = “http”, “socks4a”, “socks4”, “socks5”, “socks5h”; CURLOPT_HTTPPROXYTUNNEL =0,1; CURLOPT_INTERFACE; CURLOPT_PORT; CURLOPT_FOLLOWLOCATION=0,1; CURLOPT_UNRESTRICTED_AUTH = 0,1; CURLOPT_MAXREDIRS; CURLOPT_TCP_NODELAY=0,1; CURLOPT_USERAGENT; CURLOPT_REFERER; CURLOPT_COOKIE; CURLOPT_COOKIEFILE; CURLOPT_COOKIEJAR; CURLOPT_COOKIESESSION=0,1; CURLOPT_HTTPGET=0,1; CURLOPT_HTTPGET=0,1; CURLOPT_CRLF=0,1; CURLOPT_POSTQUOTE; CURLOPT_SSL_VERIFYPEER=0,1; CURLOPT_SSL_VERIFYHOST=0,1; CURLOPT_PROXYUSERPWD; CURLOPT_PROXYUSERPWD; CURLOPT_HTTPAUTH = “basic”, “digest”, “gss”, “ntlm”, “any safe”, “any”; CURLOPT_VERBOSE = 0,1; CURLOPT_STDERR; CURLOPT_TIMEOUT.
CHAPTER 7. GENERAL PURPOSE FUNCTIONS

IV. Socket

**Format:** `open(url/, options/)`

**Arguments:**

1. *url*, string, universal resource locator that includes protocol and the address of the resource in the standard format “`protocol://address`”. Here *protocol* is “tcp”, while *address* is the internet address (either as a number or as a name that needs resolving), and the port number, where the two are separated by the column “:`”;

2. *options*, list with entries
   - (a) *mode*, string, “connect” or “listen”;
   - (b) *timeout*, real scalar, time in seconds.

V. Serial Port

**Format:** `open(url/, options/)`

**Arguments:**

1. *url*, string, universal resource locator that includes protocol and the address of the resource in the standard format “`protocol://address`”. Here *protocol* is “serial”, while *address* is the file name of the serial port as reported by `setserial` or similar during boot time, e.g., “serial:///dev/ttyS0”;

2. *options*, list, with entries
   - 1. *data_parity_stop*, string, e.g., “8N1” for 8-bit, no-parity, 1 stop bit communications. First digit can be 5-8, second character is “N” (no parity), “E” (even parity) or “O” (odd parity), last digit is a number of stop bits (1 or 2);
   - 2. *speed*, integer, communication rate in bits-per-second, e.g., 9600, or some such number;
   - 3. *flow_control*, string, “(h)ardware,” or “(x)on—xoff” (software), or “(n)one”. The options are set internally to be mutually exclusive;
   - 4. *raw*, integer, 0 for not raw or 1 (default);
   - 5. *debug*, string, can be “hex,” “char” or “int.” Determines the way all communication received from the serial port is going to be written on the screen.
   - 6. *hupcl*, integer, 0 or 1. Determines if the serial port will be hang-up on close. May be useful for Arduino boards.
   - 7. *eol*, string, end-of-line character to be used in transmissions. E.g., “\n” (new line) or “\r” (carriage return).

**7.5.2 close**

**Format:** `close(s)`

**Arguments:**

1. *s*, string, universal resource locator used in `open(…)`;

**Abstract:** Closes the respective universal resource locator and releases rlab’s internal resources allocated for communication to it.
7.5. READ/WRITE OPERATIONS

7.5.3 writem

I. Plain text files:

Format: \texttt{writem(url, x, options)}

Arguments:

1. \texttt{url}, string, universal resource locator for a file;
2. \texttt{x}, string, real- or complex-valued matrix;
3. \texttt{options}, optional list with entries \texttt{eol; csp; format; buffer\_size}, where
   * \texttt{eol}, string for end-of-line sequence;
   * \texttt{csp}, string for column separator;
   * \texttt{format}, string matrix containing format (using matrix optimization) in which data matrix \texttt{x}
     will be written;
   * \texttt{buffer\_size}, integer, size of the buffer.

Abstract: Writes the content of matrix \texttt{x} to the file located at \texttt{url}. If the file has been
opened previously using \texttt{open}, then writing continues from the current position, otherwise
the file is closed after writing (which then makes the content of \texttt{x} being the content
of the file).

II. HDF5

Format: \texttt{writem(url,/object/,x)}

Arguments:

1. \texttt{url}, string, universal resource locator for an HDF5 file;
2. \texttt{object}, string, absolute name of the object in the HDF5 file that \texttt{x} will be written to. If \texttt{object}
is omitted then the name of the variable is used as the absolute path of the HDF5 object (with \\
\texttt{/} prepended) to which the variable is written;
3. \texttt{x}, string, real- or complex-valued matrix, dense or sparse.

Abstract: Writes the content of a native \texttt{rlab} variable \texttt{x} to an HDF5 file \texttt{url} as \texttt{object}.

Note: \texttt{rlab} stores the matrices internally in FORTRAN-like fashion (column-dominated storage).
Thus, during reading or writing using HDF5 the matrices are transposed in the process because of the
latter’s internal row-dominated storage model.

For that reason to each object created by \texttt{rlab}, to which a matrix or a vector is written, an attribute
“Transpose” is appended with an integer value of 1. When \texttt{rlab} reads a matrix from HDF5 file, if it
does not find that attribute (with that value) then it internally transposes the read matrix \textit{in situ} \cite{Catte & Twigg, 1977}.

Note: \texttt{rlab} has three native data types, namely dense complex-valued matrices, and sparse real- and
complex-valued matrices, for which there is no atomic type in HDF5. These objects are written as
follows:

1. Complex matrices are written as matrices of compounds, where the compound is comprised of
two numbers in \texttt{double} precision, where the first is named “real” and the second “imag”. As
with real or string matrices these are transposed during reading or writing, so they also have the
integer scalar attribute “Transpose” attached to them with value 1.
2. Sparse matrices are written as scalar compounds, each comprised of 7 variable length arrays, namely: \( nr \) of length 1, \( nc \) of length 1, \( nnz \) of length 1, \( order \) of length 1, \( ia \) of length \( nr + 1 \), \( ja \) of length \( nnz \), all of \texttt{integer} precision; and, \( d \) of length \( nnz \) (for real), or \( c \) of length \( 2 \cdot nnz \) (for complex), all of \texttt{double} precision. In this way the memory allocated to a sparse matrix does not have to be copied for HDF5 to write it to the file. Interested reader can examine the routines \texttt{h5_write_rlab_atomic_msr} and \texttt{h5_write_rlab_atomic_msc} in \texttt{rfileio.hdf5.c} in source code for more technical details.

III. World Wide Web

\textbf{Format:} \texttt{writem(url, filename), writem(url, , x)}

\textbf{Arguments:}

1. \textit{url}, string, universal resource locator used in \texttt{open(...)}, e.g., \texttt{“http://rlabplus.sourceforge.net”}.
2. \textit{filename}, string vector, name of the local file that is going to be transferred to the \textit{url} per specifications given in \texttt{open(...)};
3. \textit{x}, string, a variable which textual content will be transferred to the \textit{url}.

IV. Socketry

\textbf{Format:} \texttt{writem(url, x)}

\textbf{Arguments:}

1. \textit{url}, string, universal resource locator used in \texttt{open(...)}, e.g., \texttt{“tcp://127.0.0.1:4242”} (cf. \texttt{MozRepl} for port 4242);
2. \textit{x}, string, or integer or double vector, contains data to be written to the socket byte-wise.

V. Serial Port

\textbf{Format:} \texttt{writem(url, s)}

\textbf{Arguments:}

1. \textit{url}, string, universal resource locator used in \texttt{open(...)}, e.g., \texttt{“serial:///dev/ttyS0”};
2. \textit{s}
   * string vector, data which is to be written to the serial port; or
   * list which entries can be \texttt{dtr} = 0, 1 or \texttt{rts} = 0, 1 to assert the DTR or RTS lines on the port. This may be useful in controlling the devices on the port (e.g., Parallax’ RFID tag reader uses asserting one of the lines to start the transmitter and initiate the reading of a tag).

\textbf{Note:} Before writing the port is flushed. If for what ever reason the write operation is unsuccessful, the code sleeps 1 msec, flushes the port and tries to write to it again.
7.5. READ/WRITE OPERATIONS

7.5.4 readm

I. plain text files

**Format:** \( x = \text{readm}(url, n), \) or

**Format:** \( x = \text{readm}(url, opts) \)

**Arguments:**

1. *url*, string, universal resource locator for a file;
2. *n*, integer, number of rows from the first to be skipped at reading. E.g., comment associated with a data set;
3. *opts*, list, \( opts = \langle \text{skiprows}; \text{csp}; \text{min}_\text{len}; \text{comment}; \text{note}; \text{use}_\text{cols}; \text{use}_\text{rows}; \text{join}_\text{rows}; \text{join}_\text{csp}; \text{lstrip}; \text{start}; \text{stop}; \text{grep} \rangle \)

   where the options are
   
   - *skiprows*, integer, number of rows of *s* to skip before trying numeric conversion;
   - *csp*, string pattern, delimiter between numeric data columns in single entry of *s*;
   - *min_len*, integer, minimum length of string entry of *s* to try numeric conversion on;
   - *comment*, string pattern, start of the string, the end of which is the end of string (or end of line), which is removed prior to numeric conversion;
   - *note*, string pattern, description of substring that is removed from a single entry of *s* prior to numeric conversion. Please note, *comment* is a *note* which stretches to the end of the string entry;
   - *use_cols*, integer array, indices of columns in converted data that are to be returned as the result of conversion;
   - *use_rows*, integer array, indices of rows in converted data that are to be returned as the result of conversion;
   - *join_rows*, integer, number of consecutive (entries) rows in *s* that are to be concatenated prior to conversion;
   - *join_csp*, string, additional column separator inserted between joined rows (default is blank space);
   - *lstrip*, string pattern, to be removed from the string if the string starts with it;
   - *start*, string pattern, identifier in entries in *s*, presence of which initiates conversion starting from the entry after this one;
   - *stop*, string pattern, identifier in entries in *s*, presence of which terminates the conversion. The conversion may resume if another *start* pattern is found.
   - *grep*, string pattern, forces readm to process only rows that contain provided pattern.

Please note, the option list is identical to that for the function `strtod`.

**Result:** \( x \), real dense matrix.

II. HDF5

**Format:** \( x = \text{readm}(url, object, /coord/) \)

**Arguments:**

1. *url*, string, universal resource locator for an HDF5 file;
2 *object*, string, absolute name of the object in the HDF5 file will be read. This can be an atomic
data type (real, or string), or special compounds of complex number type, or *rlab* native sparse
matrix that appears as a special compound of 7 variable length arrays;

3 *coord*, integer matrix, which rows contain the coordinates in the 1-notation (first coordinate is 1,
compared to 0), representing the selection of the data that is to be read from the *object*, rather
than reading the entire object.

**Result:** $x$, matrix containing the data set in *object*. If *coord* is provided then $x$ contains that many
rows that there are rows in *coord*.

III. World Wide Web

**Format:** $x = \text{readm}(\text{url}), \text{or} \quad \text{readm}(\text{url}, \text{filename})$

**Arguments:**

1 *url*, string, universal resource locator that is the same as used in `open(...)``;
2 *filename*, string vector, name of a local file to which the data from *url* will be transferred as per
specifications given in `open``;

**Result:** $x$, string, content of the URL.

IV. Socketry

**Format:** $\text{writem}(\text{url}, x)$

**Arguments:**

1 *url*, string, universal resource locator that is the same as used in `open(...)`, e.g., “tcp://127.0.0.1:4242” (cf. *MozRepl* for port 4242);
2 *x*, string, or integer or double vector, contains data to be written to the socket byte-wise.

V. Serial Port

**Format:** $c = \text{readm}(\text{url}/, n\text{bytes}/), \text{or} \quad c = \text{readm}(\text{url}, \text{cmds}/, n, n\text{bytes}/)$

**Arguments:**

1 *url*, for the serial port communication, e.g., “serial:///dev/ttyS0”;
2,4 *nbytes*, integer, number of bytes (characters) to be read from the *url*.
2 *cmds*, string vector,
3 *n*, positive integer. In the second form it is used together with the string vector *cmds* to form a
query to the serial port that is performed *n*-times in an internal loop, and to which the response
is recorded. If *n* is not given, *n* = 1 is assumed;

**Result:** $c$, string vector of length *n*, where each entry is of length *nbytes*. 
7.5. **READ/WRITE OPERATIONS**

7.5.5 **write**

II. Binary

**Format:** \texttt{write(url,x1,x2,...)}

**Arguments:**

1. \texttt{url}, string, universal resource locator for a file;
2. \ldots \texttt{x1}, any \texttt{rlab} variable excluding functions and protected lists.

**Abstract:** Writes \texttt{x} to a file \texttt{url} in binary format.

III. HDF5

**Format:** \texttt{write(url,x1,x2,...)}

**Arguments:**

1. \texttt{url}, string, universal resource locator for an HDF5 file;
2. \ldots \texttt{x1}, any \texttt{rlab} variable excluding functions and protected lists.

**Abstract:** Writes \texttt{x} to an HDF5 file \texttt{url}, where the object to which the variable is written is determined from the structure of \texttt{x}. E.g., if \texttt{x} is the numeric type then its name with “/” prepended becomes the name of the corresponding object, e.g., the content of \texttt{rlab} variable named \texttt{y} containing a dense real matrix will be written as an object “/y”. E.g., if \texttt{x} is a list, then all its elements are visited, and non-empty ones only are written to the file using a following translation scheme:

\[(\texttt{rlab}) \texttt{x}.[\texttt{level1}].[\texttt{level2}].[\ldots].\texttt{levelN} \rightarrow /\texttt{x}/\texttt{level1}/\texttt{level2}/\ldots/\texttt{levelN} \texttt{(HDF5)}\]

**Note:** In the writing process \texttt{rlab} transposes the matrices. See comments regarding \texttt{writem}.

7.5.6 **read**

**Format:** \texttt{read(url,x)}, or \texttt{read(url)}

**Arguments:**

1. \texttt{url}, string, universal resource locator for a file;
2. \texttt{x}, any \texttt{rlab} variable excluding functions and protected lists.

**Abstract:** Reads the content of the file \texttt{url} in binary, or in HDF5, format to the variable \texttt{x} (\texttt{rlab} creates it if it does not exist), or into the global workspace.

**Note:** HDF5 is made interchangable with the native binary format. It is my sincerest recommendation to switch from the \texttt{rlab} native binary format and to the HDF5 wherever possible.

**Note:** In the future I plan to expand support for HDF5 within \texttt{rlab}, namely, introduce some support for general compound types, e.g., for accessing elements within a compound.
Specialized read/write functions

7.5.7 fwrite

Format: `fwrite(filename, data, format)`

Arguments:
1. `filename`, string;
2. `data`, matrix containing real, complex or string data to be written to file;
3. `format`, string, describes how entries from `data` will be written to file. Acceptable values are “int8”, “byte”, “char” or “integer*1” for single byte integers; “int16”, “short”, “integer*2” for two-byte integers; “int32”, “integer*4” or just “int” for 4-byte integers; “float”, “real” or “real*4” for single precision real numbers; and, “double” or “real*8” for double precision real numbers. If `data` is complex matrix then the same formats are accepted but applied to real and imaginary part. If `data` is string, than it is written as such.

Abstract: Writes raw numerical or string data to file.

7.5.8 fread

Format: `x = fread(filename, size, format, swap)`

Arguments:
1. `filename`, string;
2. `size`, integer scalar, length (number of entries) of the data vector to be read;
3. `format`, string, describes in which format `data` will be read. Acceptable values are “int8”, “byte”, “char” or “integer*1” for single byte integers; “int16”, “short”, “integer*2” for two-byte integers; “int32”, “integer*4” or just “int” for 4-byte integers; “float”, “real” or “real*4” for single precision real numbers; and, “double” or “real*8” for double precision real numbers.
4. (optional) `swap`, integer scalar 0 or 1, whether to swap byte ordering (little endian) or not (big endian).

Result: `x` numerical or string matrix.

Abstract: Read raw numerical or string data from file.

Generic Read of Text File

7.5.9 reads

I Format: `sx = reads()`

Result: `sx`, string.

Abstract: Collects an input from the keyboard until ‘return’ is pressed.

II Format: `sx = reads(filename, use_rows)`

Format: `sx = reads(filename, opts)`

Arguments:
7.5. READ/WRITE OPERATIONS

1 filename, string;
2 use_rows, integer vector, indices of rows in the file filename that are going to be read. If absent the entire file is read.

2 opts, list with entries opts =≺ skiprows; min_len; comment; note; use_rows; join_rows; join_csp; lstrip; s;
Brief descriptions of options follow.

* skiprows, integer, number of input lines to skip;
* min_len, integer, minimum length of input line for it to be processed;
* comment, string pattern, start of the string, the end of which is the end of string (or end of line), which is removed from the input line as a part of processing;
* note, string pattern, description of substring that is removed from input line as a part of processing. Please note, comment is a note which stretches to the end of the string entry;
* use_rows, integer array, indices of rows of input lines to be processed;
* join_rows, integer, number of consecutive input lines that are to be concatenated as a part of processing;
* join_csp, string, additional column separator inserted between joined input lines (default is blank space);
* lstrip, string pattern, to be removed from the input line if the line starts with it;
* start, string pattern, an identifier the input line has to contain for processing to start or to resume;
* stop, string pattern, an identifier the input line has to contain for processing to terminate. The processing may resume if another start pattern is found.
* grep, string pattern, process only rows that contain provided pattern.

Result: sx, string column-vector.

Abstract: Processes lines of text file according to provided options.

NIST’s MatrixMarket Format

7.5.10 readmm

Format: $x = \text{readmm}(fn)$

1 fn, string, name of a file that contains a single matrix in NIST’s MatrixMarket format.

Result: $x$, matrix read from the file fn.

7.5.11 writemm

Format: $\text{writemm}(fn, x)$

1 fn, string, name of a file to which the matrix will be written using NIST’s MatrixMarket format;
2 x, matrix.

Abstract: writes matrix $x$ to a file $fn$. 
HDF-specific Functions

7.5.12 h5ls

**Format:** \( s = h5ls(fn, loc, rec/) \)

1. \( fn \), string, name of a HDF file.
2. \( loc \), string, absolute address of the object in the HDF file \( fn \).
3. \( rec \), integer 0 or 1, performs recursive search within the object (if it is a group) and returns names of all the objects it finds, but only if \( loc \) ends with an “/”, see below.

**Result:** \( s \), two column string matrix where the first column gives the name of the object, while the second gives its type. Available types are “DATASET”, “GROUP”, “DATATYPE”, and “UNKNOWN”.

If \( loc \) is a group, then the result depends on how \( loc \) ends: If it ends with “/” then it returns in the first column of \( s \) the names of the members of the group, while in the second their type.

E.g., see directory test/hierarhical_data_format,

\[
\begin{align*}
>> & h5ls("./h5ex_g_iterate.h5","/") \\
& ans = \\
& /DS1 DATASET \\
& /DT1 DATATYPE \\
& /G1 GROUP \\
& /L1 DATASET \\
>> & h5ls("./h5ex_g_iterate.h5","/",1) \\
& ans = \\
& /DS1 DATASET \\
& /DT1 DATATYPE \\
& /G1 GROUP \\
& /G1/DS2 DATASET \\
& /L1 DATASET \\
>> & h5ls("./h5ex_g_iterate.h5","/G1") \\
& ans = \\
& /G1 GROUP \\
>> & h5ls("./h5ex_g_iterate.h5","/G1/") \\
& ans = \\
& /G1/DS2 DATASET \\
>>
\]

7.5.13 h5mv

I **Format:** \( s = h5mv(fn, src/, dest/) \)

II **Format:** \( s = h5mv(srcfn, src, destfn, dest) \)

**Arguments:**

1. \( srcfn \), string, name of the HDF file from which the object is moved.
2. \( src \), string, absolute address of an object in the HDF file \( srcfn \).
3 destfn, string, name of the HDF file to which the object is moved (if omitted then moving occurs in the same file srcfn).

3.4 dest, string, absolute address of an object in the destination HDF file destfn: it should either be a group, or not exist.

Abstract: Moves object src to location dest within a single HDF5 file. If dest is not given, then the object is un-linked (in that case one should still run utility 'h5repack' on the file to remove all non-linked objects; or load the file in rlab and then save it).

On return: s = 0 if no error occurred; s = 1 for errors in the first argument (e.g., improper file name); s = 2, for errors in the second argument (e.g., not string, or src cannot be “/”); and same for s = 3.

7.5.14 h5cp

I Format: s = h5cp(srcfn, src, dest)

II Format: s = h5cp(srcfn, src, destfn, dest)

Arguments:

1 srcfn, string, name of the HDF file from which the object is copied.

2 src, string, absolute address of an object in the HDF file srcfn.

3 destfn, string, name of the HDF file to which the object is copied (if omitted then assumed equal to srcfn).

3.4 dest, string, absolute address of an object in the destination HDF file: it should either be a group, or not exist.

Abstract: Copies object src to location dest within a single HDF5 file srcfn, or from srcfn to destfn. The copying cannot complete if dest is not a group.

On return: s = 0 if no error occurred; s = 1 for errors in the first argument (e.g., improper file name); s = 2, for errors in the second argument (e.g., not string, or src cannot be “/”); and same for s = 3.

7.5.15 h5ln

I Format: s = h5ln(targetfn, target, lnk)

II Format: s = h5ln(targetfn, target, lnkfn, lnk)

Arguments:

1 targetfn, string, name of the HDF file which is the target for the linking.

2 target, string, absolute address of the target object in the HDF file targetfn.

3 lnkfn, string, name of the HDF file which will contain the link (if omitted then assumed equal to targetfn).

3.4 lnk, string, absolute address of the link in the destination HDF file lnkfn: it should either be a group, or not exist.
Abstract: Creates a soft link from the target in the HDF file target fn, to the object lnk in the HDF5 file lnk fn. The copying cannot complete if dest is not a group.
On return: $s = 0$ if no error occurred; $s = 1$ for errors in the first argument (e.g., improper file name); $s = 2$, for errors in the second argument (e.g., not string, or lnk cannot be “/”); and same for $s = 3$. 
Example: Communication with Keithley instrument to retrieve the measured values (VDC) at intermediate rate.

```c
// eg_keithley.r: short the terminals for voltage measurement

// initialize serial port for communication with Keithley Integra 2700:
// 19200 baud rate, 8N1 with software control, leave default raw communication,
// and write all communication from the port as characters (debug)
serial = "serial:///dev/ttyS0";
seropt = <<>>;
seropt.data_parity_stop = '8N1';
seropt.speed = 19200;
seropt.flow_control = 'xon|xoff';
open(serial, seropt);
sleep (0.5);

// initialize Keithley:
1: SYST:PRES 'Continuous measurement mode (INIT:CONT ON)'
2: FUNC 'VOLT:DC' 'Select DCV function'
3: VOLT:DC:NPLC 1 'Set medium read rate (MED)'
WARNING: setting high read rate will clog the port and the rlab will not be able
to read from port again.
To regain the control over serial port this is what I have to do
on my laptop:
1. logout
2. login on tty console as root (ctrl+alt+f1)
3. init 3
4. setserial start (if this returns green, you are fine, red is trouble)
5. init 5
6. logout, and try to login at graphical (ctrl+alt+f7)
7. open konsole as a superuser and check whether minicom starts properly. If so, you are lucky, reset the serial port a few times.
Successful reset will be announced by the instrument beeping.

initcode = ["SYST:PRES\r", "FUNC 'VOLT:DC'\r", "VOLT:DC:NPLC 1\r"];
readcode = ["SENS:DATA:FRESH?\r"];

// number of readings
N = 100;

// data list for plotting
mydata = <<>>;
```
// use for storing the numerical values
x = zeros(N,2);
writem(serial, initcode);
sleep(0.5);

# To a query 'readcode' Keithley replies with
# -1.10276960E-06VDC,+6606.482SECS,+66586RDNG#
# this is to be extracted into a matrix with columns
# [time(sec), voltage (uV)]

// read N data points through internal loop
r = readm(serial, readcode, N);

if (any(strlen(r)==0))
{ stop("Empty response from the port\n"); }

// determine the format of the output. See note above
datarange = [1:strindex(r[1],"VDC")-1];
timerange = [strindex(r[1],"VDC")+4:strindex(r[1],"SECS")-1];

for (i in 1:r.nr)
{
    x[i;1] = strtod( substr(r[i], timerange) );
    x[i;2] = 1e6 * strtod( substr(r[i], datarange) );
}
mydata.a = x;

// single query reading within rlab loop
for (i in 1:N)
{
    r = readm(serial, readcode);
    x[i;1] = strtod( substr(r, timerange) );
    x[i;2] = 1e6 * strtod( substr(r, datarange) );
}
mydata.b = x;

// read/write query within rlab loop
for (i in 1:N)
{
    spinner();
}
7.5. READ/WRITE OPERATIONS

94 writem(serial, readcode);
95 r = readm(serial);
96 x[i;1] = strtod( substr(r, timerange) );
97 x[i;2] = 1e6 * strtod( substr(r, datarange) );
98 }
99 mydata.c = x;
100
101 // plot
102 xlabel ( "Instrument Time (sec)" );
103 ylabel ( "Instrument Reading (\gmV)" );
104 plegend( ["N queries", "1 query", "write/read"] );
105 plot ( mydata );
106 // reading times difference internal/external
107 ta = last(mydata.a)[1] - mydata.a[1;1];
108 tb = last(mydata.b)[1] - mydata.b[1;1];
109 tc = last(mydata.c)[1] - mydata.c[1;1];
110 printf("Times for %g measuremets\n", N);
111 printf("\tquery length N : %g sec\n", ta);
112 printf("\tquery length 1 : %g sec\n", tb);
113 printf("\twritem/readm : %g sec\n", tc);
Example: Obtaining historical quotes for Pfeizer, Inc. (NYSE: PFE) from Yahoo, Inc. web site.

```r
//
// file: eq1_http.r
//

// web page: needs extensive html postprocessing
url1 = "http://finance.yahoo.com/q/hp?s=PFE&a=00&b=4&c=1982&d=00&e=10&f=2010&g=d";

// csv file: needs minimal (coma, newline) postprocessing
url2 = "http://ichart.finance.yahoo.com/table.csv?s=PFE&a=00&b=4&c=1982&d=00&e=10&f=2010&g=d&ignore=.csv"

open(url1);
x1 = readm(url1);
close (url1);
open(url2);
x2 = readm(url2);
close (url2);

x1
size(x1)
colors("red");
printf("Knowledge is power\n");
pause()
colors()

printf("Similar, but more intelligible, request:\n");
pause()
x2
size(x2)
```
**Example:** Given a LabView© generated file, call it `testme.lvm`, the content of which is

```plaintext
LabVIEW Measurement
Writer_Version 0.92
Reader_Version 1
Separator Tab
Multi_Headings No
X_Columns One
Time_Pref Relative
Operator Administrator
Date 1903/12/31
Time 20:00:00
***End_of_Header***

Channels 3
Samples 5000 5000 5000
Date 2006/08/17 2006/08/17 2006/08/17
Time 23:32:01.914 23:32:01.914 23:32:01.914
X_Dimension Time Time Time
X0 0.0000000000000000E+0 0.0000000000000000E+0 0.0000000000000000E+0
Delta_X 1.000000 1.000000 1.000000
***End_of_Header***
X_Value Comment
0.000000,-5.000000E-6:-6.250000E-5,-0.003126
1.000000:-4.998000E-6,-6.250000E-5,-0.403254
2.000000,-4.996000E-6,-6.250000E-5,-0.003126
3.000000,-4.994000E-6,-6.250000E-5 -0.403254
```

**Note:** Please note that the file has been modified where space separators were replaced by commas and columns quasi-randomly.

**Example:** Let us use `reads` to obtain the header of the file without loading the whole file. *A priori* inspection of the file shows that the lines 1-21 contain miscellaneous information regarding the measurement.

```plaintext
>> reads("testme.lvm",1:21)
LabVIEW Measurement
Writer_Version 0.92
Reader_Version 1
Separator Tab
Multi_Headings No
X_Columns One

....

X_Value Comment
```

>>
Example: Now let us read the content of the file that interests us, that would be, the lines 22 on using `readm`.

```matlab
>> x = readm("testme.lvm",21)
matrix columns 1 thru 4
    0   -5e-06   -6.25e-05    -0.003126
    1  -4.998e-06   -6.25e-05    -0.403254
    2  -4.996e-06   -6.25e-05    -0.003126
    3  -4.994e-06   -6.25e-05    -0.403254

>>
```
Chapter 8

Advanced General Numerical Procedures and Algorithms
8.1 Histograms in 1-D and 2-D

\texttt{rlab} supports histograms as special lists, where 1-D histogram is \texttt{\langle bin; range \rangle}, while 2-D histogram is \texttt{\langle bin; xrange; yrange \rangle}. These definitions are derived from the Gnu Science Library histogram specifications.

// gsl's histogram
// outline:
//  |----------|----------|-------- ... --|----------|

8.1.1 hist

I Format: \( h = \text{hist}(data) \),

II Format: \( h = \text{hist}(data, n) \),

III Format: \( h = \text{hist}(data, [xmin, xmax]/, n/) \),

IV Format: \( h = \text{hist}(data, range) \)

V Format: \( \text{hist}(h, data) \)

Arguments:

1 \( data \), real vector, data to be histogrammed;

2,3 \( n \), integer, number of bins. If not given, a default \( n = 10 \) is used;

2 \( [mind, maxd] \), two component vector, lower and upper bound for the bins. If not given then \( mind = \min\{data_i\} \) and \( maxd = \max\{data_i\} \) are used;

2 \( range \), real vector, the ranges of respective bins.

Result: \( h = \langle bin; range; pos_inf; neg_inf; nan; super_max; sub_min \rangle \), where

- \( bin \), real vector of size \( n \), count for each bin;

- \( range \), real vector of size \( n + 1 \), the bounds of the bins.

In addition, \texttt{rlab} provides five special bins:

- \( pos_inf \), non-negative integer, count of how many \(+\infty\) occurred in \( x \);

- \( neg_inf \), non-negative integer, count of how many \(-\infty\) occurred in \( x \);

- \( nan \), non-negative integer, count of how many NaN's occurred in \( x \);

- \( super_max \), non-negative integer, count of how many entries in \( x \) are above the right boundary of the highest bin but not \(+\infty\);

- \( sub_min \), non-negative integer, count of how many entries in \( x \) are below the left boundary of the lowest bin but not \(-\infty\);
Note: The entry V in the list of formats above shows how new data can be added to an existing histogram.

Note: Plotting functions plot, from pgplot, and gnuplot, from gnuplot, take histogrammic variable as an argument or as an entry in their argument list. Thus, to plot histograms one need not use plhist function.

Note: Statistical functions, such as mean, variance, & c. can take histogrammatic variable as an argument.

8.1.2 hist2

I Format: \( h = \text{hist2}(data) \),

II Format: \( h = \text{hist2}(data, nx, ...) \),

III Format: \( h = \text{hist2}(data, [xmin, xmax]/, nx/, ...) \),

IV Format: \( h = \text{hist2}(data, rangex, ...) \)

Arguments:

1. data, real two-column matrix, data in form \([x_i, y_i]_i\) to be histogrammed in two dimensions;
2.3 nx, integer, number of bins for x-range. If not given, a default \( nx = 10 \) is used;
2. [xmin, xmax], two component vector, lower and upper bound for the bins in x-direction. If not given then \( xmin = \min{\{x_i\}} \) and \( xmax = \max{\{data_i\}} \) are used;
2. rangex, real vector, the ranges of respective bins in x-direction.

... the bins along y-direction are specified in the same way as in the x-direction using one of the four listed forms.

Result: \( h = << \text{bin; xrange; yrange; data >>} \), a list the elements of which are

- bin, real vector of size \( n \)-by-\( m \), count for each bin in x- and y-direction;
- xrange, real vector of size \( n + 1 \), the bounds of the bins along x-direction.
- yrange, real vector of size \( m + 1 \), the bounds of the bins along y-direction.
- data, real 2-column matrix vector, original data.

Bin \( bin[i; j] \) is bounded along x-direction from below by \( xrange[i] \) and from above by \( xrange[i + 1] \), while along y-direction from below by \( yrange[j] \) and from above by \( yrange[j + 1] \).

Note: Plotting function gnusplot, from gnuplot, takes histogrammic variable as an argument or as an entry in their argument list. Thus, one can plot histograms on the same plot as any other object gnuplot can plot. There are two poorly documented functions that help in plotting: hist_line, and hist_norm. The first determines whether the data set will be plotted as a line or a bar graph (see gnudefault), while the second determines whether the histogram will be normalized to unity (integral over the x-axis results in 1), or to the size of sample. hist_norm(i) with \( i = 1 \) selects normalized histograms, while \( i = 0 \) selects non-normalized.
8.2 Cubic spline and linear interpolation

8.2.1 csplinterp

Format: \( m = \text{csplinterp}(y, x, y', y'_f) \),

Arguments:

1. \( y \), vector, ordinatae of the points for spline interpolation;
2. \( x \), vector, abscisae of the points for spline interpolation;
3,4. \( y'_i, y'_f \), scalars, first derivatives at the end points of the interpolation interval.

Result: \( m \), matrix, three-columns, \( m = [x_i, y_i, y''(x_i)] \).

Abstract: The function finds the second derivatives for the cubic spline interpolation through the points \( [x_i, y_i] \), as follows:

- If \( y'_i \) and \( y'_f \) are given, and \( y'_i \neq y'_f \) or \( y_i \neq y_f \), the function finds standard cubic spline.
- Periodic splines are found if both, \( y'_i = y'_f \) and \( y_i = y_f \) are satisfied.
- If neither \( y'_i \) nor \( y'_f \) are given, the second derivatives are calculated using, so called, natural boundary conditions, in which the second derivatives at the end of the interpolating intervals are set to zero \textit{a priori}.

Note: Here, it is assumed, but it is not checked, that the first column of the data matrix is sorted in ascending order.

8.2.2 cspleval

Format: \( y = \text{cspleval}(x, M) \)

Arguments:

1. \( x \), vector;
2. \( M \), matrix, obtained by previous call to \textit{csplinefit}.

Result: \( y \), a vector of the size of \( x \).

Abstract: The function calculates the cubic spline interpolation for the values of \( x \), using the tabulated values of the function and the previously calculated values of the second derivatives \( M \).

Note: Here it is assumed, but it is not checked, that (i) \( x \) is sorted in ascending order, and (ii) that the values of \( x \) are within the bounds of the original data set as given in \( M \).

8.2.3 linterp

Format: \( v = \text{linterp}(u, M, /\text{order}/) \)

Arguments:

1. \( u \), real vector;
2 $M$, real 2-column matrix;
3 $\textit{order}$, integer, 0 or 1 (default).

Result: $v$, real vector of the size of $u$.

Abstract: The function interpolates the values given in $u$, using the interpolation table in $M$, depending on the $\textit{order}$. $M$ contains the rows of pairs of $\{(x_i, y_i)\}_{i=1,N}$ that are used in interpolation, which are sorted with respect to the first column, $x_i \leq x_j$ for $i \leq j$.

For $\textit{order}=1$, it finds a piece-wise linear interpolant, which passes through the points given in $M$. For $u$’s outside the interval $[x_1, x_N]$ the function performs a linear extrapolation based on the slopes at the two end-points.

For $\textit{order}=0$, it finds a piece-wise constant interpolant continuous in the end-points,

$$x_i < u_j < x_{i+1} \rightarrow v_j = y_i,$$

while, at the end-points (for $u_j = x_i$)

$$v_j = \frac{1}{2}(y_{i-1} + y_i),$$

where we assume $x_0 = -\infty$ and $y_0 = y_1$, and $x_{N+1} = +\infty$ and $y_{N+1} = y_N$. The latter are also used for extrapolation outside the interval $[x_1, x_N]$.

Note: The function assumes that $u$ is sorted in ascending order.

Note: For $\textit{order}=1$, it is possible to have discontinuities in the interpolation table $M$, by listing the discontinuity point twice, once for each limit (left and right, in that order). Consider the following example:

```matlab
>> x = [0,1,1,2]';
>> y = [1,1,2,2]';
>> m = [x,y]
   0  1
   1  1
   1  2
   2  2
>> linterp(1, m)
   1.5
```

In the example above the discontinuity occurs for $x = 1$, and the left limit is $y_L(1) = 1$ while the right limit is $y_R(1) = 2$. Observe that the value of the function at discontinuity is the average of the left and right limits.

Note: This function appeared in rlab/toolkit in two forms, lintrp and lintrp2. Here, lininterp includes both and is written in C instead of in rlab script.

8.2.4 \textit{linterp1}

Format: $y = linterp1(x, M)$

Arguments:
1 $x$, a real dense column vector;
2 \( M \), a real dense matrix.

**Result:** \( y \), a real dense vector of the size of \( x \).

**Abstract:** Just as \( linterp() \) the function \( linterp1() \) calculates the linear interpolation for values in the array \( x \), using the tabulated values given in matrix \( M \). However, here it is assumed that the ordinates of points in matrix \( M \) are equidistant. For the same reason, \( linterp1() \) does not deal with the discontinuities in the linear interpolating table \( M \).

### 8.2.5 harmsum

**Format:** \( y = harmsum(t, M) \)

**Arguments:**

1. \( t \), real vector;
2. \( M \), real 3- or 4-column matrix, \( M = \begin{bmatrix} \Omega/ & \lambda/ & A & \phi \end{bmatrix} \).

**Result:** \( y \), real vector of the size of \( t \).

**Abstract:** The function finds the sum

\[
y = \sum_i A_i \cdot e^{-\lambda t} \sin(\Omega t + \phi).
\]  

If \( \lambda \) is not given then \( \lambda \equiv 0 \) is assumed.
Example: Comparison of point-wise linear and constant interpolation over the same interpolation table. Its result is shown in Fig. [8.1], on p. 202

```plaintext
1 clearall();
2
3 gnuwins (1);  
4
5 // interpolation table
6 m = [ ... 
7 0, 0 ;...
8 1, 3 ;...
9 2, 1 ;...
10 3, 4 ...
11 ];
12
13 x = [-1:4:1/64]';
14
15 y1a = linterp(x, m, 0);  
16 y1b = linterp(m[:,1], m, 0);  
17 y2a = linterp(x, m);  
18 y2b = linterp(m[:,1], m);
19
20 // plot this
21 //
22 //
23 gnuwin (1);  
24 gnuulimits (-1,4,-3,8);  
25 gnuxtics (1);  
26 gnuytics (1);  
27 gnulegend ([ ... 
28 "table of [x,y]" ,...
29 "piece-wise constant and continuous at end-points: overall" ,...
30 "at end-points" ,...
31 "piece-wise linear and continuous: overall" ,...
32 "at end-points" ...
33 ]);  
34 gnuformat ([ ... 
35 "with points ps 5 pt 4 lc @red" ,...
36 "with lines lc @blue" ,...
37 "with points ps 3 pt 4 lc @blue" ,...
38 "with lines lc @green" ,...
39 "with points ps 3 pt 4 lc @green" ...
40 ]);  
41 gnuplot( <<a=m; b=[x,y1a]; c=[m[:,1],y1b]; d=[x,y2a]; e=[m[:,1],y2b] >>, "linterp.eps" );
```
Figure 8.1: Comparison of a point-wise linear and a constant interpolation over the same interpolation table.
8.3 Wavelet Transforms in one and two dimensions

8.3.1 dwt

Format: \( w = dwt(x, opts) \),

Arguments:

1. \( x \), real vector, equidistantly sampled array of data. Its length has to be some power of 2;

2. \( opts = << family; index >> \), list with entries:

   - \( family \), string, the name of wavelet family, which can be “daub”, “daub_c”, “haar”, “haar_c”, “bspline” and “bspline_c”, for implemented Daubechies, Haar and B-spline wavelet families and their centered versions;

   - \( index \), integer or pair of integers, index of the member in the wavelet family, where for “daub” and “daub_c” \( index = 4, 6, \ldots, 20 \), even values only, for “haar” and “haar_c” \( index = 2 \), while for “bspline” and “bspline_c” index is either a pair of integers \([i, j]\), where \( i = 1(j = 3, 5) \), \( i = 2 \), \( (j = 2, 4, 6, 8) \) or \( i = 3(j = 1, 3, 5, 7, 9) \), or an integer \( index = 100 * i + j \);

Result: \( w \), real vector, wavelet transform of \( x \).

8.3.2 idwt

\( x = idwt(w, opts) \),

Arguments:

1. \( w \), real vector, presumable wavelet transform of equidistantly sampled array of data. Its length has to be some power of 2;

2. \( opts = << family; index >> \), same as above.

Result: \( x \), real vector, inverse wavelet transform of \( w \).
8.3.3  \texttt{dwt2}

\textbf{Format:}  \( w = \texttt{dwt}(x, \texttt{opts}) \),

\textbf{Arguments:}

1. \( x \), real square matrix, sampled equidistantly in two-dimensions. Its size has to be some power of 2;

2. \( \texttt{opts} = \langle< \texttt{family}; \texttt{index}; \texttt{proc\_standard} >> \), list with entries:
   
   - \texttt{family}, string, the name of wavelet family, which can be “daub”, “daub\_c”, “haar”, “haar\_c”, “bspline” and “bspline\_c”, for implemented Daubechies, Haar and B-spline wavelet families and their centered versions;
   
   - \texttt{index}, integer or pair of integers, index of the member in the wavelet family, where for “daub” and “daub\_c” \( \texttt{index} = 4, 6, \ldots 20 \), even values only, for “haar” and “haar\_c” \( \texttt{index} = 2 \), while for “bspline” and “bspline\_c” index is either a pair of integers \([i, j]\), where \( i = 1(j = 3, 5) \), \( i = 2, (j = 2, 4, 6, 8) \) or \( i = 3(j = 1, 3, 5, 7, 9) \), or an integer \( \texttt{index} = 100 \ast i + j \);
   
   - \texttt{proc\_standard}, integer 0 (non-standard transform) or 1 (default, standard transform).

\textbf{Result:}  \( w \), real matrix, wavelet transform of matrix \( x \).

8.3.4  \texttt{idwt2}

\( x = \texttt{idwt}(w, \texttt{opts}) \),

\textbf{Arguments:}

1. \( w \), real matrix, presumable wavelet transform of a matrix sampled equidistantly in two-dimensions. Its size has to be some power of 2;

2. \( \texttt{opts} = \langle< \texttt{family}; \texttt{index}; \texttt{proc\_standard} >> \), same as above.

\textbf{Result:}  \( x \), real matrix, inverse wavelet transform of \( w \).
8.4 Roots of a real vector function

**Root of a real function of a real variable**

The solvers in this class solve the equation

\[ f(x, p) = rhs, \]  

(8.4)

where \( f \) is a real scalar function, \( rhs \) is a real scalar, while \( x \) is a real scalar variable.

8.4.1 *findroot*

- **Format:** \( x = \text{findroot}(X, rhs) \)
  
  **Arguments:**
  
  1. \( X = [x_i, y_i] \), real two-column real matrix, a tabular representation of the function \( f(x) \);
  2. \( rhs \), real scalar, the right hand side of Eq. (8.4) (default value 0);

  **Result:** \( x \), real vector, the roots of Eq. (8.4).

- **Format:** \( x = \text{findroot}(f, /p/, X, \text{options}) \)
  
  **Arguments:**
  
  1. \( f = \text{function}(x, p) \);
  2. \( p \), the parameter passed directly to the function;
  3. \( X = [xlo, xhi] \), the interval containing the root;
  4. \( \text{options} << \text{imethod}; eabs; erel; maxi; rhs >>> \). Here \( \text{imethod} \) determines the bracketing method, where
     
     * \( \text{imethod} = 0 \), bisection,
     * \( \text{imethod} = 1 \), regula falsi,
     * \( \text{imethod} = 2 \), Brent-Dekker, (default).

     while \( eabs \) and \( erel \) determine the accuracy goal

     \[ |x_{j+1} - x_j| < eabs + erel \cdot x_{j+1}, \]  

(8.5)

    to be reached in at most \( maxi \) iterations.

  **Result:** \( x \), real scalar, a root of Eq. (8.4).

- **Format:** \( x = \text{findroot}(f, dfdx, /p/, x0, \text{options}) \)
  
  **Arguments:**
  
  1.2. \( f = \text{function}(x, p) \), its jacobian \( dfdx = \text{function}(x, p) \);
  3. \( p \), the parameter passed directly to the functions;
  4. \( x0 \), initial guess for the root;
  5. \( \text{options} << \text{imethod}; eabs; erel; maxi; rhs >>> \), where \( \text{imethod} \) determines the method used,
* $i\text{method} = 0$, Newton, (default),
* $i\text{method} = 1$, Secant,
* $i\text{method} = 2$, Steffenson.

The parameters $e\text{abs}$ and $e\text{rel}$ determine the accuracy goal, while $\text{maxi}$ gives the maximal number of iterations to achieve it. A criterion for root $x = x_{j+1}$ to be returned is given in Eq. (8.5).

**Result:** $x$, real vector, a root of Eq. (8.4).
Finding a root of a real vector function of a real vector variable

8.4.2 findroots

- **Format:** \( x = \text{findroots}(f,/p/,x_0,/options/) \),

**Arguments:**

1. \( f = function(x/,p/) \), a vector function representing the root finding problem

\[
f_i(x/,p/) = 0, \quad \text{for} \ i = 1, \ldots \dim f. \tag{8.6}
\]

2. \( p \), the parameter passed directly to the function;

3. \( x_0 \), a real vector, \( \dim x_0 = \dim f \), initial guess for the root;

4. \( \text{options} = << \text{imethod}; eabs; maxi; stdout >> \). Here, \text{imethod} determines the method being used in finding the root, where

   * \text{imethod} = 0, GSL "hybrids" routine, for Hybrid algorithm, replaces Jacobian by its finite difference approximation, (default),
   * \text{imethod} = 1, GSL "hybrid", same as Hybrid algorithm, without internal scaling, though.
   * \text{imethod} = 2, GSL "newton", Newton-type iteration with finite difference approximation to Jacobian,
   * \text{imethod} = 3, GSL "broyden" algorithm, cheap approximates to the Jacobian matrix at each update.

The parameter \( eabs \) is the absolute error for convergence criterion,

\[
|x_{j+1} - x_j| < eabs, \tag{8.7}
\]

while \( maxi \) gives the maximum number of iterations to satisfy the convergence. The progress of calculation is posted on the file/device \( stdout \).

**Result:** a real vector \( x \) if the solution is found, or an empty vector if it is not.

- **Format:** \( x = \text{findroots}(f,df,/p/,x_0,/options/) \),

**Arguments:**

1.2. \( f = function(x/,p/) \) and its jacobian \( df = function(x/,p/) \);

3. \( p \), the parameter passed directly to the functions;

4. \( x_0 \), a real vector, \( \dim x_0 = \dim f \), initial guess for the root;

5. The entries in the list \text{options} depend on the choice of the group of methods, where

   I. The GSL, for which \( \text{options} = << \text{imethod}; eabs; maxi; stdout >> \), and \text{imethod} = 0, 1, 2, 3 for Powell’s method as implemented in MINPACK, with and without internal scaling, Newton method and the Newton method with global convergence.

   II. The HOMPACK, for which \( \text{options} = << \text{imethod}; \ldots >> \), and where \text{imethod} = 4, 5, 6 for PDF, PNF and PQF homothopy methods based on ODE, normal flow, and augmented Jacobian approach. Here, the list \text{options} depends on the solver and accepts the following entries,


· *curs, irep* (all solvers). Curvature of the arc (Solvers PDF and PNF). Default value 10. Number of repetitions in trying to achieve a convergence. It is meaningful to increase it if the error message is “not converged yet”. Default value 1.

· *arcre, ansre* (all solvers). The relative arc tolerance and the relative change in the solution.

· *arcae, ansae* (PNF, PQF). The absolute errors in arc length, and in the position of the root.

· *faca* (PDF). Upper limit on the length of the arc before the iteration is restarted. Default value 1 (corresponding to max arclength of $faca \cdot \sqrt{neq}$, where $neq$ is the number of equations).

**Result:** a real vector $x$ if successful, or an empty array, if not.

**Abstract:** With this combination of parameters, the solver *findroots* solves a vector equation $f(x, p) = 0$, using the jacobian of the function $f$, $df = \partial f / \partial x$.

**Note:** Homothopy methods are not general class methods for finding roots. Use them with care as the convergence is not guaranteed for every starting point. They are typically slower and very sensitive on control parameters. Check the comment lines in the fortran codes in *flibs/hompack/src*, driver files *FIXPDF.f*, *FIXPQF.f* and *FIXPNF.f*.

**Note:** (HOMPACK) The publicly available code was modified to allow for greater control of the parameters governing the solvers.

### 8.4.3 crvtrack

**Format:** $x1 = crvtrack(f, df, /params/, x0/, options/)$.  

**Arguments:**

1. $f = function(x, c, /params/)$, vector function of vector variable;

2. $df = function(x, c, /params/) = [df/dx, df/dc]$, matrix containing the extended jacobian of $f$;

3. $params$, optional parameter for user functions $f$ and $df$.

4. $x0$, a real vector which satisfies,  

$$f(x0, 0/, params/) = 0; \quad (8.8)$$

5. $options =<< imethod; \ldots >>$, the entries in the list depend on the method chosen for curve tracking, where

   - $imethod = 1$ (default) is for CONTIN, for which $options =<< ansre; ansae; faca; stdout >>$, as discussed above.

   - and $imethod = 4, 5, 6$ for HOMPACK PDF, PNF and PQF, in that order. See above for options for HOMPACK solvers and their interpretations.

**Result:** real vector $x1$, a solution of $f(x1, 1/, params/) = 0$.  


Abstract: The curve tracking program based on the homotopy algorithm, as implemented in HOMPACK, and on continuation code CONTIN by Rheinboldt and Burkardt [1991]. The code tracks a root of a non-linear equation $f(x, c) = 0$ from $x = x_0$ for $c = 0$ to some $x$ at $c = 1$, if such exists.

Note: See the example code `optrootfit/eg_c trx2.r` for comparison between the CONTIN and HOMPACK. The example is taken from CONTIN suite. I can’t say that HOMPACK solvers passes that particular example (following a root of the Freudenstein-Roth function): it losses it at half-a-way.
Example: A multidimensional root finder without derivatives:

```plaintext
format(9,9);
F = function(x, p)
options=<>>;
options.eabs = 1e-6;
options.stdout = 'test.dat'
p = [1,10];
x0=[-5;5];
for(j in 0:3)
{
options.imethod = j;
x=findroots(F, p, x0, options);
}
```

This code produces the following output (cat of the file “test.dat”):

```plaintext
RLaB: using gsl 'hybrids' solver for multidimensional root finding.
[ SNIP ]
RLaB gsl 'hybrids' solver for multidimensional root finding reports: 'success'!
RLaB: root finding lasted 0 sec.
RLaB: using gsl 'hybrid' solver for multidimensional root finding.
[ SNIP ]
RLaB gsl 'hybrid' solver for multidimensional root finding reports: 'success'!
RLaB: root finding lasted 0 sec.
RLaB: using gsl 'dnewton' solver for multidimensional root finding.
RLaB gsl 'dnewton' solver for multidimensional root finding reports: 'success'!
RLaB: root finding lasted 0 sec.
RLaB: using gsl 'broyden' solver for multidimensional root finding.
[ SNIP ]
RLaB gsl 'broyden' solver for multidimensional root finding reports: 'success'!
RLaB: root finding lasted 0 sec.
```
8.4. ROOTS OF A REAL VECTOR FUNCTION

Example: A multidimensional root finder with Jacobian:

```plaintext
// function
F = function(x, p)

// its Jacobian
dF = function(x, p)
{
  J = zeros(x.nr, x.nr);
  J[1;1] = -p[1];
  J[2;2] = p[2];
  return J;
}

options=<<>>;
options.eabs = 1e-6;
options.maxir = 1000;
options.stdout = 'test.dat'

p = [1,10];
x0=[-5;5];
for(j in 0:3)
{
  options.imethod = j;
  x=findroots(F, dF, p, x0, options);
}
```

It produces the following output (cat of the "test.dat" file):

```
RLaB: using gsl 'hybridsj' solver for multidimensional root finding.
[ SNIP ]
RLaB gsl 'hybridsj' solver for multidimensional root finding reports: 'success'!
RLaB: root finding lasted 0 sec.
RLaB: using gsl 'hybridj' solver for multidimensional root finding.
[ SNIP ]
RLaB gsl 'hybridj' solver for multidimensional root finding reports: 'success'!
RLaB: root finding lasted 0 sec.
RLaB: using gsl 'newton' solver for multidimensional root finding.
RLaB gsl 'newton' solver for multidimensional root finding reports: 'success'!
RLaB: root finding lasted 0 sec.
```
RLaB: using gsl 'gnewton' solver for multidimensional root finding.
RLaB gsl 'gnewton' solver for multidimensional root finding reports: 'success'!
RLaB: root finding lasted 0 sec.

Note: If using file output for HOMPACK solvers, there will be some mess-up regarding the printout of start/stop statements. This is because the same stream is used by both c and FORTRAN code (which is against any recommendation regarding mixing of c and FORTRAN). When dumping messages on the another terminal, there is no such problem.
8.5 Simulated annealing

For solving simulated annealing problems there are two solvers available: the basic one that is implemented in the GSL, and a more advanced one, adaptive simulated annealing (asa) code by L. Ingber [Ingber 1993].

8.5.1 siman

Format: \( x_1 = \text{siman}(\text{cost}, \text{gendist}, x_0, \text{opt}) \)

Arguments:

1. \( \text{cost} = \text{function}(x) \), cost associated with a configuration \( x \);
2. \( \text{gendist} = \text{function}(x) \), given a current configuration \( x \) it returns a new configuration;
3. \( x_0 \), real vector, initial configuration of the system;
4. \( \text{opt} = << \text{stdout}; nT; Ti; mu; Tf >> \), list with entries:
   - \( nT \), integer, determines how many attempts to improve configuration will occur at a particular “temperature”;
   - \( Ti, Tf \), scalar each, initial and final “temperature”;
   - \( mu \), positive scalar, cooling rate;
   - \( \text{stdout} \), terminal to which the run-time messages are posted

Result: \( x_1 \), real vector, possibly improved configuration of the system.

Abstract: \( \text{siman} \) performs simulated annealing (cooling) of the configuration described by a real vector \( x \), using the cost function \( \text{cost}(x) \) and the way the configuration can be changed, as described by \( \text{gendist}(x) \). New configuration is accepted if it either decreases the cost, or if it is probable with probability

\[
p = \exp \left( - \frac{\text{curr\_cost} - \text{last\_cost}}{T_{\text{cost}}} \right), \tag{8.9}
\]

known as the Boltzmann acceptance test. Here, for given \( x \) and \( x' = \text{gendist}(x) \), \( \text{curr\_cost} = \text{cost}(x') \) and \( \text{last\_cost} = \text{cost}(x) \).

Note: \( \text{siman} \) relies on the GSL random number generator, also controlled by user through \texttt{rlab}'s interface.
8.5.2 asamin

Format: \( w = \text{asamin}(\text{cost, /gendist/}, x0, b/, \text{accept, csched, opt/}) \)

Arguments:

1. \( \textit{cost} = \text{function}(x) \), cost associated with a configuration \( x \);

2. \( \textit{gendist} = \text{function}(r, x_i, Tx_i, i) \), generating distribution for \( x_i \), where \( i \) is the index of entries in vector \( x \), \( r \) is a uniform random number in \([0, 1]\), while \( Tx_i \) is "the temperature" (parameter) associated with distribution of particular \( x_i \). The function can be specified by the user, or explicitly omitted. If omitted, the default ASA generating distribution is used,

\[
f(r, x_i, Tx_i, i) = x_i + [r < 0.5? -1 : 1] \cdot \left( 1 + \frac{1}{Tx_i} \right)^{|2r-1|} - 1 \quad (8.10)
\]

3. \( x0 \), real vector, initial configuration;

4. \( b \), real 2-column matrix, bounds of \( x \), where \( b_{i,1} \leq x_i \leq b_{i,2} \);

5. \( \textit{accept} = \text{function}(\text{rnd, curr\_cost, last\_cost, Tcost}) \), acceptance test, returns 0 (curr\_cost is rejected) or 1 (curr\_cost and configuration that produced it are accepted), given a uniform random number in in \([0, 1]\), \( r \), current and last accepted cost, and the cost temperature, \( Tcost \). The function can be specified by the user or explicitly omitted, in which case it defaults to Boltzmann acceptance given by Eq. \((8.9)\);

6. \( \textit{csched} = \text{function}(Tcost) \), cost temperature scheduling function, gives user an opportunity to additionally modify the cost temperature. The function can be specified by the user or omitted, in which case it defaults to ASA default, an identity function;

7, 6, 5 opts, list, options and parameters for the solver that allow finer tuning of the behavior. Its entries are:

- \( \textit{param\_type} \) real vector of size \( x \), type of parameter of entries in \( x \): -1 or 1 for real, -2 or 2 for integer;

- \( \textit{init\_param\_temp} \) non-negative scalar, initial parameter temperature used for generating distribution function Eq. \((8.10)\);

- \( \textit{curvature} \), 0 or 1, specifies whether the matrix of second derivatives of the cost function with respect to the parameter \( x \) is required;

- \( dx \), positive scalar, used as a spacing for finite difference calculation of curvature and tangent of the cost function;

- \( \textit{seq} \), 0 or 1, specifies whether cooling will be done sequentially, over each of the entries in parameter \( x \), or simultaneously over all (default);

- \( \textit{acc\_freq\_mod, gen\_freq\_mod} \), positive integers, acceptance frequency modulus, and generating frequency modulus, see ASA manual;

- \( \textit{cost\_prec} \), positive scalar, precision in calculating the cost function;

- \( \textit{max\_cost\_repeat} \), positive integer, maximal number of repeating of cost function;

- \( \textit{no\_cost\_samples} \), positive integer, number of cost samples;

- \( \textit{temp\_ratio\_scale} \), positive scalar, cost temperature ratio scale;

- \( \textit{cost\_param\_scale\_ratio} \), positive scalar, ratio of cost parameter scale;
8.5. SIMULATED ANNEALING

*temp_ann_scale*, positive scalar, temperature annealing scale;
*limit_acc*, positive integer, acceptance limit;
*limit_gen*, positive integer, limit number of generated states;
*limit_inv_gen_states*, positive integer, limit number of invalid generated states;
*acc_to_gen_ratio*, positive scalar, accepted to generated ratio;
*stdout*, string, file or console to which the run-time messages are posted. The ASA code is compiled with maximum verbosity so if user does not specify *stdout*, the code uses /dev/null instead. Thus, if user does not require messaging, the code can be recompiled with messaging disabled. **define** options with which the code is compiled are given in Makefile.in with asamin.o. The recompilation sequence is thus: ./configure; rm asamin.o rlab; make rlab; (su) make install.

**Result:** *w*, list, result of asa calculation, with entries:

- *coef*, real vector, new values of parameter array *x* that (hopefully) decreases the cost function;
- *status*, integer, status of the solution upon completion of calculation;
- *tangent*, real vector, tangent of cost function for *x* given in *coef*;
- *curvature*, real matrix, curvature of cost function for *x* given in *coef*, if requested by user;
- *cost*, scalar, value of cost function at *x* given in *coef*.

**Abstract:** *asamin* performs adaptive simulated annealing using ASA code by [Ingber] (1993). Please, read the ASA manual if planning to use the code more seriously.
**Example:** A cheap variation of the travelling salesman problem (as compared to the version in the GSL manual, which contains the coordinates of actual North American cities).

```plaintext
haveplwin(2);

// number of cities
NCITIES = 10;
// put them on the circle
th = [0:2*pi:2*pi/NCITIES]';
dr = 4;
or0 = 3;
// now move them radially a bit
rad = or0 + dr*(2*rand(dth.nr,1)-ones(dth.nr,1));
x = [rad.*cos(th), rad.*sin(th)];
// make first and last the same
x[NCITIES+1] = x[1];
x0 = x;

// total distance for the salesman
energy = function(x)
{
    return sqrt(sum(sum((x[2:x.nr]-x[1:x.nr-1]).^2)));
};

// exchange two cities in range 2:x.nr-1
nextstep = function(x)
{
    i = 2 + int((x.nr-2)*rand(1,1));
j = 2 + int((x.nr-2)*rand(1,1));
    while(i == j)
    {
        j = 2+int((x.nr-2)*rand(1,1));
    }
c = x[i];
x[i] = x[j];
x[j] = c;
    return x;
};

// use nextstep to randomize the original arrangement of the cities
for(i in 1:1000)
{
    x = nextstep(x);
}

data=<<>>;
data.[1] = x0;
```
data.[2] = x;
plwin(1);
plstyle("line-point");
pltile("Travelling Salesman Problem");
plegend( ["original", "shuffled"] );
plot(data);

// do the simulated annealing until the improved configurations is obtained
// or one reaches maximum number of iterations, here 10.
i = 1;
options=<<>>;
options.stdout = rlab_initstderr();

while(energy(x)>energy(x0) && i<10)
{
    printf("Round No. %2i of annealing :\n",i++);
    x = siman(energy,nextstep,x,options);
    printf(" original length = %g\n annealed length = %g\n", energy(x0), energy(x));
    data.[1] = x0;
    data.[2] = x;
    plwin(2);
    plstyle("line-point");
    pltile("Travelling Salesman Annealed Solution");
    plegend( ["original", "annealed"] );
    plot(data);
}

printf("Length of the original configuration = %g\n", energy(x0));
printf("Length of the annealed solution = %g\n", energy(x));
**Example:** Consider minimization of the following cost function of \( x = [x_1, x_2] \),

\[
\text{cost}(x) = (x_1 - 1)^2 + (x_2 - 2)^2,
\]

for \( x_1 \in [0, 2] \) and \( x_2 \in [1, 3] \).

---

```r
// eg_asal.r: trivial minimization problem

// cost function

cost = function(x)
{
  rval = ((x[1]-1).^2 + (x[2]-2).^2);
  return rval;
}

// generating distribution for new x's
// we use original ASA distribution

gendist = function(r, xi, Ti, i)
{
  y = ifelse(r<0.5, -1, 1);
  z = y * Ti * ((1.0 + 1.0/Ti).^abs(2.0*r - 1.0) - 1.0);
  rval = xi + 2 * z;
  return rval;
}

// acceptance test

acctest = function(uniform_test, last_cost, curr_cost, curr_cost_temp)
{
  // use Boltzmann’s test
  delta_cost = (curr_cost - last_cost) / curr_cost_temp;
  x = min ( 1.0, exp(-delta_cost) );
  if (x >= uniform_test)
  {
    rval = 1;
  } else
  {
    rval = 0;
  }
  return rval;
}
```
8.5. SIMULATED ANNEALING

43 //
44 // cost scheduling function
45 //
46 costsched = function(cost_T)
47 {
48    rval = cost_T;
49    return rval;
50  };
51
52 // bounds of x
53 b =  [ 0, 2 ; ...  
54    1, 3];
55
56 // generate initial point
57 rng(1,"uniform", [0,2] );
58 x1 = rand();
59 rng(2,"uniform", [1,3] );
60 x2 = rand();
61 x0 = [x1,x2];
62
63 // options for asa solver: given are default values
64 aopts = <<>>;
65 aopts.stdout = stderr(); // write messages
66 aopts.curvature = 1; // find curvature of cost function, cost''
67 aopts.init_param_temp = 3; // initial parameter temperature
68 aopts.seq = 0; // sequential cooling of the parameters
69 aopts.acc_freq_mod = 100; // acceptance frequency modulus (rtm)
70 aopts.gen_freq_mod = 10000; // generated frequency modulus (rtm)
71 aopts.dx = 1e-4; // used for finite differences for cost' and cost''
72 aopts.cost_prec = 1e-18; // precision in finding the cost
73 aopts.max_cost_repeat = 5; // maximum cost repeat (rtm)
74 aopts.no_cost_samples = 5; // number of cost samples (rtm)
75 aopts.temp_ratio_scale = 1e-5; // temperature ratio scale (rtm)
76 aopts.cost_param_scale_ratio = 1; // cost parameter scale ratio (rtm)
77 aopts.temp_ann_scale = 100; // temperature annealing scale (rtm)
78 aopts.limit_acc = 1000; // limit acceptances (rtm)
79 aopts.limit_gen = 99999; // limit generated (rtm)
80 aopts.limit_inv_gen_states = 1000; // limit invalid generated states (rtm)
81 aopts.acc_to_gen_ratio = 1e-4; // accepted to generated ratio (rtm)
82
83 x = asamin(cost, gendist, x0, b, acctest, costsched, aopts);
84
85 x
8.6 Levin u-transformation for series summation acceleration

Levin u-transformation is a way of calculating the infinite sums (assuming that such exist), by considering smaller number of terms in the series then a brute-force summation would have required.

8.6.1 levinu

**Format:** \( L = \text{levinu}(S/, \text{mint}, \text{maxt}/) \)

**Arguments:**

1. \( S \), real vector;
2. \( \text{mint}, \text{maxt} \), integers.

**Result:** \( L = \langle \langle \text{sum}; \text{terms}; \text{error} \rangle, \text{list} \rangle \).

**Abstract:** Let a row-vector \( S \) contain terms of the series to be added, up to some term, say \( N \). The function \( \text{levinu} \) tries to calculate the limit of the series by using Levin u-transformation in the limit \( N \to \infty \). The result is a list \( L \) containing an estimated sum of the series \( \text{sum} \), number of terms need to calculate the sum \( \text{terms} \), and the estimated error of the summation \( \text{error} \). The integers \( \text{mint} \) and \( \text{maxt} \), if given, provide bounds for the number of terms of the series to be used.
8.7 Fitting and modeling

For fitting of the data to a model r1ab provides two solvers: least-squares solver (linear and nonlinear, where the linear relies on singular value decomposition for all matrix inversion operations, while nonlinear uses a Levenberg-Marquardt algorithm, as implemented in the GSL/MINPACK), and the orthogonal distance regression solver (as implemented in ODRPACK by Boggs et al. (1992)).

Least-squares fit

8.7.1 polyfit

Format: \( c = \text{polyfit}(Y, X, /\text{options}) \)

Arguments:

1. \( Y \), real vector, representing the dependent variable, or
   \( Y = \langle \langle \text{val}; \text{wgt} \rangle \rangle \), list, containing the values and their weights, each real vector;

2. \( X \), real vector, independent/explanatory variable.

3. \( \text{options} = \langle \langle \text{maxdeg}; f; \text{error} \rangle \rangle \), a list of parameters.

Result: \( c = \langle \langle \text{coef}; \text{error}; \text{degree} \rangle \rangle \).

Abstract: Given the data points \((x_i, y_i)\) the function finds the best-fit polynomial. The degree of the polynomial is sought in range 0..maxdeg, using either absolute error criterion, if \( \text{error} \) is given, or using \( f \)-statistics, if \( f = 0.01, 0.05, 0.1 \) is given. On return the list \( c \) contains the coefficients \( \text{coef} \) of the best-fit polynomial, its \( \text{degree} \), and its maximal \( \text{error} \) in the coefficients. The function utilizes routine DPOLFT from the SLATEC library.

8.7.2 lsfit

I Linear/Multilinear Fit

Format: \( c = \text{lsfit}(Y, X) \)

Arguments:

1. \( Y \), real vector, or
   \( Y = \langle \langle \text{val}; \text{wgt} \rangle \rangle \), containing values and their weights, both real vectors, where the weights are normalized to \( N \), the size of the sample;

2. \( X \), real matrix, containing row-wise entries for the independent variable;

Result: \( c = \langle \langle \text{coef}; \text{cov}; \text{chisq} \rangle \rangle \), list, with entries:

- \( \text{coef} \), real vector, best fit values as found by the method of least squares, as given in Eq. (8.12);
- \( \text{cov} \), real matrix, covariance matrix for vector \( \text{coef} \);
- \( \text{chisq} \), sum of squares of residuals, as given in Eq. (8.12).
Abstract: Finds real vector $\text{coef}$ by method of least squares, which minimizes the residual $\text{chisq}$ over the data set,

\[
\text{chisq} = \sum_i wgt_i \cdot (Y_i - \sum_j \text{coef}_j \cdot X_{i,j})^2.
\] (8.12)

Note: $\sqrt{\text{cov}_{f,j}}$ represents the numerical error in the solution coefficient $\text{coef}_j$.

II Polynomial fit
Format: $c = \text{lsfit}(Y, X, N)$,

Arguments:

1. $Y$, real vector, or $Y = << \text{val}; wgt >>$, containing values and their weights, both real vectors, where the weights are normalized to $N$, the size of the sample;
2. $X$, real matrix, with row-wise independent variable;
3. $N$, integer or row-vector, degree of fitting polynomial or a template of the fitting polynomial, in that order;

Result: $c = << \text{coef}; \text{cov} >>$, list with entries:

- $\text{coef}$, row-vector, polynomial coefficients in standard format;
- $\text{cov}$, real matrix, covariance matrix of vector $c$.

Abstract: Finds the least-square fit of possibly weighted data to a particular polynomial.

III General non-linear fit
Format: $c = \text{lsfit}(Y, X, p, f, dfdp, \text{options})$,

Arguments:

1. $Y$, real vector, or $Y = << \text{val}; wgt >>$, containing values and their weights, both real vectors, where the weights are normalized to $N$, the size of the sample;
2. $X$, real matrix, with row-wise independent variable;
3. $p$, real vector, initial guess for the function $f$ parameters;
4. $f =$ \text{function}(x, p), $dfdp =$ \text{function}(x, p), functions, former describes the relationship between $y$ and $x$ as $y = f(x, p)$, while the latter is its jacobian with respect to $p$;
5. $\text{options} = << \text{convtest}; \text{stdout}; \text{eabs}; \text{erel}; \text{maxi} >>$, list, options for the solver where:
   - $\text{convtest}$, integer, chooses the convergence test where 0 is for delta-test (includes $\text{erel}, \text{eabs}$, default) while 1 is for gradient test (only $\text{eabs}$);
   - $\text{stdout}$, string, filename where run-time messages are posted;
   - $\text{eabs}, \text{erel}$, positive real scalars, absolute and relative error for convergence criterion;
   - $\text{maxi}$, integer, maximal number of iterations to reach the specified tolerances;

Result: \text{list}\ast c = << \text{coef}; \text{cov}; \text{chisq} >>.
8.7. FITTING AND MODELING

- $c$, row-vector, new values of parameter array $p$;
- $cov$, real matrix, covariance matrix of vector $p$.
- $\text{chisq}$, sum of squares of residuals.

Abstract: Performs the least-square fit of array $p$ using the function $y = f(x, p)$ and the dataset \{X, Y\}.

Orthogonal distance regression

8.7.3 odrfit

Format: $c = \text{odrfit}(\{\text{Y}\}, X, p, f, /\text{dfdp}, \text{dfdx}/, \text{options})$.

Arguments:

1. $Y$, real matrix the rows of which are target values or list $\langle \text{val}; \text{wgt} \rangle$ with $\text{val}$ real matrix and $\text{wgt}$ positive real column-vector. If $Y$ is given an explicit model, $y = f(x, p)$, is assumed. In the absence of $Y$ an implicit model, $f(x, p) = 0$, is assumed;

2. $X$, matrix the rows of which are explanatory values, or list $\langle \text{val}; \text{wgt}; /\text{fix} \text{x}/; /\text{dx}/ \rangle$ with $\text{val}$ real matrix, $\text{wgt}$ positive real column-vector, $\text{fix} \text{x}$ real matrix of the same size as $x$ containing 1’s for fixed entries and 0’s for free, $\text{dx}$ real matrix of the same size as $x$ that provides an initial guess for the orthogonal-distance regression procedure;

3. $p$, real vector, initial guess for the parameters of the function $f$;

4. $f = \text{function}(x, p)$, function, describes the relationship between $Y$ and $X$ using the parameter array $p$;

5. $\text{dfdx} = \text{function}(x, p)$, function, jacobian of $f$ with respect to $x$. If omitted it is found numerically using a central difference scheme;

6. $\text{dfdp} = \text{function}(x, p)$, function, jacobian of $f$ with respect to $p$. If omitted it is found numerically using a central difference scheme;

7. $\text{options} = \langle\langle \text{fix} \text{x}; \text{scale} \text{p}; \text{stdout}; \text{taufac}; \text{sstol}; \text{partol}; \text{imethod}; \text{maxi} \rangle\rangle$, list, parameters that control different aspects of the calculation:
   - $\text{fix} \text{x}$, real vector, signals the solver if a component $p[i]$ is fixed ($\text{fix} \text{x}[i] = 1$), or free ($\text{fix} \text{x}[i] = 0$);
   - $\text{scale} \text{p}$, real vector, multiplier with which the appropriate component in array $p$ is scaled internally so that the numeric procedure is more stable. E.g., if $p[i]$ is expected to be of the order of $10^{-6}$, then $\text{scale} \text{p}[i] = 10^6$.
   - $\text{imethod}$, integer, 0 for orthogonal distance regression, 1 for least-squares solver. ODR least-squares solver can work with $f$ a vector function;
   - $\text{maxi}$, integer, maximum number of iterations to reach the specified tolerances;
   - $\text{stdout}$, string, filename to which run time messages are posted;
   - $\text{partol}$, $\text{taufac}$, $\text{sstol}$, scalars, tolerances that control the convergence as follows: $\text{taufac}$ is the initial size of the trust region (default 1.0), $\text{sstol}$ is a stopping tolerance for the sum-of-squares (default -1.0, let the code decide), and $\text{partol}$ is a stopping criterion for the parameter tolerance (default -10.0, let code decide).
Result: \( c = \langle \text{coef}; \text{cov}; /dx; /dy; \text{status} \rangle \), list, its entries are:

- \text{coef}, real vector, best fit values for the parameter array \( p \);
- \text{cov}, real matrix, covariance matrix of the best fit parameters;
- \( dx, dy \), real matrices, error in \( x \) and in \( y \) for best fit parameters. \( dx \) appears only if solver is in odr mode;
- \text{status}, integer, status of solution,

\[
\text{status} = I_5 I_4 I_3 I_2 I_1, \tag{8.13}
\]

where for \( \text{status} < 5 \) the solver is making progress toward the solution, with \( I_1 = 1 \), for convergence in sum of squares, 2 for parameter convergence, 3 for both sum of squares and parameter and 4 for reaching the iteration limit. If \( \text{status} > 5 \), the following two cases may occur

- \( I_5 = 0 \) questionable result, where for \( I_4 \neq 0 \) derivatives are possibly not correct, for \( I_3 \neq 0 \) the problem is not full rank at the solution, for \( I_1 \neq 0 \) see above. E.g., \( \text{status} = 1001 \) occurs when \( y \)'s are poorly scaled (converged solution, but the solver is not sure about correctness of the derivatives).
- \( I_5 \geq 1 \) fatal error, where \( I_5 = 1 \) for number of parameters greater than the size of the fitting set, \( I_5 = 4 \) for error in derivatives, \( I_5 = 6 \) for numerical error (e.g., NaN's appeared). E.g., \( \text{status} = 60000 \) occurs when the solver fails to converge.

The following control mechanism encompasses the above information

```matlab
1  ... 
2  c = odrfit (y, x, p0, f, dfdp, dfdx, opts);
3  if (int(c.status) && int(3) > 0 || c.status < 5)
4     {
5         // fit successful: it is a good practice to check whether number of iterations
6         // needs to be increased, tolerances changed, et c.
7         ...
8  ...
9     else
10        // fit failed: reading the messages produced by the solver may give clues
11        ...
12     }
13  ... 
14  ...
```

Abstract: Finds the best fit of \( Y \) to \( f(x, p) \) by adjusting \( p \) and \( Y \) (least-squares and odr mode), and \( x \) (odr only).

Note: The solver is based on the package ODRPACK by Boggs et al. [1992]. The manual for ODRPACK is part of the rlab distribution, and is located in `flibs/odrpack/doc`. 
Example: Nonlinear least squares fit using the GSL solvers.

```c
//
// simple least square non-linear fit
//
srand( iseed() );

NITER = 100; // this is to test for segmentation faults
func = function(x,p)
{  
  return p[1]*x[;1]+x[;2]) + p[2]*(x[;1].^2+x[;2].^2);
};

dfunc = function(x,p)
{  
  y=[];  
  y[;1] = (x[;1]+x[;2]);  
  y[;2] = (x[;1].^2+x[;2].^2);  
  return y;
};

// nonlinear fit
//
x = rand(100, 2);
p0 = [10,2];  
y = func(x,p0) + 0.1*rand(100,1);  
p=[1,1];  
for(i in 1:NITER)
{
  z=lsfit(y,x,p,func,dfunc);
}
```
Example: Nonlinear fit using orthogonal-distance regression solver (ODRPACK).

```c
// odr: a simple demonstration

srand( iseed() );

NITER = 1000; // test for segmentation faults

func = function(x,p)
{
  // model
  return p[1].*(x[;1]+x[;2]) + p[2].*(x[;1].^2+x[;2].^2);
}

dfdp = function(x,p)
{
  // jacobian df/dp
  y=[];
  y[;1] = (x[;1]+x[;2]);
  y[;2] = (x[;1].^2+x[;2].^2);
  return y;
}

dfdx = function(x,p)
{
  // jacobian df/dx
  y=[];
  y[;1] = p[1] + p[2].*2*x[;1];
  return y;
}

// generate noisy data
x = rand(100, 2);
p0 = [10,2];
y = func(x,p0) + 0.5*rand(100,1);

options=<<>>;
options.stdout = rlab_initstderr(); // printout what you are doing
p=rand(1,2);
// p=[10,1];

for(i in 1:NITER)
{
  spinner();
}
The report generated by the code is

```
*******************************************************
* ODRPACK VERSION 2.01 OF 06-19-92 (DOUBLE PRECISION) *
*******************************************************

*** INITIAL SUMMARY FOR FIT BY METHOD OF ODR ***

--- PROBLEM SIZE:
   N = 100 (NUMBER WITH NONZERO WEIGHT = 100)
   NQ = 1
   M = 2
   NP = 2 (NUMBER UNFIXED = 2)

--- CONTROL VALUES:
   JOB = 00000
   = ABCDE, WHERE
   A=0 ==> FIT IS NOT A RESTART.
   B=0 ==> DELTAS ARE INITIALIZED TO ZERO.
   C=0 ==> COVARIANCE MATRIX WILL BE COMPUTED USING
         DERIVATIVES RE-EVALUATED AT THE SOLUTION.
   D=0 ==> DERIVATIVES ARE ESTIMATED BY FORWARD DIFFERENCES.
   E=0 ==> METHOD IS EXPPLICIT ODR.
   NDIGIT = 16 (ESTIMATED BY ODRPACK)
   TAUFAC = 1.00E+00

--- STOPPING CRITERIA:
   SSTOL = 1.00E-14 (SUM OF SQUARES STOPPING TOLERANCE)
   PARTOL = 2.17E-19 (PARAMETER STOPPING TOLERANCE)
   MAXIT = 50 (MAXIMUM NUMBER OF ITERATIONS)

[ S N I P ] ...
[ S N I P ] ...

RLaB: built-in solver ODRPACK reports success.
RLaB: ODRPACK calculation lasted 0.13 sec.
```
8.8 Numerical integration

Basic Integration of a Real Scalar Function of a Real Scalar Variable

The solvers in this section rely on the GSL/QUADPACK integration package.

8.8.1 nintegrate

I Format: \( x = \text{nintegrate}(f,/p/,X/,\text{options/}) \)

Arguments:

1. \( f = \text{function}(x,/p/) \), a function;
2. \( p \), its parameters,
3. \( \text{options} = <\text{eabs}; \text{erel}; \text{maxi}; \text{ikey} > \), a list of options for the solver.

Result: \( x \), a real scalar, result of integration.

Abstract: The function calculates the definite integral on the intervals, the endpoints of which are given in a row-vector \( X = [x_1, \ldots, x_N] \). It is allowed for endpoints \( x_1 \) and \( x_N \) to be \(-\infty\) and \( \infty \), respectively. Depending on the bounds of integration, appropriate method from the GSL is chosen: for integration on \([-\infty, a]\) the integrator \( \text{qagil} \) is chosen, on \([b, \infty]\) it is \( \text{qagiu} \), and for \([-\infty, \infty]\) it is \( \text{qagi} \). If the interval of integration is finite, then it is \( \text{qag} \) that is used, and if it does not succeed then \( \text{qags} \) is called. The latter can deal with unknown singularities fairly successfully. Parameters \( \text{eabs} \) and \( \text{erel} \) determine absolute and relative errors in computation of integral, while \( \text{maxi} \) is the maximum number of subdivisions of the subinterval allowed to achieve desired accuracy.

If the interval is finite and the function is well behaved, than the integrator \( \text{qag} \) that relies on Gauss-Konrod method is used. This integrator may receive parameter \( \text{ikey} \) from user, which determines the integration rule,

- \( \text{ikey} = 1 \), for 15 point rule,
- \( \text{ikey} = 2 \), for 21 point rule,
- \( \text{ikey} = 3 \), for 31 point rule, (default),
- \( \text{ikey} = 4 \), for 41 point rule,
- \( \text{ikey} = 5 \), for 51 point rule,
- \( \text{ikey} = 6 \), for 61 point rule.

In short,

\[
\text{nintegrate}(f,/p/,[x_1,..x_N]) = \sum_{i=2}^{N} \int_{x_{i-1}}^{x_i} f(x,/p/) \, dx.
\]  

II Format: \( x = \text{nintegrate}(t/,X/) \)

Arguments:

1. \( t = [x_i, y_i]_{i=1,N} \), a matrix of tabulated values of the function;
2. \( X = [\text{xlo}, \text{xhi}] \), integration interval;

Result: So applied \( \text{nintegrate} \) integrates a function given in tabular form, using the method of overlapping parabolas (SLATEC, \text{davint}).
8.8. NUMERICAL INTEGRATION

8.8.2 nintegrate

Format: \( x = nintegrate(f, p, X, \text{options}) \)

Arguments:
1. \( f = function(x, p) \), a real scalar function;
2. \( p \), its parameters;
3. \( X = [x_1, \ldots, x_N] \), a real vector of integration intervals;
4. \( \text{options} = \langle eabs; erel; maxi \rangle \), a list of options.

Result: \( x \), a real scalar.

Abstract: The function calculates a definite integral on finite intervals the end-points of which are given in row-vector \( X = [x_1, \ldots, x_N] \). The GSL integrator this function relies on, \( qagp \), that is, can deal with singularities and discontinuities (if they are given as interior integration end-points). This function cannot deal with \( \pm \infty \)'s the end-points. As was the case with \( \text{ ninintegrate} \), the precision of integration is determined by the absolute and relative errors, \( eabs \) and \( erel \), respectively, while \( maxi \) determines the maximum number of subdivisions allowed to achieve the desired accuracy.

8.8.3 cauchypval

Format: \( x = cauchypval(f, p, X, \text{options}) \)

Arguments:
1. \( f = function(x, p) \), a real scalar function;
2. \( p \), its parameters;
3. \( X = [xlo, c, xhi] \), a real vector containing the integration interval with the special point \( c \);
4. \( \text{options} = \langle eabs; erel; maxi \rangle \), a list of options.

Result: \( x \), a real scalar.

Abstract: The function calculates the cauchy principal value integral,

\[
cauchypval(f, p, [xlo, c, xhi]) = \mathcal{P} \int_{xlo}^{xhi} f(x, p) \frac{x - c}{x - c} \, dx = \lim_{\epsilon \to 0} \left( \int_{xlo}^{c-\epsilon} f(x, p) \frac{x - c}{x - c} \, dx + \int_{c+\epsilon}^{xhi} f(x, p) \frac{x - c}{x - c} \, dx \right).
\]

(8.15)

As was the case with \( \text{ninintegrate} \) and \( \text{nintegrate} \), the precision of integration is determined by the absolute and relative errors, \( eabs \) and \( erel \), respectively, while \( maxi \) determines the maximum number of subdivisions allowed to achieve the desired accuracy.

Note: The function being integrated on \([xlo, xhi]\) is \( \frac{f(x, p)}{x - c} \) and \( f(x, p) \).
CHAPTER 8. ADVANCED GENERAL NUMERICAL PROCEDURES AND ALGORITHMS

SPECIAL INTEGRALS OF A REAL SCALAR FUNCTION OF A REAL SCALAR VARIABLE

8.8.4 nintqaws

Format: \[ x = \text{nintqaws}(f, p, xlo, xhi, \alpha, \beta, \mu, \nu, \text{options}) \]

Arguments:
1. \( f = \text{function}(x, p) \), a real scalar function;
2. \( p \), its parameters;
3, 4. \( xlo, xhi \), real scalars, lower and upper bound of integration;
5-8. \( \alpha, \beta, \mu, \nu \), the parameters for the weight function;
9. \text{options} = \langle \langle \text{eabs; erel; maxi} \rangle \rangle \), a list of options.

Result: \( x \), a real scalar.

Abstract: The function calculates following integral
\[ \text{nintqaws}(f, p, xlo, xhi, \alpha, \beta, \mu, \nu) = \int_{xlo}^{xhi} dx \, f(x, p) \, (x - xlo)^\alpha (xhi - x)^\beta \log^\mu(x - xlo) \log^\nu(xhi - x). \]  
\[(8.16)\]

As was the case with \text{nintegrate}, the precision of integration is determined by the absolute and relative errors, \text{eabs} and \text{erel}, respectively, while \text{maxi} determines the maximum number of subdivisions allowed to achieve the desired accuracy.

8.8.5 nintcos, nintsin

Format: \( x = \text{nintcos}(f, p, X, \omega, \text{options}) \), \( x = \text{nintsin}(f, p, X, \omega, \text{options}) \)

Arguments:
1. \( f = \text{function}(x, p) \), a real scalar function;
2. \( p \), its parameters;
3. \( X = [x_1, \ldots, x_N] \), a real vector, bounds of integration intervals;
4. \( \omega \), a real scalar;
5. \text{options} = \langle \langle \text{eabs; erel; maxi; maxn} \rangle \rangle \), a list of options.

Result: \( x \), a real scalar.

Abstract: The function calculates integral,
\[ \text{nint} \begin{cases} \cos & \text{cos} \\ \sin & \text{sin} \end{cases} (f, p, X, \omega) = \sum_{i=2}^{N} \int_{x_{i-1}}^{x_i} dx \, f(x, p) \, \cos(\omega x) \sin(\omega x), \]  
\[(8.17)\]

where it is allowed for the uppermost bound to be \(+\infty\). As was the case with \text{nintegrate}, the precision of integration is determined by the absolute and relative errors, \text{eabs} and \text{erel}, respectively, while \text{maxi} determines the maximum number of subdivisions allowed to achieve the desired accuracy. Parameter \text{maxn} has a similar purpose.
8.8.6 nintlt

**Format:** \( x = \text{nintlt}(s, k, f, p, X, \text{options}) \)

**Arguments:**

1. \( s \), real vector, range of values of \( s \) for which the Laplace transform is to be calculated;
2. \( k \), integer vector, range of values of \( k \) for which the Laplace transform is to be calculated. If not given, \( k = 0 \) is assumed;
3. \( f = \text{function}(x, p) \), real scalar;
4. \( p \), entity, parameter for \( f \);
5. \( X = [x_1, \ldots, x_N] \), real vector, end points of integration intervals. If not given \([0, +\infty]\) is assumed;
6. \( \text{options} = << \text{eabs}; \text{erel}; \text{maxi}; \text{ikey} >> \), a list of options for the GSL integrators.

**Result:** \( x \), real matrix of size \( \text{dim} s \)-by-\( \text{dim} k \).

**Abstract:** The function calculates Laplace transform over a given integration interval or over real half-line,

\[
nintlt(s, k, f) = s^k \int_0^{\infty} dx \exp(-sx) f(x)
\]  

(8.18)

8.8.7 invlt

**Format:** \( y = \text{invt}(f, p, t, \text{options}) \)

**Arguments:**

1. \( f = \text{function}(s, p) \), complex function, Laplace transform inverse of which is being sought;
2. \( p \), entity, parameter for \( f \);
3. \( t \), real vector, range of values of \( t \) for which the Laplace transform is to be calculated;
4. \( \text{options} = << \text{erel}; \text{sigma}; \text{ssbar}; \text{maxfeval} >> \), a list of options for the solver, where \( \text{erel} \) is the relative error, \( \text{sigma} \) is the abcissa of convergence, \( \text{ssbar} \) is a mysterious parameter related to the sine/cosine transform used in finding the inverse, and \( \text{maxfeval} \) is a maximum number of function evaluations used to obtain the inverse.

**Result:** \( y \), real vector, inverse of Laplace Transform at points \( t \).

**Abstract:** The function calculates an inverse Laplace transform by means of expansion in sine and cosine fourier series [D’Amore et al., 1999]. The method utilizes the Q-D algorithm to accelerate the convergence of the series. The summation of successive terms is truncated when the estimated truncation error is less than provided tolerance (relative error).
8.8.8 nintconv

Format: \( x = \text{nintconv}(f_1, p_1, f_2, p_2, x, c, \text{options}) \)

Arguments:

1, 2 \( f_1 = \text{function}(x, p_1) \), real function of a real variable;
\( p_1 \), entity, a parameter for the function \( f_1 \)

3, 4 \( f_2 = \text{function}(x, p_2) \), real function of a real variable;
\( p_2 \), entity, a parameter for the function \( f_2 \);

5 \( x \), scalar or vector, integration range where \( x = [x_{lo}, x_{hi}] \) or, \( x_{lo} = 0 \) and \( x_{hi} = x \);

6 \( c \), scalar, offset value. If not given then \( c = x_{hi} \);

6, 7 \( \text{options} = << \text{eabs; erel; maxi; ikey} >> \), a list of options for the GSL integrators.

Result: \( x \), scalar.

Abstract: calculates a convolution of two functions on an interval \([x_{lo}, x_{hi}]\), with the offset \( c \),

\[
\text{nintconv}(f_1, p_1, f_2, p_2, [x_{lo}, x_{hi}], c) = \int_{x_{lo}}^{x_{hi}} dx \ f_1(x, p_1) \cdot f_2(c - x, p_2) \quad (8.19)
\]
8.8. NUMERICAL INTEGRATION

BASIC INTEGRATION ON HYPERCUBE AND MINIMAL SIMPLEX DOMAINS IN MULTIDIMENSIONS

These functions utilize functions by Genz [1986] for integration in multidimensions.

8.8.9 nintmd

Format: \[ x = \text{nintmd}(f, p, X, \text{options}) \]

Arguments:

1. \( f = \text{function}(x, p) \), a function being integrated;
2. \( p \), a parameter passed directly to the function \( f \);
3. \( X \), a hypercube domain of the integration, a sequence of integration intervals for each dimension of the vector variable \( x \),
   \[ X = \begin{bmatrix} x_{lo_1} & x_{hi_1} \\ x_{lo_2} & x_{hi_2} \\ \vdots \\ x_{lo_n} & x_{hi_n} \end{bmatrix}, \]
   \[ \text{(8.20)} \]
   where \( x_{lo_i} < x_i < x_{hi_i} \), for \( i = 1, \ldots \dim x = n \);
4. \( \text{options} \), a list containing the entries
   - \( \text{maxi} \), maximal number of function evaluations to reach the desired precision;
   - \( \text{mini} \), minimal number of function evaluations the code will do before trying to estimate the integral;
   - \( \text{eabs} \), absolute error in the estimate of the integral;
   - \( \text{erel} \), relative error in the estimate of the integral;

Result: \( \text{double } x \).

Abstract: The function integrates a scalar function of a vector variable \( x \) over a hypercube \( X = [x_{lo}, x_{hi}] \),

\[ \text{nintmd}(f, p, X) = \prod_{i} \int_{x_{lo_i}}^{x_{hi_i}} dx_i \ f(x, p). \]  
\[ \text{(8.21)} \]

8.8.10 nintsimplex

Format: \( x = \text{nintsimplex}(f, p, S, \text{options}) \)

Arguments:

1. \( f = \text{function}(x, p) \), a function being integrated;
2. \( p \), a parameter passed directly to the function \( f \);
3 $S$, a matrix containing stacked vertices of a minimal simplex that represents the domain of the integration,

$$S = \begin{bmatrix} s_{1,1} & s_{1,2} & \ldots & s_{1,n} \\ s_{2,1} & s_{2,2} & \ldots & s_{2,n} \\ \vdots \\ s_{n+1,1} & s_{n+1,2} & \ldots & s_{n+1,n} \end{bmatrix}. \quad (8.22)$$

Minimal simplex in a n-dimensional space consists of $n + 1$ non-colinear points. The code does not check the rank of matrix $S$.

4 $options$, a list containing the entries

- $maxi$, maximal number of function evaluations to reach the desired precision;
- $mini$, minimal number of function evaluations the code will do before trying to estimate the integral;
- $eabs$, absolute error in the estimate of the integral;
- $erel$, relative error in the estimate of the integral;
- $ikey = 0, \ldots, 5$, integration rule (default 4).

Result: $double \ x$.

Abstract: The function integrates a scalar function of a vector variable $x$ over a minimal simplex $S$,

$$\text{nintsimplex}(f, p, X) = \int_S \prod_i dx_i \ f(x, p). \quad (8.23)$$
MONTE CARLO INTEGRATION ON HYPERCUBE DOMAINS IN MULTIDIMENSIONS

In a Monte Carlo scheme the function $f$ is integrated by sampling the points from the hypercube domain using the function $f$ as a statistical weight.

Note: Monte Carlo integration uses an integer random number generator, which can be tuned by the user.

8.8.11 mcnint

Format: $x = \text{mcnint}(f, p, X, \text{options})$

Arguments:

1. $f = \text{function}(x, p)$, the function being integrated;
2. $p$, a parameter passed directly to the function;
3. $X$, a matrix containing appended real vectors $xlo$ and $xhi$, $X = [xlo, xhi]$;
4. $\text{options}$, determines the method of integration, where
   - $\text{options} = \text{ncalls}$ is an integer, or $\text{options} = << \text{imethod; ncalls} >>$, with $\text{imethod} = 0$. The integration is performed using "plain" Monte Carlo method. The parameter $\text{ncalls}$ determines the number of points over which the function $f$ is evaluated. Default value is $\text{ncalls} = 100,000$. "Plain" is a default Monte Carlo solver.
   - $\text{options} = << \text{imethod; ncalls; efrac; alpha; dither} >>$, with $\text{imethod} = 1$. The integration is performed using MISER Monte Carlo method. Besides $\text{ncalls}$ the other parameters are:
     * $\text{efrac}$, the parameter specifies fraction of the currently available function calls. Default value is 0.1.
     * $\text{alpha}$, this parameter controls how estimated variance for two subregions of a bisection are combined when allocating the points. Default value is 2.0.
     * $\text{dither}$, this parameter introduces a random fractional variation of size into each bisection. Default value is 0.1.

Note: The MISER algorithm is of Press and Ferrar. See GSL manual for details and further reading.

- $\text{options} = << \text{imethod; ncalls; thncalls; chicomp; alpha; mode; ostream} >>$, with $\text{imethod} = 2$. The integration is performed using VEGAS Monte Carlo method. Besides $\text{ncalls}$, VEGAS requires $\text{thncalls}$, the number of points over which the integrator “warms-up.” Its default value is $\text{thncalls} = \text{ncalls}/10$. Other parameters controlling the solver are
  * $\text{chicomp}$, convergence criterion is given by $\text{chicomp}$, it measures acceptable distance from target value 1. Default value is $\text{chicomp} = 0.5$.
  * $\text{alpha}$, the parameter controls the stiffness of the rebinning algorithm. Default value is $\text{alpha} = 1.5$.
  * $\text{mode}$, importance sampling used by algorithm, with values, $\text{mode} = 0$, for importance sampling, $\text{mode} = 1$ for stratified sampling, and $\text{mode} = 2$, for algorithm to pick on its own (default).
  * $\text{ostream}$, a file or a console where the above verbal messages are dumped or printed. If $\text{ostream}$ is given the code assumes maximal verbosity.
Note: The VEGAS algorithm was written by Lepage. See GSL manual for more information and further reading.

Note: For VEGAS algorithm the GSL manual mentions a parameter “stage”. Here, the parameter is utilized as following. For thermalization stage = 0, that is, code creates a new mesh, while for actual calculation stage = 1, that is, the code utilizes mesh created (and optimized) during thermalization. If an alternative implementation is required, drop me an email.

Result: double x, the result of integration.
8.9 Tchebychev polynomials and differentiation

8.9.1 tchebyfit
Format: $c = \text{tchebyfit}(f, /p/, n)$
Arguments:
1. $f = \text{function}(x, p)$, function;
2. $p$, entity, $f$’s parameter;
3. $n$, integer.
Result: $c$, real vector.
Abstract: Finds the expansion of $f$ on $[-1, 1]$ in terms of a series of tchebyshev polynomial up to the order $n$.

8.9.2 tchebyval
Format: $y = \text{tchebyval}(x, c)$
Arguments:
1. $x$, real vector, the elements of which are in $[-1, 1]$;
2. $c$, real vector, coefficients of a tchebyshev polynomial.
Result: $y$, real vector.
Abstract: Finds the value of tchebyshev series $c$ at $x$.

8.9.3 tchebyint
Format: $ic = \text{tchebyint}(c)$
Arguments:
1. $c$, real vector, coefficients of a tchebyshev polynomial.
Result: $ic$, coefficients of a tchebyshev polynomial.
Abstract: Integrates a tchebishev polynomial in $c$ and represents the result in terms of a new tchebishev polynomial $ic$.

8.9.4 tchebyder
Format: $dc = \text{tchebyder}(c)$
Arguments:
1. $c$, real vector, coefficients of a tchebyshev polynomial.
Result: $dc$, coefficients of a tchebyshev polynomial.
Abstract: Differentiates a tchebishev polynomial in $c$ and represents the result in terms of a new tchebishev polynomial $dc$. 

8.9.5 ndiff

- **Format:** \( yp = ndiff(f, [p], x0, xlo, xhi) \)

  **Arguments:**
  1. \( f = function(x, [p]) \), function;
  2. \( p \), entity, \( f \)'s parameters;
  3. \( x0 \), real vector, arguments at which \( df/dx \) is found;
  4,5. \( xlo, xhi \), real scalars, end-points of \( x0 \).

  **Result:** \( yp \), real vector of size \( x0 \).

  **Abstract:** Finds numerically \( df/dx \) at point(s) \( x0 = [x0_1, ..., x0_N] \). If \( xlo \) and \( xhi \) are not given that for all points in \( x0 \) a central difference scheme is used. If \( xlo \) and \( xhi \) are given then
  - for \( x0_1 = xlo \), a forward difference scheme is used at \( x0_1 \),
  - for \( x0_N = xhi \), a backward difference scheme is used at \( x0_N \),
  - otherwise \( xlo \) and \( xhi \) are ignored and the scheme reverts to central difference.

- **Format:** \( yp = ndiff(m) \)

  **Arguments:**
  1. \( m \), two-column real matrix, \( m = [x_1, y_1; \ldots; x_N, y_N] \).

  **Result:** \( yp \), real vector.

  **Abstract:** Finds numerical derivative of a scalar function \( f \) given in the form of a table \( [x_i, y_i]_{i=1,N} \). At endpoints the forward/backward difference formula is used while for midpoints the central difference formula is used. The values \( x_i \) are ascending but need not be equidistant.
8.10 Numerical differentiation of a real vector function

8.10.1 ndiffs
Format: \( y = \text{ndiffs}(f,/p/,x0) \)
Arguments:
1. \( f = \text{function}(x,/p/) \), function;
2. \( p \), entity, \( f \)'s parameter;
3. \( x0 \), real vector.

Result: \( y \), real vector.
Abstract: Finds numerical derivative \( df/dx \) at \( x0 \), using a central-difference method.

8.10.2 ndiv
Format: \( y = \text{div}(f,/p/,x0) \)
Arguments:
1. \( f = \text{function}(x,/p/) \), function;
2. \( p \), entity, \( f \)'s parameter;
3. \( x0 \), real vector.

Result: \( y \), real scalar.
Abstract: Finds \( \nabla \) or divergence of an \( f \) at \( x0 \), using central-difference method. Here, it is important that \( \dim f = \dim x \).
8.11 Minimization of a real function of a real vector variable

Finding a minimum of a real function of a real variable

8.11.1 findmin

Format: \( x = \text{findmin}(f, /p/, X, /\text{options}/) \)

Arguments:
1. \( f = \text{function}(x, /p/) \), the function being minimized;
2. \( p \), a parameter passed directly to the function;
3. \( X = [xlo, xhi] \), a search interval \( X = [xlo, xhi] \);
4. \( \text{options} = \langle eabs; erel; maxi; imethod \rangle \), a list, where \( eabs \) and \( erel \) are the absolute and relative errors, \( maxi \) is the maximal number of iterations allowed to achieve them, \( imethod = 0 \) for Brent method (default), and 1 for the golden section.

Result: a real scalar \( x \in X \).

Abstract: The function finds a minimum of a scalar function \( f \) in the interval \( X = [xlo, xhi] \) by using a bracketing-type method.

Minimization of a real function in multi-dimensions

8.11.2 diffevol

Format: \( c = \text{diffevol}(	ext{cost,} /p/, x0, /\text{options}/) \)

Arguments:
1. \( \text{cost} = \text{function}(x, /p/) \), function of vector variable \( x \) being minimized;
2. \( p \), anything, passed directly to the function;
3. \( x0 \), real matrix, stacked column vectors of initial guesses for \( x \). The number of columns determines the size of a population from which the guesses for optimal solution will be constructed depending on the strategy.
4. \( \text{options} = \langle \text{strategy; } f; \text{f } \text{dither; cr; genmax; stdout; iprint; eabs; target; terr; width; width_wgt; max_fail_count} \rangle \), is a list of options for the solver,
   - \text{strategy}, integer in range 1 through 11, determines how the new samples are generated from the ones provided in matrix \( x0 \): 1 for DE/best/1/exp, 2 for DE/rand/1/exp, 3 for DE/rand-to-best/1/exp, 4 for DE/best/2/exp, 5 for DE/rand/2/exp, 6 for DE/best/1/bin, 7 for DE/rand/1/bin, 8 for DE/rand-to-best/1/bin, 9 for DE/rand-to-best/1/bin, 10 for DE/best/2/bin, 11 for DE/rand/2/bin.
   - \( f \), positive scalar, usually between 0.5 and 1, and only in rare cases greater than 1;
   - \( \text{f}_\text{dither} \), positive scalar, magnitude of non-negative noise added to \( f \):
     \[
     f \rightarrow f + f_\text{dither} \cdot u, \tag{8.24}
     \]
     where \( u \) is a uniform random variate from \([0, 1]\). If not provided it is set to zero. The authors observe that \( f = 0.5 \) and \( f_\text{dither} = 0.5 \) improves convergence for noisy cost function;
• \( cr \), positive scalar, between 0 and 1, with 0, 0.3, 0.7 and 1. being worth to be tried first;
• \( genmax \), positive scalar, maximal number of generations during which the optimization is attempted;
• \( bounds \), two-column matrix which rows represent lower and upper bound on the particular parameter. The algorithm used for resetting the value if it went out of bounds is for lower bound,
\[
t_i = b_i + u \cdot (p_{j,i} - b_i),
\]
while for upper bound
\[
t_i = B_i + u \cdot (p_{j,i} - B_i).
\]
Here, \( p_{j,i} \) is the old value of the \( j \)-th member of population, \( t_i \) is the newest best guest, \( b_i \) is the lower bound, and \( B_i \) is the upper bound for the \( i \)-th component of the parameter vector. \( u \) is the uniform random variate from \([0, 1]\).

This algorithm for dealing with the boundaries is implemented in code \texttt{DeWin} by \cite{Storn:1995}. The solver does not check whether the initial population is within \( bounds \). Hypothetically, one would provide the bounds, then use the bounds to generate initial population, no? Use \( \pm \infty() \) for components that are unbounded from above/below.
• \( stdout \), string, file or terminal to which the solver will post the run-time messages;
• \( sprint \), positive integer, if \( stdout \) is provided then every so many generation the solver prints messages about the progress of the optimization;
• \( eabs \), positive real scalar, convergence criterion, if spread of costs among the population is below this value the solver stops. If not provided, the solver ignores it;
• \( width \), \( width \_wgt \), positive real scalar, convergence criterion, if (euclidean) variance of current population \( w < width \), the solver stops. If not provided, the solver ignores it. The optional array \( width \_wgt \) of the dimension of parameter array provides relative weights of each component in calculation of \( width \):
\[
w^2 = \sum_{i=1}^{D} width \_wgt_i \cdot w_i^2,
\]
where \( w_i^2 \) is the variance of the \( i \)-th component in the population. Default value for weights is unity (all enter equally). This may be helpful if parameters differ in orders of magnitude;
• \( max \_fail \_count \), positive integer, maximal number of iterations in which no improvements are found. If reached the solver stops. If not provided the solver ignores it;
• \( target \), \( terr \), goal of optimization, if known. If the solution found by the solver approaches this value within the bound \( terr \), then solver stops. If not provided, then solver ignores it.

\textbf{Result:} \( c = < coe f; coe f \_cost; pop; pop \_cost > > \), list with entries:
• \( coe f \), real vector, currently the best solution;
• \( coe f \_cost \), scalar, cost of the best solution;
• \( pop \), matrix of stacked column vectors containing the current population. This can be used to restart the optimization as matrix \( x0 \);
• \( pop \_cost \), real vector, cost of each of the parameter vectors in current population \( pop \).

\textbf{Abstract:} Solver does the (un)constrained optimization of a function \( cost(x, /p/) \) in multidimensions using the method of differential evolution of \cite{Storn:1995}, see references in the source code.
8.11.3 findmins

(1) Format: \( x = \text{findmins}(f, /p/, x0/ \text{, options/}) \)

Arguments:

1. \( f = \text{function}(x, /p/), \) function of vector variable \( x \) being minimized;
2. \( p, \) parameter passed directly to the function;
3. \( x0, \) real vector, initial guess;
4. \( \ast \text{imethod = } 0 \)
   options =<<imethod; eabs; ss; maxi; stdout; target; standstill >>, options for the solver,
   - \( eabs, \) positive real scalar, minimum size of the simplex to finish the computation;
   - \( ss, \) positive real vector of size \( x \), simplex size, (by default \( ss_j = 1 \)) in optimization search;
   - \( maxi, \) positive integer, maximum number of tries in which the minimum is expected to be found;
   - \( target, \) real scalar, known value of the goal of optimization. If that values is reached during optimization the solver stops immediately and reports success.
   - \( standstill, \) positive integer, allowed number of iterations of the solver during which the objective function does not change. When reached the solver stops and reports “iteration did not converge yet.”
   - \( stdout, \) string, file or terminal to which the solver will post the run-time messages.

Abstract: Finds unconstrained minimum of \( f \) using the Nelder-Mead (simplex) method, as implemented in the GSL (Galassi et al., 2002).

\( \ast \text{imethod = } 1 \)
options =<< imethod; maxi; rhobeg; rhoend; npt >>, options for the solver,
- \( maxi, \) positive integer, maximum number of iterations that solver will use to find the minimum. If not given default value \( maxi = 1000 \) is used;
- \( rhobeg, rhoend, \) real scalars. They must be set to the initial and final values of a trust region radius, so both must be positive, and \( rhoend \leq rhobeg \). Typically \( rhobeg \) should be about one tenth of the greatest expected change to a variable, and \( rhoend \) should indicate the accuracy that is required in the final values of the variables. If not given the values \( rhoend = 0.1 \) and \( rhobeg = 10 \) are used.
- \( npt, \) positive integer. \( npt \) is the number of interpolation conditions. Its value must be in the interval \( \lfloor N + 2, (N + 1)(N + 2)/2 \rfloor \), where \( N \) is the dimension of the vector \( x \). If not given a default value of their mean is used.

Abstract: Finds unconstrained minimum of \( f \) using the NEWUOA software package. (Powell, 2004)

\( \ast \text{imethod = } 2 \)
options =<< \ldots; bounds >>, additional option for the solver,
- \( bounds, \) two column real matrix with number of rows equal to size of \( x0 \), where \( \text{bound}_{i,1} \leq x_i \leq \text{bound}_{i,2} \) (8.28)

Abstract: Finds constrained minimum of \( f \) using the BOBYQA software package. (Powell, 2004)
Abstract: Finds constrained minimum of \( f \) using the LINCOA software package. (Powell, 2004)

\[
\text{constraints}_i \leq \text{constraints}(x^0, p)_i \leq \text{bounds}_i.
\]

(8.30)

Here the original Powell’s formulation of bounds is allowed, where bounds is a single column real vector, and the constraints read

\[
\text{bounds}_i \leq \text{constraints}(x^0, p)_i.
\]

(8.31)

Abstract: Finds constrained minimum of \( f \) using the COBYLA software package. (Powell, 2004)

Result: \( x \), real vector.

(2) Format: \( x = \text{findmins}(f, df, /p/, x^0/, options/) \)

Arguments:

1. \( f = \text{function}(x/, p/) \), scalar dimensional function being minimized;
2. \( df = \text{function}(x/, p/) \), its jacobian;
3. \( p \), parameter passed directly to the functions;
4. \( x^0 \), real vector, initial guess;
5. \( options =<< \text{imethod}; \ldots >> \), a list, see below.

Result: \( x \), real vector.

Abstract: The function finds an unconstrained minimum of \( f \) using its jacobian \( df = df/dx \). It uses the GSL library of optimizers, the package CONMAX by [Kaufman Jr. et al. (December 1996)], and (Proxymal Bundle) NDA package for non-smooth optimization by [Luksan and Vlcek (2001)]. The entry \( \text{imethod} \) in the list \( options \) determines which method is used, where

- The GSL methods:
  - \( \text{imethod} = 0 \), for Fletcher-Reeves conjugate gradient method, (default);
  - \( \text{imethod} = 1 \), for Polak-Ribiere conjugate gradient algorithm;
  - \( \text{imethod} = 2 \), for Broyden-Fletcher-Goldfarb-Shanno conjugate gradient algorithm;
  - \( \text{imethod} = 3 \), for the steepest descent method;

for all of which \( options =<< \text{imethod}; \text{eabs}; \text{h}; \text{maxi}; \text{stdout}; \text{tol} >> \), where
* `eabs` (default 1e-4) is used to test the gradient for convergence,
* `h` (default 0.01) is the initial step size,
* `tol` (default 1e-4) is the tolerance used to determine next step in the iteration (measures how non-ortogonal the next step may be compared to the previous one),
* `maxi` is the maximal number of iterations in which the minimum is sought,
* `stdout` determines where the run-time messages are posted (terminal or file).

– CONMAX (`imethod = 4`), for which

options =<< imethod; convf; istep; maxi; nrkstep; stdout; tolcon >>>, where

* `convf` = `[encsm, linsm]` are the minimum relative change in error norm, and the number of steps in which it has to be achieved in order for iteration to continue (to avoid dwelling of the algorithm near shallow minima);
* `istep` (default 0) determines type of next step search: 0 for RK and successive linear programming (SLP) with a trust region, 1 for SLP only, 2 for RK only;
* `maxi`, a maximal number of iterations allowed to find the minimum;
* `nrkstep` (default 1) is the number of the steps in the underlying Runge-Kutta (RK) for each iteration;
* `stdout` is where the messages go (file or console);
* `tolcon` is the tolerance in fixing the constraint.

– Proximal bundle:

* PBUN (`imethod = 5`), basic algorithm;
* PNEW (`imethod = 6`), basic algorithm with Newton step and numerical approximation to Hessian.

Here, options =<< imethod; convx; convf; eta; maxi; met; mes; tol; stdout >>>, with

* `convx` = `[tolx, mtesx]` is the error in `x` and maximal number of iterations to reach it;
* `convf` = `[tolf, mtesf]`, same for the value of function being optimized;
* `eta`, a distance measure parameter;
* `maxi`, maximal number of iterations allowed (internally, this is equal to maxfv, maximal number of function evaluations allowed);
* `met` = 1, 2, 3, the weights updates (quadratic, local min, quasi-Newton condition);
* `mes` = 1, 2, 3, 4, interpolation selection in line search (bisection, two point quadratic, three point quadratic, three point cubic);
* `tol` = `[tolg, told, tols, tolp]`, different tolerances in convergence criterion, where `tolg` is for the Lagrangian function gradient, `told` is for descent direction, `tols` is for function decrease in a line search, `top` for a significant modification of the next line search direction.

Note: In the list options if an entry is set to zero this makes code use the default value.

### 8.11.4 conmins

Format: `x = conmins(f, df, *, *, /p/, x0, /F/, /options/)`

Arguments:

1, 2 f = `function(x, p/)` and its jacobian df = `function(x, p/)`, where df_{i,j} = \frac{\partial f_i}{\partial x_j}, the function being optimized;
### 8.11. MINIMIZATION OF A REAL FUNCTION OF A REAL VECTOR VARIABLE

3.4 three possibilities:

..g,dg.. \( g = \text{function}(x/p) \) and its jacobian \( dg = \text{function}(x/p) \), where \( dg_{i,j} = \partial g_i/\partial x_j \), the function representing the constraints, \( g(x) \leq 0 \);

..a,b.. this is for linear constraints, expressed as \( a \cdot x \) in \( b \), where \( b \) is a two column matrix, \( a \) is a matrix with the same number of columns as in \( x \), and the constraint is expressed as

\[
\begin{align*}
    b_{i,1} & \leq (a \cdot x)_i \leq b_{i,2} \\
\end{align*}
\]  

\( (8.32) \)

, , i.e., explicitly omitted any constraints, then this puts solvers in unconstrained mode. The advantage of this compared to using \texttt{findmins} is that it is possible to solve unconstrained problems of the first and the second type for any dimension of optimization function \( f \) (\texttt{findmins} allows only \( \text{dim } f = 1 \)).

5 \( p \), a parameter for the functions \( f, g, df, dg \), passed directly to the functions, if not needed has to be explicitly omitted;

6 \( x_0 \), an initial guess ;

7 \( F \), the optimization goal, can be omitted;

8 \( \text{options} = \langle \langle \text{imethod}; \ldots \rangle \rangle \), a list which allows fine tuning of the performance of the solver, can be omitted;

Observe, both \( F \) and \( \text{options} \) can be dropped from the list of arguments, then the default solver CONMAX is assumed.

**Result:** a vector \( x \).

**Abstract:** The solvers under this function perform constrained minimization, depending on the size and type of the problem, where

1. for nonlinear constraints and any dimension of the optimization function \( f \) and any type of optimization problem, the solver CONMAX (\text{imethod} = 0, default solver) is available.  
   N.B.: code checks whether the constraints are linear or not. If they are not, it defaults to CONMAX;

2. for linear constraints and any dimension of the optimization function \( f \), the solvers CONMAX and Proximal Bundle PMIN (\text{imethod} = 1, 2, 3) are available. While solver CONMAX can handle mixed type of optimization problem, PMIN can handle only a single type.  
   N.B.: code checks whether the optimization problem is of single or of mixed type. If it is a mixed problem, it defaults to CONMAX;

3. for linear constraints and one dimensional optimization function \( f \), the solvers CONMAX and Proximal Bundle PBUN, PNEW and PMIN (\text{imethod} = 1, 2, 3, in that order) are available.  
   N.B.: for multidimensional problem, all proximal bundle methods default to PMIN, unless the problem is mixed (see above).

Recall, there are two types of optimization problems,

I the first type, or \texttt{minmax}: find \( x \) that minimizes \( \max_i f_i(x) \);

II the second type: find \( x \) that minimizes \( \max_i |f_i(x) - F_i| \), where \( F_i \) is called an optimization goal.
User can choose the type of optimization problem, by setting or omitting $F$, where if $F$ is given then: if for some $i$ if $F_i = \inf()$ this implies that the $i$-th constraint is of the first type, while for $-\inf() < F_i < \inf()$ it is of the second type.

**Note:** A solver PMIN from the proximal bundle package uses recursive quadratic programming with the BFGS variable metric update.

**Note:** In the test suite, downloadable from rlabplus.sourceforge.net there are 5 scripts that give an overview of optimization solvers and their performance. These are located in directory test/optrootfit, and have cryptic names `eg_min1.r` through `eg_min5.r`. Use these scripts as templates for your own optimization problems.

**Note:** The test problem for all solvers is finding a minimum of a paraboloid in two dimensions. The reason why proximal bundle appears to be the fastest is because it uses quadratic updates in search (takes $\sim 7$ iterations to hit the minimum. A better test is probably needed to get the better idea about the comparative performance of the solvers.
8.11. MINIMIZATION OF A REAL FUNCTION OF A REAL VECTOR VARIABLE

**Example:** Finding a minimum of a parabolic function.

```plaintext
// paraboloid with minimum at (p[1],p[2])
F = function(x, p)
{
  return 10*(x[1]-p[1]).^2 + 20*(x[2]-p[2]).^2 + p[3];
};
// derivative of the paraboloid function
dF = function(x, p)
{
  return [20*(x[1]-p[1]); 40*(x[2]-p[2])];
};
p = [1;2;30]; // function parameters
x0 = [-5;7]; // initial guess
NITER = 100; // this is to test for segmentation faults
// parameters for f method
abserr = 1e-3;
ss = [2;1]; // initial size of the box per dimension
ostream = "test.dat";
maxiter = 1000;
options = <<>>;
options.eabs = abserr;
options.ss = ss;
options.stdout = ostream;
options.maxi = maxiter;
for( i in 1:NITER)
{
  x1=findmins(F, p, x0, options);
}
// parameters for fdf methods
abserr = 1e-3;
tol = 1e-4;
h = 0.01; // step size for fdf
ostream = "test.dat";
maxiter = 1000;
options = <<>>;
options.eabs = abserr;
options.tol = tol;
options.h = h;
options.stdout = ostream;
options.maxi = maxiter;
```
for(i in 1:NITER)
{
    for (j in 0:6)
    {
        options.imethod = j;
        x2 = findmins(F, dF, p, x0, options);
        //x2 = findmins(F, dF, p, x0, options);
    }
}
8.12 Statistics

Since rlab 2.3.1.n, $n > 0$, certain algebraic and statistical operations are implemented on dense matrices, as well, as on statistical lists, $\langle \text{val}; \text{wgt} \rangle$. Consider an example

```matlab
>> x = [1:10]';
>> w = ones(x);
>> y = x; flipud(y);
>> z = [[\text{val}=x;wgt=w]] .* y;
[x,w,y,z.val, z.wgt]
1 1 10 10 0.01
2 1 9 18 0.012345679
3 1 8 24 0.015625
4 1 7 28 0.0204081633
5 1 6 30 0.0277777778
6 1 5 30 0.04
7 1 4 28 0.0625
8 1 3 24 0.111111111
9 1 2 18 0.25
10 1 1 10 1
```

The algebraic operations that work with statistical lists are

- **addition**, or “+”;
- **subtraction**, or “-”;
- **element multiply**, “*”;
- **left and right element division**, or “/” and “\”;
- **matrix multiply**, “*”;
- **element power**, “.ˆ”

In operations between statistical lists and “normal” scalars, the later are assumed to have infinite weights. Here, matrices \textit{val} and \textit{wgt} have to obey certain conditions to be able to make statistical list:

1. weight matrix \textit{wgt} cannot have entries that are zero, infinity or \textit{nan};
2. matrix of values have to be provided. If weight matrix \textit{wgt} is not specified, the weight calculation is not performed;
3. the dimensions of weight matrix \textit{wgt} have to be less than or equal to dimension of value matrix \textit{val}. If the two dimensions differ the matrix optimization is used to match weights to values.

The formulae for entries in \textit{wgt} of the result of an operation are calculated assuming that the relationship between standard deviation $\sigma$ of a value and its weight $w$ is,

$$w = \frac{1}{\sigma^2}.$$ (8.33)
8.12.1 \textbf{isstat}

Format: \( i = \text{isstat}(y) \)

Arguments:

1. \( y \), anything.

Abstract: Returns 1 if \( y \) is statistical list, 0 otherwise (even if \( y \) does not exist).

8.12.2 \textbf{mean}

Format: \( y = \text{mean}(x/, \text{opts}/) \)

Arguments:

1. \( x \), real matrix, or, statistical list \( \ll \text{val}; \text{wgt} \gg \); or, 1-D histogram list \( \ll \text{bin}; \text{range} \gg \);

2. \( \text{opts}=\ll \text{row\_dominant}; \text{ignore\_inf}; \text{use\_datum}; \text{ignore\_datum}; \text{factor} \gg \), list of options applicable if \( x \) is a real matrix, where

   - \( \text{row\_dominant} \), integer 0 or 1, whether to consider as a single datum the columns (\( \text{row\_dominant}=1 \)) - that is, find average row-by-row, or the rows (\( \text{row\_dominant}=0 \)) - that is, find average column-by-column (default);
   - \( \text{ignore\_inf} \), integer 0 or 1, whether to ignore infinities as matrix entries. Default is \( \text{ignore\_inf}=1 \), that is, they are ignored;
   - \( \text{factor} \), integer array containing indices of columns (\( \text{row\_dominant}=0 \)) or rows (\( \text{row\_dominant}=1 \)) which mean will be calculated, while the rest will be filled with NaN’s. The default value is all data.
   - \( \text{use\_datum} \), integer array containing indices of rows (\( \text{row\_dominant}=0 \)) or columns (\( \text{row\_dominant}=1 \)) over which to calculate the mean. The default value is the entire range of data.

   Please note: One array applies to entire matrix;
   - \( \text{ignore\_datum} \), integer array containing indices of rows (\( \text{row\_dominant}=0 \)) or columns (\( \text{row\_dominant}=1 \)) which are to be excluded in calculating the mean. The default value is an empty set.

   Please note: One array applies to the entire data matrix \( x \).

   Please note: one cannot simultaneously supply \( \text{ignore\_datum} \) and \( \text{use\_datum} \). If one does anyway, the latter is ignored.

Result: \( y \), real scalar or vector.

Abstract: Finds mean of \( x \).

8.12.3 \textbf{var}

Format: \( y = \text{var}(x/, \text{opts}/) \)

Arguments:

1. \( x \), real matrix, or, statistical list \( \ll \text{val}; \text{wgt} \gg \); or, 1-D histogram list \( \ll \text{bin}; \text{range} \gg \);

2. \( \text{opts}=\ll \text{row\_dominant}; \text{ignore\_inf}; \text{use\_datum}; \text{ignore\_datum}; \text{factor}; \text{bias}; \text{mean} \gg \), list of options applicable if \( x \) is a real matrix, where
• \textit{row\_dominant}, integer 0 or 1, whether to consider as a single datum the columns (row\_dominant=1) - that is, find average row-by-row, or the rows (row\_dominant=0) - that is, find average column-by-column (default);
• \textit{ignore\_inf}, integer 0 or 1, whether to ignore infinities as matrix entries. Default is ignore\_inf=1, that is, they are ignored;
• \textit{factor}, integer array containing indices of columns (row\_dominant=0) or rows (row\_dominant=1) which mean will be calculated, while the rest will be filled with NaN’s. The default value is all data.
• \textit{use\_datum}, integer array containing indices of rows (row\_dominant=0) or columns (row\_dominant=1) over which to calculate the mean. The default value is the entire range of data.
Please note: One array applies to entire matrix;
• \textit{ignore\_datum}, integer array containing indices of rows (row\_dominant=0) or columns (row\_dominant=1) which are to be excluded in calculating the mean. The default value is an empty set.
Please note: One array applies to entire matrix.
Please note: one cannot simultaneously supply ignore\_datum and use\_datum. If so, the latter is ignored.
• \textit{bias}, integer 0 or 1, for unbiased (bias=0) or biased (default, bias=1) variance;
• \textit{mean}, scalar or array of length same as dimension of data (depends on row\_dominant). In combination with bias provides variance with mean (bias=0) or fixed mean (bias=1).

\textbf{Result}: \(y\), real scalar or vector.

\textbf{Abstract}: Finds variance of \(x\).

\textbf{8.12.4 min,max,minmax,mini,maxi}

\textbf{Format}: \(y = \text{min}(x/, opts/)\)

\textbf{Arguments}:

1. \(x\), real matrix;
2. \(\text{opts} = \langle \text{row\_dominant}; \text{ignore\_inf}; \text{flat}; \text{reject\_above}; \text{reject\_below} \rangle\), list of options applicable if \(x\) is a real matrix, where
   • \textit{row\_dominant}, integer 0 or 1, whether to consider as a single datum the columns (row\_dominant=1) - that is, find average row-by-row, or the rows (row\_dominant=0) - that is, find average column-by-column (default);
   • \textit{ignore\_inf}, integer 0 or 1, whether to ignore infinities as matrix entries. Default is ignore\_inf=1, that is, they are ignored;
   • \textit{flat}, 0 or 1, assume \(x\) is a vector.
   • \textit{reject\_above}, real scalar, reject values above this value when searching for maximum or minimum.
   • \textit{reject\_below}, real scalar, reject values below this value when searching for maximum or minimum.

\textbf{Result}: \(y\), scalar or vector of same number of rows (row\_dominant=1) or columns as \(x\) (row\_dominant=0, default).

\textbf{Abstract}: Find minimum or maximum value or their positions of \(x\) within some limits (\textit{reject\_below} or \textit{reject\_above}, if given).
8.12.5 covar

Format: \( y = \text{covar}(x/., \text{opts}/) \)

Arguments:

1. \( x \), real matrix;

2. \( \text{opts} = \langle \text{row\_dominant; ignore\_inf; use\_datum; ignore\_datum; factor; bias; mean} \rangle \), list of options applicable if \( x \) is a real matrix, where
   
   - \( \text{row\_dominant} \), integer 0 or 1, whether to consider as a single datum the columns (\( \text{row\_dominant}=1 \)) - that is, find average row-by-row, or the rows (\( \text{row\_dominant}=0 \)) - that is, find average column-by-column (default);
   
   - \( \text{ignore\_inf} \), integer 0 or 1, whether to ignore infinities as matrix entries. Default is \( \text{ignore\_inf}=1 \), that is, they are ignored;
   
   - \( \text{factor} \), integer array containing indices of columns (\( \text{row\_dominant}=0 \)) or rows (\( \text{row\_dominant}=1 \)) which mean will be calculated, while the rest will be filled with NaN’s. The default value is all data.
   
   - \( \text{use\_datum} \), integer array containing indices of rows (\( \text{row\_dominant}=0 \)) or columns (\( \text{row\_dominant}=1 \)) over which to calculate the mean. The default value is the entire range of data.
     Please note: One array applies to entire matrix;
   
   - \( \text{ignore\_datum} \), integer array containing indices of rows (\( \text{row\_dominant}=0 \)) or columns (\( \text{row\_dominant}=1 \)) which are to be excluded in calculating the mean. The default value is an empty set.
     Please note: One array applies to entire matrix.
     Please note: one cannot simultaneously supply \( \text{ignore\_datum} \) and \( \text{use\_datum} \). If so, the latter is ignored.
   
   - \( \text{bias} \), integer 0 or 1, for unbiased (\( \text{bias}=0 \)) or biased (default, \( \text{bias}=1 \)) variance;
   
   - \( \text{mean} \), scalar or array of length same as dimension of data (depends on \( \text{row\_dominant} \)). In combination with \( \text{bias} \) provides variance with mean (\( \text{bias}=0 \)) or fixed mean (\( \text{bias}=1 \)).

Result: \( y \), real square matrix of size of dimension of data in \( x \).

Abstract: Finds the covariance matrix of \( x \). The data mean values can be provided in vector \( m \).

8.12.6 absdev

Format: \( y = \text{absdev}(x/., m/) \)

Arguments:

1. \( x \), real vector or matrix; or, list \( << \text{val; wgt} >> \) with vector entries \( \text{val} \) for the values and \( \text{wgt} \) as their weights;

2. \( m \), real vector, guessed means of the data in vector \( x \);

Result: For \( x \) a vector or list, \( y \) is a real scalar or vector of size of \( m \) containing the mean absolute difference of the vector data (with respect to given mean).
If \( x \) is matrix, then the mean absolute differences of its columns (with respect to provided mean) are returned.
Abstract: This function returns the absolute deviation of data, from its own mean or from the mean given as the real vector \( m \),

\[
\text{absdev}(x/, m/) = \frac{1}{N} \sum |x_i - m|.
\]  

(8.34)

8.12.7 skew

Format: \( y = \text{skew}(x/, m, s/) \)

Arguments:

1. \( x \), real vector or matrix; or, list \( << \text{val}; \text{wgt} >> \) with vector entries \( \text{val} \) for the values and \( \text{wgt} \) as their weights;

2.3. \( s, m \), real scalars;

Result: \( y \), a real scalar.

Abstract: This function returns the skew of the data in \( x \) column-wise, either from its own mean and standard deviation, or from the mean \( m \) and standard deviation \( s \), given as the additional parameters,

\[
\text{skew}(x/, m, s/) = \frac{1}{N} \sum \left( \frac{x_i - m}{s} \right)^3.
\]  

(8.35)

8.12.8 kurtosis

Format: \( y = \text{kurtosis}(x/, m, s/) \)

Arguments:

1. \( x \), real vector or matrix; or, list \( << \text{val}; \text{wgt} >> \) with vector entries \( \text{val} \) for the values and \( \text{wgt} \) as their weights;

2.3. \( m, s \), real scalars;

Result: \( y \), a real scalar.

Abstract: This function returns the kurtosis of the data in the column-vector \( MDR \,*x \), either from its own mean and standard deviation, or from the mean \( m \) and standard deviation \( s \), as a square root of the variance, given as the additional parameters,

\[
\text{kurtosis}(x/, m, s/) = -3 + \frac{1}{N} \sum \left( \frac{x_i - m}{s} \right)^4.
\]  

(8.36)

8.12.9 median

Format: \( y = \text{median}(x) \)

Arguments:

1. \( x \), a real matrix with row-wise entries;

Result: \( y \), a real dense matrix.

Abstract: \textit{median} returns the median for each column of the matrix \( x \). where it is assumed that the columns are sorted in ascending order.
8.12.10 quantile

Format: \( y = \text{quantile}(x, q) \)

Arguments:

1. \( x \), real vector;
2. \( q \), real bounded vector, \( 0 \leq q_i \leq 1, \forall i \).

Result: \( y \), real vector.

Abstract: Finds quantile of data in vector \( x \). Tolerates NaN’s if they are sorted “on top,” or “on bottom” (default rlab behavior).

Permutations and combinations

In rlab a permutation is an integer array of length \( n \) containing each of the digits in range \( 1, \ldots n \) once, while a \( k \)-combination of an \( n \)-set is an integer array of length \( k \) where each digit is in range \( 1, \ldots n \) and appears only once.

8.12.11 invperm

Format: \( iy = \text{invperm}(ix) \)

Arguments: a permutation (integer array) \( ix \)

Result: a permutation \( iy \), an integer array.

Abstract: The function finds an inverse permutation of the permutation \( ix \), so that \( ix \cdot iy = 1 \).

8.12.12 revperm

Format: \( iy = \text{revperm}(ix) \)

Arguments: a permutation (integer array) \( ix \)

Result: a permutation \( iy \), an integer array.

Abstract: The function reverses the permutation \( ix \).

8.12.13 nextperm

Format: \( iy = \text{nextperm}(ix) \)

Arguments: a permutation (integer array) \( ix \)

Result: a permutation \( iy \), an integer array.

Abstract: The function finds the next lexicographic permutation of a permutation \( ix \).
8.12.14 prevperm
Format: $iy = \text{prevperm}(ix)$
Arguments: a permutation (integer array) $ix$
Result: a permutation $iy$, an integer array.
Abstract: The function finds the previous lexicographic permutation of a permutation $ix$.

8.12.15 validperm
Format: $i = \text{validperm}(ix)$
Arguments: a permutation (integer array) $ix$
Result: integer $i = 0, 1$
Abstract: The function tests whether an integer array $ix$ is a valid permutation or not.

8.12.16 mulperm
Format: $iz = \text{mulperm}(ix, iy)$
Arguments: two permutations (integer arrays) $ix$ and $iy$
Result: a permutation $iz$, an integer array.
Abstract: The function finds the composition of two permutations $iz = ix \cdot iy$.

8.12.17 nextcomb
Format: $iy = \text{nextcomb}(n, ix)$
Arguments: a $k$-combination of an $n$-set (integer array) $ix$
Result: a $k$-combination of an $n$-set $iy$, an integer array.
Abstract: The function finds the next lexicographic combination of a combination $ix$.

8.12.18 prevcomb
Format: $iy = \text{prevcomb}(n, ix)$
Arguments: a $k$-combination of an $n$-set (integer array) $ix$
Result: a $k$-combination of an $n$-set $iy$, an integer array.
Abstract: The function finds the previous lexicographic combination of a combination $ix$.

8.12.19 validcomb
Format: $i = \text{validcomb}(n, ix)$
Arguments: a $k$-combination of an $n$-set (integer array) $ix$
Result: $i = 0, 1$
Abstract: The function tests whether $n, ix$ is a valid $k$-combination of an $n$-set.
8.12.20  ncomb

Format:  \( y = ncomb(n, k) \)

Arguments:  integers \( n, k \)

Result:  double \( y \), where

\[
y = \binom{n}{k} = \frac{n!}{k! (n-k)!}
\]  \hspace{1cm} (8.37)
8.13 Random Number Generators

In rlabplus there are four types of random number generators (RNG): integer, continuous, discrete and histogrammic. Here, integer random number generators (IRNG) are the basic ones, and they are then used for generating continuous random numbers or for sampling from discrete or histogrammic distributions.

For each group there are two rlabplus functions, one for setting the parameters of a generator (irng, rng, drng, hrng), and the other for sampling from it (irand, rand, drand, hrand). For integer and continuous generators, the generators are further subdivided into user and system generators. Here, the former can be sampled by user, while the latter cannot. However, all generators are configurable by user. The separation may be useful for high-level functions that require a dedicated generator (e.g., monte-carlo integration, simulated annealing).

All function rely on the Gnu Science Library for performing various tasks.

Integer Random Number Generators

8.13.1 irng

I Format: irng(i/, opts/)

Arguments:

1  i, integer, index of the generator, where \(i \in \{1, \ldots, 64\}\), of which the generator number \(\{1, \ldots, 32\}\) can be sampled by user, while those \(\{33, \ldots, 64\}\) can be sampled by system, only;

2  opts list with entries \(\langle \text{name, seed, offset, range} \rangle\), where

The GSL team recommends "taus2", "gfsr4" (default) or "mt19937";

* seed, integer scalar, number to be used in initialization of the generator. If omitted local source of entropy /dev/urandom is used first time the generator is initialized. If same sequence of random number is to be generated by a script, the script should set value of seed;

* offset, range, integer scalars, for limiting the random integeres to range between offset and offset + (range - 1).
Note: offset and range apply only to generators that are directly sampled by user. The generators that are used by other solvers ignore them.
Note: Useful if one needs random integers \(U\) in certain range, in which case using \(U\%range\) is a sign of ignorance.

II Format: irng(i)

Arguments:

1  i, integer, index of the generator, where \(i \in \{1, \ldots, 32\}\), only.
Abstract: Sets generator $i$ as a default IRNG.

III Format: irng()

Abstract: Prints the information about currently defined IRNGs.

8.13.2 irand

Format: $y = \text{irand}(x)$, or $y = \text{irand}(nr, nc)$, or $y = \text{irand}()$

Arguments:

1 $x$, matrix, or $nr$, integer;
2 $nc$, integer.

Result: $y$, single integer or integer matrix of size $x$, or of size $nr$-by-$nc$, sampled from the default IRNG.

8.13.3 irng_state

I Format: $x = \text{irng\_state}(i)$

Arguments:

1 $i$, integer, No. of an existing IRNG.

Result: $x$, list, with entries

- name, string, name of the IRNG;
- state, integer array, byte-wise state of the IRNG;

Abstract: gets the state of IRNG No. $i$.

II Format: $\text{irng\_state}(i,x)$

Arguments:

1 $i$, integer, in range $\{1, \ldots 64\}$;
2 $x$, list, contains the name and state of the generator, see above.

Abstract: sets the IRNG No. $i$ to the generator which name and state are given in list $x$.

Allows saving, restoring or duplication of an IRNG.

Note: Internal GSL representation of the random integers is unsigned long int. These are stored internally in rlab as long int. One may convert between the two formats:

```plaintext
>> format(12,12);
>> x = irand()
-603153703
>> text(x,"%u")
3691813593
```
Note: If more than 64 IRNGs are required, or if 32 user IRNGs is not enough, in `gsl_rng.c` locate the following lines and change them accordingly, then rebuild `rlab`.

```c
#define RLAB_IRNG_MAXNO 64
#define RLAB_IRNG_MAXUSERNO 32

#define RLAB_IRNG_SIMAN (RLAB_IRNG_MAXNO-6)
define RLAB_IRNG_MC (RLAB_IRNG_MAXNO-5)
define RLAB_IRNG_HRNG (RLAB_IRNG_MAXNO-4)
define RLAB_IRNG_DRNG (RLAB_IRNG_MAXNO-3)
define RLAB_IRNG_SAMPLE (RLAB_IRNG_MAXNO-2)
define RLAB_IRNG_SHUFFLE (RLAB_IRNG_MAXNO-1)
define RLAB_IRNG_UNIFORM (RLAB_IRNG_MAXNO)
define RLAB_GSL_IRNG_DEFAULT gsl_rng_gfsr4
```
Continuous Random Number Generators

8.13.4 rng

I Format: rng(i, pdf, param/, j/)

Arguments:

1 i, integer, No. of RNG, in range 1, ..., 64, of which 1, ..., 32 can be sampled by user, while 33, ..., 64 can be sampled by system, only.

2 pdf, string, name of the continuous distribution function, see below;

3 param, real vector, the parameters for the particular distribution, see below;

4 j, integer, No. of IRNG which will be used in sampling of the desired distribution, if not given the default IRNG will be assigned.

Abstract: Configures one of many random number generators. Available are:

A. Continuous RNG and their pdf, param:

“normal”, [µ, σ], for Gaussian distribution with mean µ and standard deviation σ;

“normaltail”, [µ, σ, a], for random deviates from the upper tail of Gaussian distribution with mean µ and standard deviation σ, and where returned values are larger than the lower limit a, which has to be greater than µ, a > µ;

“exp”, µ, for exponential distribution with mean µ.

“laplace”, [µ, a], for Laplace distribution with mean µ and width a;

“exppow”, [µ, a, b], for random variates from the exponential power distribution with mean µ, scale parameter a and exponent b;

“cauchy”, [µ, a], for random variates from the Cauchy (Lorentz) distribution with mean µ and scale parameter a;

“rayleigh”, σ, for random variates from Rayleigh distribution with scale parameter σ.

“rayleightail”, [a, σ], for random variates from Rayleigh distribution with scale parameter σ and lower limit a.

“levy”, [µ, c, α], for random variates from Levy symmetric stable distribution, with mean µ, scale c and exponent α, where 0 < α ≤ 2.

“levyskew”, [µ, c, α, β], for random variates from Levy skew alpha-stable distribution, with mean µ, scale c, exponent α and skewness β.

“gamma”, [a, b], for random variates from the gamma distribution with exponent a and scale b.

“uniform”, [xlo, xhi], for uniform random variates in xlo-to-xhi interval.

“lognormal”, [ζ, σ], for random variates from lognormal distribution, with log-mean ζ, and standard deviation σ.

“chisq”, ν, for random variates from χ² distribution with ν degrees of freedom.

“F”, [ν₁, ν₂], for random variates from F-distribution with degrees of freedom ν₁ and ν₂.

“t”, ν, for random variates from t-distribution with ν degrees of freedom.

“beta”, [a, b], for random variates from beta distribution, with parameters a and b.

“logistic”, [µ, a], for random variates from logistic distribution with mean µ and scale a.
“pareto”, \([a, b]\), for random variates from Pareto distribution of order \(a\), where \(b\) is the lower limit of the variates.
“weibull”, \([a, b]\), for random variates from Weibull distribution with scale \(a\) and exponent \(b\).
“gumbel1”, \([a, b]\), for random variates from Gumbel distribution of type-1 with scale \(a\) and exponential scale \(b\).
“gumbel2”, \([a, b]\), for random variates from Gumbel distribution of type-2 with exponent \(a\) and exponential scale \(b\).

B. Discrete (integer) RNG and their pdf, \(param\):
“poisson”, \(\mu\), for random integer variates from the Poisson distribution with mean \(\mu\).
“binomial”, \([p, n]\), for random integer variates from the binomial distribution, with a trial probability \(p\) and \(n\) being total number of trials.
“negbinomial”, \([p, n]\), for random integer variates from the negative binomial distribution, for number of failures \(k\) before \(n\) successes. It is similar to binomial distribution, but \(n\) need not be an integer. If \(n\) is an integer than the distribution is called Pascal’s.
“geometric”, \(p\), for random integer variates from the geometric distribution, for number of trials with probability \(p\) until the first success.
“hypergeom”, \([n_1, n_2, t]\), for random integer variates from the hypergeometric distribution, for getting a number of successes from the first batch with \(n_1\) elements, and the rest the second batch with \(n_2\) elements, to a number of draws given as \(\max\{t, t - n_2, n_1\}\).
“log”, \(p\), for random integer variates from the logarithmic distribution, with probability \(p\).

II Format: \(rng(i)\)

1 \(i\), integer, No. of RNG, in range 1,\ldots32.

Abstract: Declares the random number generator No. \(i\) the default one. Subsequent executions of \(rand()\) will sample from this generator.

III Format: \(rng()\)

Abstract: Writes the information about current RNGs to the konsole.

8.13.5 rand

Format: \(y = rand(nrow, ncol)\), or \(y = rand(x)\)

Arguments:

1 \(nrow\), integer, number of rows, or \(x\), matrix;

2 \(ncol\), integer, number of columns;

Result: \(y\), real matrix of size \(x\), or of size \(nrow\)-by-\(ncol\), that contains real numbers sampled from the default RNG.

Note: The default is a uniform distribution on \([0,1]\) interval.
8.13.6 randomize

I Format: \( \text{randomize}(x, d, \text{opts}) \),

Arguments:

1. \( x \), dense matrix, real or complex;
2. \( d \), vector, real or complex. See \( \text{opts} \);
3. \( \text{opts} \), list, with entries:
   * \( \text{bandwidth}_\text{lower}, \text{bandwidth}_\text{upper} \), integers, lower and upper bandwidth of the matrix \( x \);
   * \( \text{sparsity} \), scalar in range \( (0, 1) \), sparsity of matrix \( x \) (probability that a matrix element is zero);
   * \( \text{symmetric}, \text{hermitean} \), integer, 0 or 1, determines whether \( x \) is symmetric or not;
   * \( \text{singular-values} \), integer, 0 or 1, determines whether \( d \) is the diagonal of the matrix \( x \), or represents its singular values. Default is diagonal.

Abstract: Given \( d \) or \( \text{opts} \) replaces the content of \( x \) with a matrix of specified properties, e.g., with given singular values, or with a given diagonal.

Note: \( \text{randomize} \) is based on testing routines from \texttt{lapack}, v. 3.0, where the LAPACK’s RNG is linked to the dedicated \texttt{rlab}’s uniform RNG, No. 64, which in turn uses IRNG No. 64 (which user can tweak).

II Format: \( \text{randomize}(x) \),

Arguments:

1. \( x \), dense matrix, real or complex;

Abstract: Fills the entries of \( x \) with the samples from default RNG.

8.13.7 urandom

Format: \( y = \text{urandom}(\text{nrow}, \text{ncol}) \), or \( y = \text{urandom}(x) \)

Arguments:

1. \( \text{nrow} \), integer, number of rows, or \( x \), matrix;
2. \( \text{ncol} \), integer, number of columns;

Result: \( y \), real matrix of size \( x \), or of size \( \text{nrow} \)-by-\( \text{ncol} \), that contains real numbers sampled from the uniform distribution on \( [0, 1] \) interval. It is a system RNG No. 64, and it uses IRNG No. 64, as well.

8.13.8 gaussian

Format: \( y = \text{gaussian}(\text{nrow}, \text{ncol}) \), or \( y = \text{gaussian}(x) \)

Arguments:

1. \( \text{nrow} \), integer, number of rows, or \( x \), matrix;
2 \( ncol \), integer, number of columns;

**Result:** \( y \), real matrix of size \( x \), or of size \( nrow \)-by-\( ncol \), that contains real numbers sampled from the gaussian distribution with mean 0 and standard deviation 1. It is a system RNG No. 63, and it uses IRNG No. 63, as well.
Example: Basic structure of RNGs and IRNGs after starting rlab and calling `urandom` once:

```plaintext
>> urandom()
   0.63423978

>> irng()
Currently defined integer random number generators :
   system  64: gfsr4, size =  65540
   (*) is a user-default generator.
       1

>> rng()
Currently defined random number generators :
   system  64: uniform, with irng no. 64
       nparam = 2
       p[1] = 0
       p[2] = 1
   (*) is a user-default generator.
       1

>>
```
Discrete Random Number Generators

8.13.9  drng

I  Format:  \texttt{drng(i,x,w,j/)}

Arguments:

1  \(i\), integer, index of the discrete generator in range 1\ldots32;
2  \(x\), real, complex, or character vector;
3  \(w\), real positive vector of relative weights;
4  \(j\), integer, index of the IRNG to be used for sampling (user or system).

Abstract:  The function generates a look-up table for the \(i\)-th discrete random generator. Vector \(x\) provides the set from which the entries are sampled with the relative weights given in vector \(w\).

II  Format:  \texttt{drng(i)}

Arguments:

1  \(i\), integer, index of the discrete generator in range 1\ldots32;

Abstract:  Sets DRNG No. \(i\) as a default DRNG.

III  Format:  \texttt{drng()}

Abstract:  Prints the information about available DRNGs.

Note:  If more than 32 DRNGs are needed, then change

\begin{verbatim}
#define RLAB_GSL_DRNG_SIZE 32
\end{verbatim}

in \texttt{gsl_rnd.c} to desired value and recompile.

Note:  If IRNG for DRNG is not supplied, then a system IRNG No. 61 is used.

8.13.10  drand

Format:  \(y = drand()\), or \(y = drand(nrow,ncol)\), or \(y = drand(x)\)

Arguments:

1,2  \(nr,nc\), integers, or \(x\), matrix.

Result:  \(y\), matrix of size \(nr\)-by-\(nc\), or of size of \(x\), or 1-by-1, the entries of which contains the samples from the default DRNG.

\begin{verbatim}
>> x = char([65:90]'); // use uppercase alphabet to sample from
>> w = ones(x);       // each character has the same weight
>> drng(1,x,w,12);    // set discrete random number generator No. 1, that uses
                     // IRNG No. 12
>> drand(1,7)         // sample 1x7 matrix from the drng No. 1
M A R I J A N
>>
\end{verbatim}
CHAPTER 8. ADVANCED GENERAL NUMERICAL PROCEDURES AND ALGORITHMS

Histogrammic Random Number Generators

8.13.11 hrng

I Format: \texttt{hrng}(i, range, bin/, j/)

Arguments:

1. $i$, integer, index of a histogrammic generator in range 1...32;
2. \texttt{range}, real vector, contains range according to the GSL histogram specifications;
3. \texttt{bin}, real vector, contains bin content according to the GSL histogram specifications;
4. $j$, integer, index of the IRNG to be used for sampling (user or system).

Abstract: Generates a probability density function based on a histogrammic variable of the GSL type, which entries \texttt{bin} and \texttt{range} are supplied.

II Format: \texttt{hrng}(i)

Arguments:

1. $i$, integer, index of a histogrammic generator in range 1...32;

Abstract: Sets HRNG No. $i$ as a default HRNG.

III Format: \texttt{hrng}()

Abstract: Prints information about available HRNGs.

Note: If more than 32 HRNGs are needed, then change

\begin{verbatim}
#define RLAB_GSL_HRNG_SIZE 32
\end{verbatim}

in \texttt{gsl_rnd.c} to desired value and recompile.

Note: If IRNG for HRNG is not supplied, then a system IRNG No. 60 is used.

8.13.12 hrand

Format: $y = hrand()$, or $y = hrand(nr, ncol)$, or $y = hrand(x)$

Arguments:

1,2 \texttt{nr, nc}, integers, or \texttt{x}, matrix.

Result: $y$, matrix of size \texttt{nr}-by-\texttt{nc}, or of size of \texttt{x}, or 1-by-1, the entries of which are sampled from the default HRNG.
8.13. RANDOM NUMBER GENERATORS

Other Functions

8.13.13 shuffle

Format: \( y = \text{shuffle}(x, n) \)

Arguments:

1. \( x \), matrix, row-wise collection of data.
2. \( n \), integer, \( n \geq x.nr \), number of data points.

Result: \( y \), matrix, contains \( n \) rows of data matrix \( x \) drawn without replacement.

Note: \( \text{shuffle}() \) uses a system IRNG No. 63.

8.13.14 sample

Format: \( y = \text{sample}(x, n) \)

Arguments:

1. \( x \), matrix;
2. \( n \), integer, in range \( 0, \ldots x.nr \).

Result: \( y \), matrix, contains rows of data matrix \( x \) drawn \( n \) times with replacement.

Note: \( \text{sample}() \) uses a system IRNG No. 62.

8.13.15 pdf, cpf

Format: \( y = \text{pdf.}[\text{dist}](x, p) \), \( y = \text{pdf.}[\text{dist}][[x, p]] \)

Format: \( y = \text{cpf.}[\text{dist}](x, p) \), \( y = \text{cpf.}[\text{dist}][[x, p]] \)

Arguments:

1. \( x \), real matrix;
2. \( p \), real matrix, rows of parameters for the probability distribution function \( \text{dist} \).

Result: \( y \), real matrix, probability distribution function or cumulative probability function for the distribution \( \text{dist} \).

Abstract: \( \text{pdf} \) exists for the following distributions: F, beta, binomial, cauchy, chisq, dirichlet, exp, exppow, gamma, geometric, gumbel1, gumbel2, hypergeom, laplace, log, logistic, lognormal, multinom, negbinomial, normal, normalbiv, normaltail, pareto, poisson, rayleigh, rayleightail, t, uniform and weibull, all from the GSL, and: dd.

\( \text{cpf} \) exists for the following distributions: F, beta, binomial, cauchy, chisq, exp, exppow, gamma, geometric, gumbel1, gumbel2, hypergeom, laplace, logistic, lognormal, negbinomial, normal, pareto, poisson, rayleigh, t, uniform, and weibull.
Note: \( dd \) stands for a drift-diffusion distribution of arrival times, with parameters \( \bar{t} \), the mean arrival time under drift, and \( r \), which is a ratio \( 4r^2 = t_\delta/\bar{t}^2 \), with \( t_\delta = L^2/\delta \), where \( \delta \) is a diffusion constant and \( L \) is a path length. The drift-diffusion distribution in 1-D is given by

\[
p(t) = \frac{|r|}{\sqrt{\pi t}} \exp\left(-r^2 \frac{(\bar{t} - t)^2}{t}\right)
\]  

(8.38)
Note: pdf and cpf are lists and not built-in functions. For that reason to access its entries within the user-defined functions, their names have to be specified as global within the functions,

```plaintext
myfunc = function(x, r)
{
    // r[1]->mu
    // r[2]->sigma
    // r[3]->p(x)
    global(cpf);
    // following formulations are equivalent:
    rval = cpf.normal(x, r[1], r[2]) - p[3]; // three parameters
    rval = cpf.normal(x, r[1,2]) - p[3]; // two parameters
    rval = cpf.normal( [x, r[1,2]] ) - p[3]; // single parameter
    return rval;
}
```

Now “myfunc” can be used for, say, inverse problem:

```plaintext
invnormal = function(p,mu,sigma)
{
    x0 = mu + sigma;
    params = [mu, sigma, p];
    return findroot(myfunc,params, x0);
}
```

Note: The inverse problem can be solved somewhat simpler if one uses findroot1 rather than findroot:

```plaintext
if (!exist(myfunc))
{
    myfunc = cpf.normal;
}

invnormal = function(p,mu,sigma)
{
    global(myfunc);
    if (p > 0.01 && p < 0.99)
    {
        xlo = mu - 3 * sigma;
        xhi = mu + 3 * sigma;
    }
    else
    {
        xlo = mu - 20 * sigma;
        xhi = mu + 20 * sigma;
    }
    params = [mu, sigma];
    return findroot1(myfunc, p, params, [xlo,xhi]);
}
```

Please recall that while findroot(func,...) solves func = 0, findroot1(func1,func2,...) solves func1 = func2, where func2 can be a scalar, as well.
8.14 Ordinary differential equations (ODE)

ODE Initial Value solvers

8.14.1 odeiv

I Format: \( y = \text{odeiv}(f,/p/,t,y_0/,options/) \)

Arguments:

1. \( f = \text{function}(t,y,/p/), \) function, specifies an ODE initial value problem
   \[
   y'(t) = f(t,y,/p/); \\
   \]
   \( (8.39) \)
2. \( p, \) entity, passed directly to \( f; \)
3. \( t = [t_1, \ldots t_N], \) real vector of size at least 2, points at which \( y(t) \) is to be evaluated;
4. \( y_0 = y(t_1), \) real vector, initial value;
5. \( \text{options} = \ll \text{imethod}; \text{ady}; \text{adydt}; \text{eabs}; \text{erel}; \text{h}; \text{phase_space}; \text{stdout} >>, \) list, its entries are
   * \( \text{imethod}, \) integer, chooses method for solving the ODE: 0 for \( \text{adams’} \) (default), and the GSL solvers: 1 for \( \text{rk2}, \) 2 for \( \text{rk4}, \) 3 for \( \text{rkf45}, \) 4 for \( \text{rkck45}, \) 5 for \( \text{rk8pd}, \) 6 for \( \text{rk2imp}, \) 7 for \( \text{rk4imp}, \) 8 for \( \text{gear1}, \) and 9 for \( \text{gear2} \) method.
   * \( \text{eabs} \) and \( \text{erel} \) (all) are the absolute and relative errors used in determining the step-size of integration;
   * \( \text{step}, \) positive scalar, (all) the initial step size;
   * \( \text{ady} \) and \( \text{adydt} \) (the GSL) are the ratios between increment in value of the function and in the value of the first derivative that enter the step size criterion,
     \[
     D_i = \text{eabs} + \text{erel} \cdot (\text{ady} \cdot |y_i| + \text{adydt} \cdot h \cdot |y'_i|); \\
     \]
     \( (8.40) \)
   * \( \text{stdout}, \) string, (all) filename where the run-time messages are posted.

II Format: \( y = \text{odeiv}(f,dfdyt,/p/,time,y_0/,options/), \)

Arguments:

1. \( f = \text{function}(t,y,/p/), \) function, see Eq. \( (8.39); \)
2. \( dfdyt = \text{function}(t,y,/p/), \) function, extended jacobian of \( f \) with respect to \( y \) and to \( t, \) where
   \[
   dfdyt[i;j] = \frac{\partial f_i}{\partial y_j}, \text{ for } i,j = 1 \ldots \text{neq}, \]
   \( (8.41) \)
   and
   \[
   dfdyt[i;\text{neq}+1] = \frac{\partial f_i}{\partial t}, \text{ for } i = 1 \ldots \text{neq}, \]
   \( (8.42) \)
3. \( p, \) entity, passed directly to \( f \) and \( dfdyt; \)
4. \( t = [t_1, \ldots t_N], \) real vector of size at least two, points at which \( y(t) \) is to be evaluated;
5. \( y_0 = y(t_1), \) real vector, initial values for \( y; \)
6. \( \text{options} = \ll \text{imethod}; \text{ady}; \text{adydt}; \text{eabs}; \text{erel}; \text{h}; \text{stdout}; \text{phase_space} >>, \) list, its entries are:
8.14. ORDINARY DIFFERENTIAL EQUATIONS (ODE)

* `imethod`, integer, chooses the method for solving the ODE: 0 for backward difference formulae, as implemented in code DVODE by Brown, Hindmarsh, and Byrne (1989), 1 for `blended implicate method` by Brugnano and Magherini (2005) (default), and 2 for the GSL solver `bsimp`;

* `eabs` and `erel`, positive scalars, (all) absolute and relative errors used in determining the step-size of integration;

* `step`, positive scalar, (all) initial step-size;

* `ady` and `adydt`, non-negative scalars, (the GSL) ratios between increment in value of the function and in the value of the first derivative that enter the step size criterion in Eq. (8.40).

* `stdout`, string, (all) filename to which the messages generated by the solver are posted.

**Result:** `y`, real matrix, contains `[t, y(t)]` evaluated at specified times.

**Abstract:** Solves a system of ODE (8.39).

**Note:** The ODE solvers have the following feature. If the function describing the first derivatives returns NaN’s then the calculation is immediately halted, and the solution for user’s specified mesh is returned up to the last point that produced non-NaN result.

**Note:** Option `phase_space` modifies the output of the solver, so it reports not just `[t, y(t)]` but also the time derivatives of the configuration variables, that is, `[t, y(t), y'(t)]`. 
ODE Boundary Value solvers

8.14.2 odebv

I basic boundary value problem solvers ACDC [Cash & Wright, 1995], MIRKDC [Enright & Muir 1999] and COLDAE [Ascher & Spieteri 1993].

Format: \( y = \text{odebv}(\text{dimu}, n\text{lb}c, f, df/,/p/, X, gL, gR, dgL, dgR/,/options/), \)

Arguments:

1. \( \text{dimu} \), integer, size of the problem \((\text{dimu} = \text{dim} f)\);
2. \( n\text{lb}c \), integer, dimension of the left boundary condition function \((n\text{lb}c = \text{dim} gL)\). Consequently, \( \text{dim} gR = \text{dimu} - n\text{lb}c \);
3. \( f, df = \text{function}(x, u/,/p/) \), function, specifies the ODE problem
   \[
   u' = f(x, u). 
   \]  
   and its Jacobian with respect to \( u \), \( df[i; j] = \partial f_i / \partial u_j \);
4. \( p \), entity, passed directly to \( f \) and \( df \);
5. \( X \), real matrix, passes the following information to the solver:
   - \( X = [a, b] \), provides left and right points of the interval on which the solution of ODEBV is sought;
   - \( X = [x_1, \ldots x_N] \), provides a mesh which will be improved upon in subsequent calculations (but which will be included in the solution’s mesh);
   - \( X = [x_1, u_1 \ldots u^{(\text{dimu}-1)} \ldots] \), provides an initial mesh but also an initial guess for the ODEBV solution which will be subsequently improved upon.
6-10. \( gL, gR, dgL, dgR = \text{function}(u/,/p/) \), functions, specify the boundary conditions
   \[
   gL(uL/,/p/) = 0, 
   gR(uR/,/p/) = 0, 
   \]  
   and their Jacobians, \( dgL, R[i; j] = \partial (dgL, R)_i / \partial u_j \).

11. \( \text{options} = << \text{imethod; tol; stdout} >>, \) list, with entries
   - \( \text{imethod} \), integer, 0 for ACDC, 1-3 for mirk221, mirk343 and mirk563, in that order, 4 for coldae (in colsys mode);
   - \( \text{tol} \), positive vector, tolerances for each of the component in the solution (thus \( \text{tol} \) is of the size \( \text{udim} \));
   - \( \text{stdout} \), string, filename to which the messages created by the solver are posted.

II boundary value problem with automatic continuation, ACDC

Format: \( y = \text{odebv}(\text{dimu}, n\text{lb}c, f, df/,/p/, X, gL, gR, dgL, dgR, EP/,/options/), \)

Arguments:

1. \( \text{dimu} \), integer, size of the problem \((\text{dimu} = \text{dim} f)\);
2. \( n\text{lb}c \), integer, dimension of the left boundary condition function \((n\text{lb}c = \text{dim} gL)\). Consequently, \( \text{dim} gR = \text{dimu} - n\text{lb}c \);
3.4 \( f, df = \text{function}(x, u, p, e) \), function, specifies the ODE problem

\[
u' = f(x, u, e).
\]  

(8.46)

and its jacobian with respect to \( u \), \( df[i; j] = \partial f_i / \partial u_j \);

5. \( p \), entity, passed directly to \( f \) and \( df \);

6. \( X \), real matrix, passes the following information to the solver:

\begin{itemize}
  \item \( X = [a, b] \), provides left and right points of the interval on which the solution of ODEBV is sought;
  \item \( X = [x_1, \ldots x_N] \), provides a mesh which will be improved upon in subsequent calculations (but which will be included in the solution’s mesh);
  \item \( X = [x_1, u_1 \ldots u^{(\text{dimu}-1)} \ldots] \), provides an initial mesh but also an initial guess for the ODEBV solution which will be subsequently improved upon.
\end{itemize}

7-10 \( gL, gR, dgL, dgR = \text{function}(u, p, e) \), function, specify the boundary conditions

\[
gL(uL, p, e) = 0,
\]  

(8.47)  

\[
gR(uR, p, e) = 0,
\]  

(8.48)

and their Jacobians, \( dgL, R[i; j] = \partial (dgL, R)_i / \partial u_j \).

11. \( EP = [e_{\text{max}}, e_{\text{min}}] \), pair of positive scalars, initial and final values for the continuation parameter \( e \), where \( 1 \geq e_{\text{max}} > e_{\text{min}} > 0 \);  

12. \( \text{options} = << \text{tol} ; \text{stdout} >> \), list,

\begin{itemize}
  \item \( \text{tol} \), positive vector, the tolerances for each of the component of the solution (thus \( \text{tol} \) is of the size \( \text{udim} \));
  \item \( \text{stdout} \), string, filename where the run-time messages are posted.
\end{itemize}
ODAE Initial Value solvers

8.14.3 odae

I Format: \( y = odae\left(f, df, p, m, t, y_0, yp_0, \text{options} / \right) \)

II Format: \( y = odae\left(f, df, p, t, y_0, yp_0, \text{options} / \right) \)

Arguments:

1,2 If \( m \) is given then \( f, df = function(t, y, p) \), otherwise \( f, df = function(t, y, yp, p) \), function and its jacobian with respect to \( y \) (and \( yp \)). If \( df \) is not given, it is calculated numerically internally;

3 \( p \), entity, parameter passed to the function;

4 \( m \), real matrix, mass matrix for the ODAE problem,

\[
m \cdot y'(t) = f(t, y).
\] (8.49)

If \( m \) is omitted, then the implicite ODAE problem is assumed,

\[
f(t, y, y') = 0.
\] (8.50)

5(4) \( t = [t_1, \ldots t_N] \), real vector, points at which \( y(t) \) is evaluated;

6(5) \( y_0 = y(t_1) \), real vector, initial values;

7(6) \( yp_0 = y'(t_1) \), real vector, initial values;

8(7) \( \text{options} = << \text{maxder; maxi; tol_abs; tol_rel; step; stdout} >> \), list with entries

- \( \text{imethod} \), integer, 0 for MEBDFI (default), 1 for DDASKR package, 2 for BiMD package. The packages MEBDFI and DDASKR can handle both Eq. (8.49) and (8.50), while BiMD only Eq. (8.49).

- \( \text{tol_abs} \) and \( \text{tol_rel} \), real scalar or vector of size \( \text{dim} y \), relative and absolute tolerances in determining the step-size of integration. Possibilities: (i), both scalars, \( rtol \times abs(y(i)) + atol \), (ii), scalar and vector, \( rtol \times abs(y(i)) + atol(i) \) or \( rtol(i) \times abs(y(i)) + atol \), or (iv) both vectors, \( rtol(i) \times abs(y(i)) + atol(i) \);

- \( \text{maxi} \), integer, maximal number of iteration to reach the tolerances;

- \( \text{order} \), integer, maximal order (for MEBDFI 1...7, for DDASKR 1...5, for BiMD 4,6,8,10,12);

- \( \text{index1}, \text{index2}, \text{index3} \), integer, the number of variables \( y \) of index 1,2 and 3, respectively, for solvers MEBDFI and BiMD. The system of equations (8.49) or (8.50) has to be defined so that index 1 variables precede index 2 variables, which in turn precede index 3 variables. If these are not given by the user somewhat brave assumption is made that \( \text{index1} = \text{dim} f \), \( \text{index2} = \text{index3} = 0 \);

- \( \text{stdout} \), string, filename where run-time messages are posted.

Abstract: Solves a system of ordinary differential-algebraic equations (ODAE) as an initial value problem (8.49) and (8.50). It uses solvers MEBDFI by Abdulla and Cash (1999), DDASKR by Petzold, Brown, Hindmarsh, and Ulrich (2006), and BiMD by Brugnano and Magherini (2005).

Note: choice between implicit or linear-explicit mode of the solver is based on whether the fourth argument \( (m \text{ or } t) \) is a square matrix \( (m) \) or a vector \( (t) \).
Sturm-Liouville Eigenvalue Problem with Boundary conditions

SLEIGN2 by [Bailey et al.] (2001) is a very powerful code for finding the eigenvalues and eigenfunctions of a single second order ODE using Sturm-Liouville theory. It can deal with separated and coupled boundary conditions. For serious use of the code the user is referred to the publications of the authors, most of which can be downloaded from their web-site, listed in the bibliography. The rlab interface sticks to the nomenclature introduced by the authors. Thus to fully utilize the code it is good idea to check their papers first. In order to use the functions within rlab user should be familiar with the Sturm-Liouville differential equation,

\[-(p(x) y'(x))' + q(x) y(x) = \lambda w(x) y(x),\]  

(8.51)

where the boundary conditions for \( y \) for \( x \in [a,b] \) can be,

- separated and regular,
  \[ A_1 y(a) + A_2 p(a) y'(a) = 0 \]
  \[ B_1 y(b) + B_2 p(b) y'(b) = 0 \]

  (8.52)

- separated and not regular,
  \[ A_1 [y,u](a) + A_2 [y,v](b) = 0 \]
  \[ B_1 [y,U](b) + B_2 [y,V](b) = 0 \]

  (8.53)

where \([f,g](x) = (f(p g') - (p f'))g\)(x) is the Wronskian for the Sturm-Liouville problem, Eq. (8.51).

- coupled: periodic, semi-periodic and general periodic,

  \[
  \begin{bmatrix}
  [y,U](b) \\
  [y,V](b)
  \end{bmatrix} = \exp(i \alpha) \mathbf{K} \begin{bmatrix}
  [y,u](a) \\
  [y,v](a)
  \end{bmatrix}
  \]

  (8.54)

The function odesl is actually a list, containing two functions, odesl.eign for separated boundary value problems and odesl.coup for coupled boundary value problems.

8.14.4 odesl

- Separated regular and not-regular boundary value problem

  **Format:** \( Y = \text{odesl.eign}(P,Q,W,/UV/,/p/,x,bc,eix/,nc,atab,options/) \)

  **Arguments:**

  1-3 \( P,Q,W = \text{function}(x/,p/) \), scalar functions specifying the Sturm-Liouville problem;
  4 \( UV = \text{function}(x/,p/) \), a vector function returning a vector of length 4 containing the functions used in Wronskian, \([u(x),u'(x),v(x),v'(x)]\);
  5 \( p \), a parameter passed directly to the functions;
  6 the mesh array \( x = [a/,x_1,\ldots,x_N/,b] \), where \( a \) and \( b \) are the boundary points (infinity included), and \( x_1,\ldots,x_N \) is the relative mesh in \([-1,1]\) (and can be omitted if only an eigenvalue is desired), scales so \(-1 \to a\) and \(1 \to b\), at which the eigenfunction will be calculated;
  7 \( bc = [A_1,A_2;B_1,B_2] \) are the boundary conditions, see Eqs. (8.52) and (8.53);
  8 \( eix \) is the index of the eigenvalue being sought;
  9 \( nc = [nca,ncb] \) is the type of the boundary points;
10 atab = [p0ata,qfata,p0atb,qfatb], tells what kind of behavior functions \( p \) and \( q \) have at the endpoints (equal to zero? finite?), affirmative carries 1, while negative -1. If not given the code tries to determine that on itself (it is capable of recognizing 0 and \( \infty \) as the result of computations).

11(10,9) options =<< tol; stdout >>, a list with tolerance \( tol \) used in computation of eigenvalue, and stdout where the run-time messages are posted.

If \( nc \) is not given the function assumes default regular behavior, \( nc = [1, 1] \).

**Result:** a list \( Y =<< eigval; eigfun; info; slfun >> \), where \( eigval \) is the eigenvalue with index \( eix \), \( eigfun \) is the eigenfunction on, now absolute, mesh in the form \([x, f(x)]\), and \( info \) is the message code from SLEIGN2. Output variable \( slfun \) is a 9-element array, containing the following information: \( slfun(1) \) - point where two pieces of eigenfunction \( Y \) match; \( slfun(2) \) - left endpoint of the (truncated) interval, \( x_a \); \( slfun(3) \) - value of \( \theta \) at left \( x_a \), where \( y(x_a) = \rho \cdot \sin(\theta) \); \( slfun(4) \) - value of \( f \) at \( x_a \), where \( \rho = \exp(f) \); \( slfun(5) \) - right endpoint, \( x_b \), of the (truncated) interval; \( slfun(6) \) - value of \( \theta \) at \( x_b \); \( slfun(7) \) - value of \( F \) at \( x_b \); \( slfun(8) \) - final value of integration accuracy parameter \( \epsilon \). \( slfun(9) \) - the constant \( Z \) in the polar form transformation.

**Note:** The variable \( slfun \) may be useful for an eigenfunction problem on semi-infinite or infinite domain, where its entries can be used in a formulation of a finite initial/boundary value problem for an \( odeiv/odebv(\ldots) \) integrator (provided that the function returned the eigenvalue).

- **Coupled boundary value problem**

  **Format:** \( Y = odesl.coup(P,Q,W,/[UV]/,[/p]/,x,bc1,be2,alfa,eix/,nc,atab/) \),

  **Arguments:** two new parameters compared to \( odesl.eign \), \( bc2 = [K_{11}, K_{12}; K_{21}, K_{22}] \) a matrix \( K \) for the RHS of Eq. (8.54) and \( alfa \), the phase factor next to it.

  **Result:** a list \( Y =<< eigval; eigfun; info; slfun >> \), where \( eigval \) is the eigenvalue with index \( eix \), \( eigfun \) is the eigenfunction on, now absolute, mesh in the form \([x, f(x)]\), and \( info \) is the message code from SLEIGN2.

**Note:** An interval \([a, b]\) can be finite, seminfinite or infinite. Use built-in function \( inf() \) to set an infinite boundary. In this case \( odesl \) cannot calculate the eigenfunction but only the eigenvalue. To obtain the eigenfunction the array \( slfun \) can be useful.

**Note:** Numerical classification of the endpoints \( a \) and \( b \), for the array \([nca, ncb]\), is as following:

\( nca, ncb = \)

1 regular (R),
2 weakly regular (WR),
3 limit cycle non-oscilatory (LCNO),
4 limit cycle oscillatory (LCO),
5 limit point regular at finite point (LPR),
6 limit point (LP),
7 limit point at infinity (LPI),
8 limit point bad behaved (LPB).

All the examples in Bailey et al. 2001 deal with nca, ncb = 1, 2, 3, 4, 6.
Example: An ODE initial value problem.

```matlab
// function for first derivatives
function dxdt = derivative(t, x, p){
    global(pi);
    // where
    // x[1] -> x(t)
    // x[2] -> p(t)
    // and parameters
    // p[1] = ul
    // p[2] = alfa
    // p[3] = omega
    dxdt = zeros(2,1);
    dxdt[1] = x[2];
    return dxdt;
}

// physical data
oa = sqrt(0.003); // atomic trapping frequency
ua = 2.22; // on-site self interaction
ul = pi^2; // lattice potential
TIME_UNIT = 1e-3; // time is in milliseconds
LATTICE_SPACING = 0.39e-6; // lambda/2 from experiment

printf('Enter acceleration
');
printf('Recommended values: 0.1, 0.3, 0.5, 1, 5, 7, 10\n[m/s^2] > ');
acc=strtod(getstring());
alfa = 0.5 * pi * acc * TIME_UNIT^2 / LATTICE_SPACING;

// numerical parameters for odeiv solver
options = <<>>;
options.imethod = 6; // rk2imp
options.eabs = 1e-8;
options.erel = 0;
options.stdout = rlab_initstderr();

// parameters of the calculation
p=[];
p[1] = ul;
p[2] = alfa;
p[3] = oa;

// initial conditions: BEC in the center of the trap
x0=[];
```
\[ x_0[1] = 0; \]
\[ x_0[2] = 0; \]

// Time integration
\[ t_0 = 0; \quad // \text{initial time} \]
\[ t_1 = 240; \quad // \text{final time} \]
\[ dt = 0.5; \quad // \text{time step} \]
\[ y = \text{odeiv( derivative, p, [t0:t1:dt], x0, options);} \]
\[ \quad \text{plot( y);} \]

The execution of the script produces the following run-time messages on the output device here "/dev/pts/2",

```
RLaB+4Linux: STDERR terminal on /dev/pts/2 already initialized!
RLaB: using gsl ODE integrator for initial value problem
RLaB: step method is 'rk2imp'.
RLaB: control method is 'standard'.
t = 0  x [1] = 0  x [2] = 0
 t = 0.5 x [1] = 0.4027 x [2] = 1.31731
 t = 1  x [1] = 0.963787 x [2] = 0.994439
 t = 1.5 x [1] = 1.51769 x [2] = 1.15053
.............
t = 238.5 x [1] = 9.03449 x [2] = 0.991843
t = 239 x [1] = 9.52706 x [2] = 0.978998
t = 239.5 x [1] = 10.0128 x [2] = 0.9641
 t = 240  x [1] = 10.4915  x [2] = 0.950382
RLaB: gsl ODE integrator 'rk2imp' reports 'success'.
RLaB: ODE integration lasted 821.13 sec.
```
Example: Consider an ODE boundary value problem,
\[ y''(x) = y(x) - \cos(\pi x), \quad \text{with } y(-1) = 1, y(1) = 0. \] (8.55)

```c
1 //
2 // test of the ode boundary value integrator acdc/twpbv on the problem:
3 // u'' - u = - cos(pi*x)
4 // with BC: u(-1) = u(1) = 0
5 //
6 haveplwin( 1 );
7 //
8 // u' = f(x,u,p)
9 //
10 f = function(x,u,p)
11 {
12    global(pi);
13    return [ u[2]; u[1] - cos(pi*x) ]
14 ;
15 ;
16 //
17 // df(x,u,p) = \partial f / \partial u
18 //
19 df = function(x,u,p)
20 {
21    global(pi);
22    return [ 0 , 1 ; 1, 0 ];
23 ;
24 ;
25 //
26 // gL = gL(uL,p)
27 //
28 gL = function(u,p)
29 {
30    return u[1]-1.0;
31 ;
32 dgL = function(u,p)
33 {
34    return [1,0];
35 ;
36 //
37 // gR = gR(uR,p)
38 //
39 gR = function(u,p)
40 {
41    return u[1];
42 ;
43 dgR = function(u,p)
44 {
45 //
46```
return [1,0];

// parameters
p=0;

\[X = [-1,1];\]

dimu = 2;
nlbc = 1;

options = <<>>;

options.tol = [1e-10,1e-10];

options.stdout = rlab_initstderr();

\[y = odebv(dimu,nlbc,f,df,p,X,gL,gR,dgL,dgR,options);\]

plot(y);

// plot the graph for manual: quick and dirty
xmgfilename("odebv.gr",0);

xmrange([-1,1,-2,2],0);

xmgxlabel(['"x", "y(x)"],0);

xmgyticks([0.5,4],0);

xmgprint( y );
Example: Consider an ODE boundary value problem,

$$\epsilon y''(x) - y(x) = -(\epsilon \pi^2 + 1) \cos(\pi x) \quad \text{with } y(-1) = y(1) = 0.$$  \hfill (8.56)

in a limit of small $\epsilon$, $1 > \epsilon > 0$. An appropriate code is

```cpp
plwins( 1 );

f = function(x,u,p,ep){
    global(pi);
    return [ u[2]; u[1]/ep -(pi^2+1/ep)*cos(pi*x) ];
}

df = function(x,u,p,ep){
    global(pi);
    return [ 0 , 1 ; 1/ep, 0 ];
}

gL = function(u,p,ep)
{
    return u[1];
}

dgL = function(u,p,ep)
{
    return [1,0];
}

gR = function(u,p,ep)
{
    return u[1];
}

dgR = function(u,p,ep)
{
```
One easily finds out that the solution develops boundary layers (apparent in the first derivative of the solution) at the endpoints.
Example: Consider the following (trivial) ODAE initial value problem:

\[
y_1'(t) = -4 \cdot \sin(2\pi t) \cdot (y_1(t) - y_2(t) - 3),
\]

where

\[
0 \equiv y_1(t) + y_2(t) - 1,
\]

with the initial conditions \(y_1(0) = 1, y_2(0) = 0\) and \(y_1'(0) = y_2'(0) = 0\). It is integrated for \(t \in [0, 1]\) by the following script:

```c
// file: eg_odaei1.r

// Solves (trivial) problem:
// d/dt y[1] = -4 * sin(2*pi*t) * (y[1]-y[2]-3),
// 0 = y[1] + y[2] - 1,
// with initial conditions:
// y0[1] = 0,
// y0[2] = 1,
// for T in [0,1].

f = function (t,y)
{
    global(pi);
    rval = zeros(2,1);
    rval[1] = -4 * sin(2*pi*t) * (y[1]-y[2]-3);
    return rval;
}

df = function (t,y)
{
    global(pi);
    yjac = zeros(2,2);
    yjac[1;1] = -4 * sin(2*pi*t); // d/dy[1] f[1]
    yjac[1;2] = 4 * sin(2*pi*t); // d/dy[2] f[1]
    return yjac;
}

gnuwins (1);

// mass matrix
m = [ 1, 0; 0, 0];
```
\[ T = [0:1:1/64]; \]
\[ y0 = [0,1]; \]
\[ yp0 = [0,0]; \]

// integration options
\[ \text{opts} = <<>>; \]
\[ \text{opts.step} = 1e-8; \]
\[ \text{opts.maxi} = 10000; \]
\[ \text{opts.tol_abs} = 1e-6*[1,1]; \]
\[ \text{opts.tol_rel} = 1e-2*[1,1]; \]

\[ y1 = <<>>; \]
\[ y2 = <<>>; \]

// integrate
\[ \text{for} \ (i \ \text{in} \ [0,1,2]) \{ \]
\[ \quad \text{opts.imethod} = i; \]
\[ \quad y1.[i] = \text{odaei}(f,df,,m,T,y0,yp0,opts); \]
\[ \quad y2.[i] = \text{odaei}(f,,m,T,y0,yp0,opts); \]
\[ \} \]

gnutitle ("DAE solvers");
gnxlabel("x");
gnylabel("y");
gnucmd("set grid xtics ytics");
gnulimits(0,1,-1.5,2.5);
gnuxtics(0.1,2);
gnuytics(0.5,4);
gnulegend(["(MEBDFI) y[1]", "(MEBDFI) y[2]", "(DDASKR) y[1]", "(DDASKR) y[2]", "(BIMD) y[1]", "(BIMD) y[2]""]);
gnuformat(["with lines", "with lines", "with points", "with points", "with points"]);
gnuplot (<<a=y1.[0][;1,2];b=y1.[0][;1,3];
c=y1.[1][;1,2];d=y1.[1][;1,3]; e=y1.[2][;1,2];f=y1.[2][;1,3]>>,
"fig/dae.pdf");
Example: Consider a Sturm-Liouville problem for Legendre equation,

$$-((1-x^2)y'(x))' + \frac{1}{4} y(x) = \lambda y(x), \text{ for } x \in (-1,1),$$

(8.59)

with the boundary condition function $u(x) = 1$ and $v(x) = \frac{1}{2} \log \left( \frac{1+x}{1-x} \right)$ for all $x \in (-1,1)$, with $A_1 = 1, A_2 = 0, B_1 = 1, B_2 = 0$.

The following code, see test/ode/eg_odesl3.r, finds a few eigenvalues and eigenfunctions and plots them.

```r
// sturm-liouville problem
// Legendre equation

NITER = 100;

p = function(x) {
  return 1-x^2;
}

q = function(x) {
  global(nu);
  return 0.25;
}

w = function(x) {
  return 1;
}

uv = function(t) {
  global(nu);
  y = zeros(1,4);
  // u(x)
  y[1] = 1;
  // u'(x)
  y[2] = 0;
  // v(x)
  y[3] = 0.5*log((1+t)/(1-t));
  // v'(x)
  y[4] = 1/(1 - t^2);
  return y;
}

// interval and relative mesh
a = -1;  // LCNO <-> 3
b = 1;  // LCNO <-> 3
xi = [-1+1/64:1-1/64:1/128];
x = [a,xi,b];

// type of boundary points
nc = [3,3];

// boundary conditions
bc = [1,0;1,0];

myeig=[];
options = <<>>;
options.stdout = rlab_initstderr();
options.tol = 1e-6;
eixrange = [ 2:20 ];

for(eix in eixrange)
{
    spinner();
    me = odesl.eign(p,q,w,,,x,bc,eix,nc,options);
    myeig = [myeig; (eix+1)^2, me.eigval];
    pltitle("Eigenstate "+text(eix)+" for a particle in the box");
    xlabel ("x");
    ylabel ("y(x)" );
    plstyle("line");
    pllegend("eigenfunction");
    plot( me.eigfun );
}
printf(" Done
");
printf("Eigenvalues :
");
myeig
### Table 8.1: Timings of *rlabplus* ODEIV solvers on Van-der-Pol differential equation for not-so-stiff choice of parameters, as measured by scripts `eg_odeiv1.r` and `eg_odeiv2.r` from test suite for ODE in `test/ode`. Score is the time in seconds it takes the script to solve the ODE problem 20 times, on a Xeon 2.8GHz processor. Absolute and relative errors were $1 \times 10^{-6}, 1 \times 10^{-6}$. The integrators *adams* and *rk8pd* are the excellent first choice. * marks the default solvers in each class.

<table>
<thead>
<tr>
<th>imethod</th>
<th>Integrator</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td><em>adams</em></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td><em>rk2</em></td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td><em>rk4</em></td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td><em>rkf45</em></td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td><em>rkck45</em></td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td><em>rk8pd</em></td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td><em>rk2imp</em></td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td><em>rk4imp</em></td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td><em>gear1</em></td>
<td>736</td>
</tr>
<tr>
<td>9</td>
<td><em>gear2</em></td>
<td>26</td>
</tr>
</tbody>
</table>

| 0*      | *vode*     | 2     |
| 1       | *bin*      | 19    |
| 2       | *bsimp*    | 63    |

### Table 8.2: Timings of *rlabplus* ODEBV solvers on the example script `eg_odebv1.r` in `test/ode/`. Score is the time in seconds it takes the solver to solve the problem 1000 times, on a Xeon 2.8GHz processor. Tolerances for each component were $1 \times 10^{-9}, 1 \times 10^{-9}$. The integrators *acdc* and *mirk563* are the excellent first choice for non-stiff problems. For stiff ones *acdc* is probably the best if there is no initial guess for the solution (performance of *mirkdc* largely depends on the quality of initial guess). * marks the default solver.

<table>
<thead>
<tr>
<th>imethod</th>
<th>Integrator</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td><em>acdc</em></td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td><em>mirk221</em></td>
<td>&gt; 100</td>
</tr>
<tr>
<td>2</td>
<td><em>mirk343</em></td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td><em>mirk563</em></td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td><em>colsys</em></td>
<td>18</td>
</tr>
</tbody>
</table>
Figure 8.2: A solution of Van-der-Pol equation for not-so-stiff choice of parameters, as calculated by ODEIV solvers. The timings of the solvers are shown in Table 8.1. The figure was produced using quick-and-dirty interface to xmgrace from where it was printed as an eps file.

Figure 8.3: A plot of actual solution of ODEBV in example script. The timings of the solvers are shown in Table 8.2. The figure was produced using quick-and-dirty interface to xmgrace from where it was printed as an eps file.
Figure 8.4: A plot of actual solution of ODAEI in example script obtained using Gnuplot interface (see project libgnuplot.so).
8.15 Special functions

This section describes the functions from the GSL special functions library, which are available in rlab. User may not be aware of that, but rlab comes with two special functions libraries: the GSL and cephes. For to me yet unknown reason, cephes functions are slower than the GSL’s, so I am investigating this further.

N.b.: this compendium of rlab functions is not intended for a person to learn about special functions. For more information check the GSL manual, or Mathematica® web pages.

Airy Functions, their Derivatives and the respective zeros

8.15.1 AiryAi

Format: \( y = \text{AiryAi}(x) \)

Arguments: MDR *x,

Result: MDR *y.

Abstract: The function calculates the Airy Ai function,

\[
\text{AiryAi}(x) = \frac{1}{\pi} \int_{0}^{\infty} \cos(\frac{1}{3}t^3 + xt) dt.
\]  

(8.60)

8.15.2 AiryAiPrime

Format: \( y = \text{AiryAiPrime}(x) \)

Arguments: MDR *x,

Result: MDR *y.

Abstract: The function calculates the first derivative of Airy Ai function.

8.15.3 AiryZeroAi

Format: \( y = \text{AiryZeroAi}(i) \)

Arguments: integer i,

Result: MDR *y.

Abstract: The function calculates the \( i \)-th zero of Airy Ai function.

8.15.4 AiryZeroAiPrime

Format: \( y = \text{AiryZeroAiPrime}(i) \)

Arguments: integer i,

Result: MDR *y.

Abstract: The function calculates the \( i \)-th zero of the first derivative of Airy Ai function.
8.15.5 AiryBi

Format: \( y = \text{AiryBi}(x) \)
Arguments: \( MDR \times x \),
Result: \( MDR \times y \).
Abstract: The function calculates the Airy Bi function,

\[
\text{AiryBi}(x) = \frac{1}{\pi} \int_{0}^{\infty} \left( e^{-\left(\frac{1}{3}\right)t^3} + \sin\left(\frac{1}{3}\right)t^3 + xt\right) dt.
\] (8.61)

8.15.6 AiryBiPrime

Format: \( y = \text{AiryBiPrime}(x) \)
Arguments: \( MDR \times x \),
Result: \( MDR \times y \).
Abstract: The function calculates the first derivative of Airy Bi function.

8.15.7 AiryZeroBi

Format: \( y = \text{AiryZeroBi}(i) \)
Arguments: \( \text{integer} \ i \),
Result: \( MDR \times y \).
Abstract: The function calculates the \( i \)-th zero of Airy Bi function.

8.15.8 AiryZeroBiPrime

Format: \( y = \text{AiryZeroBiPrime}(i) \)
Arguments: \( \text{integer} \ i \),
Result: \( MDR \times y \).
Abstract: The function calculates the \( i \)-th zero of the first derivative of Airy Bi function.

Bessel Functions

8.15.9 BesselI

Format: \( y = \text{BesselI}(x, \mu) \)
Arguments: \( \text{double} \ \mu, \ MDR \times x \),
Result: \( MDR \times y \).
Abstract: The function calculates the regular modified cylindrical Bessel function of the order \( \mu \).
8.15.10 BesselJ

Format: \( y = BesselJ(x, \mu) \)

Arguments: double \( \mu \), MDR \( x \)

Result: MDR \( y \).

Abstract: The function calculates the regular cylindrical Bessel function of the order \( \mu \).

8.15.11 BesselZeroJ

Format: \( y = BesselZeroJ(i, \mu) \)

Arguments: double \( \mu \), integer \( i \)

Result: MDR \( y \).

Abstract: The function calculates the \( i \)-th zero of the regular cylindrical Bessel function of the order \( \mu \).

8.15.12 BesselK

Format: \( y = BesselK(x, \mu) \)

Arguments: double \( \mu \), MDR \( x \)

Result: MDR \( y \).

Abstract: The function calculates the irregular modified cylindrical Bessel function of the order \( \mu \).

8.15.13 BesselY

Format: \( y = BesselY(x, \mu) \)

Arguments: double \( \mu \), MDR \( x \)

Result: MDR \( y \).

Abstract: The function calculates the irregular cylindrical Bessel function of the order \( \mu \). If \( \mu \) is integer then different formulas are used than if \( \mu \) is real.

8.15.14 Besseli

Format: \( y = Besseli(x, l) \)

Arguments: integer \( l \), MDR \( x \)

Result: MDR \( y \).

Abstract: The function calculates the regular modified spherical Bessel function of the integer order \( l = 0, 1, 2, \ldots \), where

\[
Besseli(l, x) = \sqrt{\frac{\pi}{2x}} BesselI(l + \frac{1}{2}, x). \tag{8.62}
\]
8.15.15  Besselj

Format: $y = Besselj(x, m)$
Arguments: integer $m$, MDR *x
Result: MDR *y.
Abstract: The function calculates the regular spherical Bessel function of the integer order $m = 0, 1, 2, \ldots$.

8.15.16  Besselk

Format: $y = Besselk(x, l)$,
Arguments: integer $l$, MDR *x
Result: MDR *y.
Abstract: The function calculates the irregular modified spherical Bessel function of the integer order $l = 0, 1, 2, \ldots$, where

\[ Besselk(l, x) = \sqrt{\frac{\pi}{2x}} BesselK(l + \frac{1}{2}, x). \]  \hspace{1cm} (8.63)

8.15.17  Bessely

Format: $y = Bessely(x, m)$,
Arguments: integer $m$, MDR *x
Result: MDR *y.
Abstract: The function calculates the irregular spherical Bessel function of the integer order $m = 0, 1, 2, \ldots$.

Clausen function

8.15.18  Clausen

Format: $y = Clausen(x)$,
Arguments: MDR *x
Result: MDR *y.
Abstract: The function calculates the Clausen function,

\[ Clausen(x) = -\int_0^x dt \log(2 \sin(\frac{t}{2})). \]  \hspace{1cm} (8.64)

Coulomb Functions

8.15.19  Hydrogen

Format: $y = Hydrogen(n, l, Z, x)$,
Arguments: integer $n$, integer $l$, double $Z$, MDR *x,
Result: MDR *y.
8.15. SPECIAL FUNCTIONS

Abstract: The function calculates normalized wave function of the hydrogen bound state. The normalization is chosen so that the wave function is \( \psi(n,l,r) = R_n(r) Y_{lm} \).

Dawson Function

8.15.20 Dawson

Format: \( y = Dawson(x) \),
Arguments: \( MDR *x \),
Result: \( MDR *y \).

Abstract: The function calculates the Dawson function,

\[
Dawson(x) = \exp(-x^2) \int_0^x dt \exp(t^2). \tag{8.65}
\]

Debye Function

8.15.21 Debye

Format: \( y = Debye(x,n) \),
Arguments: integer \( x \), \( MDR *x \),
Result: \( MDR *y \).

Abstract: The function calculates the Debye function,

\[
Debye(n,x) = \frac{n}{x^n} \int_0^x dt \frac{t^n}{\exp(t) - 1}, \tag{8.66}
\]

where \( n = 1 \ldots 4 \).

Elliptic Integrals

8.15.22 EllipticD

Format: \( y = EllipticD(k,phi,n) \),
Arguments: double \( n \), \( MDR *phi \), double \( k \),
Result: \( MDR *y \).

Abstract: The function calculates the Elliptic integral \( D \), where

\[
EllipticD(k,phi,n) = RD(1 - \sin^2 phi, 1 - k^2 \sin^2 phi), \tag{8.67}
\]

and \( RD \) is a Carlson symmetric form of elliptical integrals, see below.
8.15.23 EllipticE

Format: \( y = \text{EllipticE}(k) \) or \( y = \text{EllipticE}(k, \phi) \),

Arguments: \( MDR *k, \) double \( \phi \),

Result: \( MDR *y \).

Abstract: The function calculates the Elliptic integral \( E(k) \) if single argument is given or \( E(\phi, k) \) if two arguments are given, where

\[
\text{EllipticE}(k, \phi) = \int_0^{\phi} dt \sqrt{1 - k^2 \sin^2(t)}, \quad (8.68)
\]

and

\[
\text{EllipticE}(k) = \text{EllipticE}\left(\frac{\pi}{2}, k\right). \quad (8.69)
\]

8.15.24 EllipticF

Format: \( y = \text{EllipticF}(k, \phi) \),

Arguments: \( MDR *k, \) double \( \phi \),

Result: \( MDR *y \).

Abstract: The function calculates the function \( F(\phi, k) \), where

\[
\text{EllipticF}(k, \phi) = \int_0^{\phi} dt \frac{1}{\sqrt{1 - k^2 \sin^2(t)}}, \quad (8.70)
\]

8.15.25 EllipticJacobi

Format: \( y = \text{EllipticJacobi}(u, m) \),

Arguments: double \( u \), double \( m \),

Result: \( MDR *y \).

Abstract: The result of this function is a row-matrix with calculated Jacobian Elliptic functions \( y = [\text{sn}(u|m), \text{cn}(u|m), \text{dn}(u|m)]. \)

8.15.26 EllipticK

Format: \( y = \text{EllipticK}(k) \),

Arguments: double \( k \),

Result: \( MDR *y \).

Abstract: The function calculates the Elliptic integral \( K(k) \), where

\[
\text{EllipticK}(k) = \text{EllipticF}\left(\frac{\pi}{2}, k\right). \quad (8.71)
\]
8.15.27 EllipticP

Format: $y = \text{EllipticP}(k, \phi, n)$,
Arguments: double $n$, MDR *phi, double $k$,
Result: MDR *y.
Abstract: The function calculates the Elliptic integral $P$, where

$$\text{EllipticP}(n, \phi, k) = \int_{0}^{\phi} dt \frac{1}{(1 + n \sin^2 t) \sqrt{1 - k^2 \sin^2(t)}}.$$  \hfill (8.72)

Carlson Forms of Elliptic Integrals

8.15.28 EllipticRC, EllipticRD, EllipticRF, EllipticRJ

Format: $y = \text{EllipticRC}(x, y)$,
Format: $y = \text{EllipticRD}(x, y, z)$,
Format: $y = \text{EllipticRF}(x, y, z)$,
Format: $y = \text{EllipticRJ}(x, y, z, p)$,
Arguments: double $x$, double $y$, double $z$, double $p$,
Result: MDR *y.
Abstract: These are, so called, Carlson symmetric forms of elliptic integrals $RC$, $RD$, $RF$ and $RJ$, where

$$RC(x, y) = \frac{1}{2} \int_{0}^{\infty} dt (t + x)^{-1/2} (t + y)^{-1},$$  \hfill (8.73)
$$RD(x, y, z) = \frac{3}{2} \int_{0}^{\infty} dt (t + x)^{-1/2} (t + y)^{-1/2} (t + z)^{-3/2},$$  \hfill (8.74)
$$RF(x, y, z) = \frac{1}{2} \int_{0}^{\infty} dt (t + x)^{-1/2} (t + y)^{-1/2} (t + z)^{-1/2},$$  \hfill (8.75)
$$RJ(x, y, z, p) = \frac{3}{2} \int_{0}^{\infty} dt (t + x)^{-1/2} (t + y)^{-1/2} (t + z)^{-1/2} (t + p)^{-1}.$$  \hfill (8.76)

Error Functions

8.15.29 Erf

Format: $y = \text{Erf}(x)$,
Arguments: MDR *x,
Result: MDR *y.
Abstract: The result of this function is a the error function

$$\text{Erf}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} dt \exp(-t^2).$$  \hfill (8.77)
8.15.30 Erfc

Format: $y = \text{Erfc}(x)$,

Arguments: MDR *x,

Result: MDR *y.

Abstract: The result of this function is the complement of the error function

$$\text{Erfc}(x) = 1 - \text{Erf}(x).$$  \hfill (8.78)

8.15.31 ErfQ

Format: $y = \text{ErfQ}(x)$,

Arguments: MDR *x,

Result: MDR *y.

Abstract: The result of this function is an upper tail of the Gaussian probability function,

$$\text{ErfQ}(x) = \frac{1}{2\pi} \int_x^\infty dt \exp(-\frac{t^2}{2}).$$  \hfill (8.79)

8.15.32 ErfZ

Format: $y = \text{ErfZ}(x)$,

Arguments: MDR *x,

Result: MDR *y.

Abstract: The result of this function is the Gaussian probability of MDR *x,

$$\text{ErfZ}(x) = \frac{1}{2\pi} \exp(-\frac{x^2}{2}).$$  \hfill (8.80)

Exponential Integrals

8.15.33 ExpIntegralE

Format: $y = \text{ExpIntegralE}(x,n)$,

Arguments: integer n=0,1, MDR *x,

Result: MDR *y.

Abstract: This function calculates,

$$\text{ExpIntegralE}(x,n) = \begin{cases} n = 1 & E_1(x) = \Re \int_x^\infty dt \exp(-t) \frac{\exp(-x t)}{t^2} \\ n = 2 & E_1(x) = \Re \int_1^\infty dt \frac{\exp(-t)}{t^2} \end{cases}.$$  \hfill (8.81)
8.15. **SPECIAL FUNCTIONS**

8.15.34 **ExpIntegralEi**

Format: \( y = \text{ExpIntegralEi}(x) \),
Arguments: \( MDR \times x \),
Result: \( MDR \times y \).

Abstract: This function calculates,

\[
\text{ExpIntegralEi}(x) = \mathcal{P} \int_{-x}^{\infty} \frac{\exp(-t)}{t}.
\]  \hspace{1cm} (8.82)

Here, \( \mathcal{P} \) is the Cauchy principal value of the integral.

8.15.35 **SinhIntegral**

Format: \( y = \text{SinhIntegral}(x) \),
Arguments: \( MDR \times x \),
Result: \( MDR \times y \).

Abstract: This function calculates,

\[
\text{SinhIntegral}(x) = \int_{0}^{x} \frac{\sinh t}{t}.
\]  \hspace{1cm} (8.83)

8.15.36 **CoshIntegral**

Format: \( y = \text{CoshIntegral}(x) \),
Arguments: \( MDR \times x \),
Result: \( MDR \times y \).

Abstract: This function calculates,

\[
\text{CoshIntegral}(x) = \Re \left( \gamma_E + \log x + \int_{0}^{x} \frac{\cosh t - 1}{t} \right).
\]  \hspace{1cm} (8.84)

Here, \( \gamma_E \) is the Euler’s constant.

8.15.37 **SinIntegral**

Format: \( y = \text{SinIntegral}(x) \),
Arguments: \( MDR \times x \),
Result: \( MDR \times y \).

Abstract: This function calculates,

\[
\text{SinIntegral}(x) = \int_{0}^{x} \frac{\sin t}{t}.
\]  \hspace{1cm} (8.85)
8.15.38  CosIntegral

Format: \( y = \text{CosIntegral}(x) \),

Arguments: \( MDR \times x \),

Result: \( MDR \times y \).

Abstract: This function calculates,

\[
\text{CosIntegral}(x) = - \int_x^\infty dt \frac{\cos t}{t}.
\]  

(8.86)

8.15.39  AtanIntegral

Format: \( y = \text{AtanIntegral}(x) \),

Arguments: \( MDR \times x \),

Result: \( MDR \times y \).

Abstract: This function calculates,

\[
\text{AtanIntegral}(x) = \int_0^x dt \frac{\arctan t}{t}.
\]  

(8.87)

Fermi-Dirac Function

8.15.40  FermiDiracIntegral

Format: \( y = \text{FermiDiracIntegral}(x, m) \),

Arguments: \( \text{double} \ m = -1, 0, 1, 2, -1/2, 1/2, 3/2, MDR \times x \),

Result: \( MDR \times y \).

Abstract: This function calculates complete Fermi-Dirac integral for given \( m \),

\[
\text{FermiDiracIntegral}(x, m) = \frac{1}{\Gamma(m + 1)} \int_0^\infty dt \frac{t^m}{\exp(t - x) + 1}.
\]  

(8.88)

Gamma Function

8.15.41  Beta

Format: \( y = \text{Beta}(a, b) \),

Arguments: \( \text{double} \ a > 0, \text{double} \ b > 0 \),

Result: \( MDR \times y \).

Abstract: This function calculates Beta function,

\[
\text{Beta}(a, b) = \frac{\Gamma(a) \Gamma(b)}{\Gamma(a + b)}.
\]  

(8.89)
8.15.42 Beta

Format: \( y = \text{BetaRegularized}(a, b, x) \)

Arguments: double \( a > 0 \), double \( b > 0 \), double \( x \in [0, 1] \),

Result: MDR \( *y \).

Abstract: This function calculates regularized Beta function,

\[
\text{BetaRegularized}(a, b, x) = \frac{B_x(a, b)}{\Gamma(a + b)}.
\] (8.90)

8.15.43 Gamma

Format: \( y = \text{Gamma}(a) \),

Arguments: double \( a > 0 \),

Result: MDR \( *y \).

Abstract: This function calculates Gamma function,

\[
\text{Gamma}(a) = \int_0^a dt \, t^{a-1} \exp(-t).
\] (8.91)

8.15.44 GammaRegularized

Format: \( y = \text{GammaRegularized}(a, x) \),

Arguments: double \( a > 0 \), double \( x > 0 \),

Result: MDR \( *y \).

Abstract: This function calculates regularized Gamma function,

\[
\text{Gamma}(a, x) = \int_x^\infty dt \, t^{a-1} \exp(-t).
\] (8.92)

The function \( (8.92) \) is also known as normalized incomplete Gamma function.

8.15.45 GammaRegularizedC

Format: \( y = \text{GammaRegularizedC}(a, x) \),

Arguments: double \( a > 0 \), double \( x > 0 \),

Result: MDR \( *y \).

Abstract: This function calculates complementary normalized Gamma (GSL name for it) function

\[
\text{GammaRegularizedC}(a, x) = \frac{1}{\Gamma(a)} \int_x^\infty dt \, t^{a-1} \exp(-t).
\] (8.93)
8.15.46 LogGamma

Format: \( y = \text{LogGamma}(a) \),
Arguments: \( \text{double } a > 0 \),
Result: \( \text{MDR } * y \).
Abstract: This function calculates natural logarithm of Gamma function,

\[
\text{LogGamma}(a) = \ln \Gamma(a).
\] (8.94)

8.15.47 RecGamma

Format: \( y = \text{RecGamma}(a) \),
Arguments: \( \text{double } a > 0 \),
Result: \( \text{MDR } * y \).
Abstract: This function calculates a reciprocal of the Gamma function,

\[
\text{RecGamma}(a) = \frac{1}{\Gamma(a)},
\] (8.95)
using the real Lanczos method.

8.15.48 Pochhammer

Format: \( y = \text{Pochhammer}(a, x) \),
Arguments: \( \text{double } a > 0 \),
Result: \( \text{MDR } * y \).
Abstract: This function calculates the Pochhammer symbol,

\[
\text{Pochhammer}(a, x) = (a)_x = \frac{\Gamma(a + x)}{\Gamma(x)}.
\] (8.96)
The Pochhammer symbol is also known as the Apell symbol.

Gegenbauer Functions

8.15.49 GegenbauerC

Format: \( y = \text{GegenbauerC}(n, l, x) \),
Arguments: \( \text{double } n \geq 0, \text{double } l > -1/2, \text{double } x \),
Result: \( \text{double } y \).
Abstract: This function calculates the Gegenbauer polynomial,

\[
\text{GegenbauerC}(n, l, x) = C^{(l)}_n(x).
\] (8.97)
8.15. SPECIAL FUNCTIONS

Hypergeometric Functions

8.15.50 Hypergeometric0F1
Format: \( y = \text{Hypergeometric0F1}(a, x) \),
Arguments: double \( a \), double \( x \),
Result: double \( y \).
Abstract: This function calculates,
\[
\text{Hypergeometric0F1}(a, x) = {}_0 F_1(a, x).
\] (8.98)

8.15.51 Hypergeometric1F1
Format: \( y = \text{Hypergeometric1F1}(a, b, x) \),
Arguments: double \( a \), double \( b \), double \( x \),
Result: double \( y \).
Abstract: This function calculates,
\[
\text{Hypergeometric1F1}(a, b, x) = {}_1 F_1(a, b, x).
\] (8.99)

8.15.52 HypergeometricU
Format: \( y = \text{HypergeometricU}(a, b, x) \),
Arguments: double \( a \), double \( b \), double \( x \),
Result: double \( y \).
Abstract: This function calculates confluent hypergeometric function,
\[
\text{HypergeometricU}(a, b, x) = U(a, b, x).
\] (8.100)

8.15.53 Hypergeometric2F1
Format: \( y = \text{Hypergeometric2F1}(a, b, c, x) \),
Arguments: double \( a \), double \( b \), double \( c \), double \( |x| < 1 \),
Result: double \( y \).
Abstract: This function calculates Gauss hypergeometric function,
\[
\text{Hypergeometric2F1}(a, b, c, x) = {}_2 F_1(a, b, c, x).
\] (8.101)

8.15.54 Hypergeometric2F0
Format: \( y = \text{Hypergeometric2F0}(a, b, x) \),
Arguments: double \( a \), double \( b \), double \( x \),
Result: double \( y \).
Abstract: This function calculates
\[
\text{Hypergeometric2F0}(a, b, c, x) = {}_2 F_0(a, b, x).
\] (8.102)
Laguerre Functions

8.15.55 LaguerreL

Format: $y = \text{LaguerreL}(n, a, x)$

Arguments: integer $n$, double $a > -1$, MDR *$x$,

Result: MDR *$y$.

Abstract: This function calculates generalized Laguerre polynomials,

$$\text{LaguerreL}(n, a, x) = L_n^a(x).$$  \hfill (8.103)

Lambert Functions

Lambert’s function $W = W(x)$, in GSL notation, or ProductLog in that of Mathematica®, is defined as a solution to the equation

$$W(x) \exp(W(x)) = x.$$  \hfill (8.104)

8.15.56 ProductLog

Format: $y = \text{ProductLog}(x/, m/)$

Arguments: double $x$, integer $m = 0$ (default) -1.

Result: double $y$.

Abstract: For $m = 0$ the function calculates a solution on the principal branch of Lambert’s W function, $W_0(x)$, while for $m = -1$ it calculates a solution on secondary valued branch, $W_{-1}$.

Legendre Functions and Spherical Harmonics

8.15.57 LegendreP, LegendreQ

Format: $y = \text{LegendreP}(l, x)$, or $\text{LegendreP}(l, m, x)$, or $y = \text{LegendreQ}(l, x)$,

Arguments: integer $l \geq m$, integer $m \geq 0$, MDR *$x$,

Result: MDR *$y$.

Abstract: The function returns, depending on the number of arguments,

$$\text{LegendreP}(l, x) = P_l(x),$$  \hfill (8.105)

for two arguments, or

$$\text{LegendreP}(l, m, x) = P_l^m(x),$$  \hfill (8.106)

for three.
8.15.58 LegendreSphericalP

Format: $LegendreSphericalP(l, m, x)$

Arguments: double $l$, integer $m$, double $x$,

Result: $MDR \times y$.

Abstract: The function returns the value of the normalized associated Legendre polynomial, suitable for use in spherical harmonics.

8.15.59 LegendreConicalP

Format: $LegendreConicalP(l, m, x)$

Arguments: double $l$, double $2 \cdot m = -1, 0, 1, 2, \ldots$, double $x$,

Result: $MDR \times y$.

Abstract: The function returns the value of the conical function

$$ConicalP(z, m, x) = P_{-1/2+m} ^{-m/2+1/2}(x). \quad (8.107)$$

If $2 \cdot m$ is even, this is a regular cylindrical conical function and if $2 \cdot m$ is odd, this is a regular spherical conical function.

8.15.60 LegendreH3d

Format: $y = LegendreH3d(m, l, n)$

Arguments: integer $m$, double $l$, double $n$,

Result: double $y$.

Abstract: The function returns the special value of Legendre function, which is the eigenvalue of Laplacian on 3-dimensional hyperbolic space $H3d$,

$$LegendreH3d(m, l, n) = L_{m}^{H3d} (l, n). \quad (8.108)$$

Psi Functions

8.15.61 Digamma

Format: $y = Digamma(x)$

Arguments: double $x$,

Result: double $y$.

Abstract: The function returns digamma function,

$$Digamma(x) = \frac{\Gamma'(x)}{\Gamma(x)}. \quad (8.109)$$
8.15.62 PolyGamma

Format: \( y = \text{PolyGamma}(m, x) \)

Arguments: integer \( m \), double \( x \),

Result: double \( y \).

Abstract: The function returns,

\[
\text{PolyGamma}(m, x) = \frac{d^m}{dx^m} \varphi(x),
\]

where

\[
\varphi(x) = \frac{\Gamma'(x)}{\Gamma(x)}
\]

Synchrotron Functions

8.15.63 SynchrotronF

Format: \( y = \text{SynchrotronF}(x, n) \)

Arguments: integer \( n = 1, 2 \), double \( x \),

Result: double \( y \).

Abstract: Calculates first and second synchrotron function.

Transport Functions

8.15.64 TransportF

Format: \( y = \text{TransportF}(x, n) \)

Arguments: integer \( n = 2, 3, 4, 5 \), double \( x \),

Result: double \( y \).

Abstract: Calculates the transport function,

\[
\text{TransportF}(n, x) = \int_0^x dt \frac{t^n}{(\exp t - 1)^2}, \text{ for } n = 2, 3, 4, 5.
\]

Zeta Functions

8.15.65 Zeta

Format: \( y = \text{Zeta}(s) \), or \( y = \text{Zeta}(s, x) \)

Arguments: double \( s \), double \( x \),

Result: double \( y \).
Abstract: This function, depending on the number of arguments, calculates

\[ Z_t(a) = \sum_{j=0}^{\infty} \frac{1}{j^s}, \]  

for single argument, or

\[ Z_t(a, x) = \sum_{j=0}^{\infty} \frac{1}{(j + x)^s}. \]  

The function (8.113) is known as Riemann Zeta Function, while the function (8.114) is known as Hurwitz Zeta Function.
Chapter 9
Special Topics Solvers

9.1 Time Series Analysis (TISEAN)

TISEAN is a collection of software written in C and FORTRAN which provides a basic set of functions for tackling the problem of time series analysis. The TISEAN library, unlike gsl, does not require installation of additional libraries. TISEAN is copyrighted by Rainer Hegger, Holger Kantz and Thomas Schreiber from Institut für Physikalische und Theoretische Chemie Universität Frankfurt (Main), and Max-Planck-Institut für Physik komplexer Systeme Dresden.

9.1.1 rmap

Format: \( y = \text{rmap}(f, /p/, x0, n0, nt) \),

Arguments:

1. \( f = \text{function}(x, /p/) \), a vector function, \( p \), its parameter entity, where \( \dim f = \dim x0 \);
2. \( x0 \), a real vector, initial point of iteration;
3. \( n0 \), an integer, a number of iterations being calculated;
4. \( nt \), an integer, number of iterations being discarded (thermalization).

Result: \( y \), a real matrix of size \( n0 \)-by-\( \dim x0 \).

Abstract: \( \text{rmap} \) applies the recursion defined by the function \( f(x, /p/) \),

\[
    x_{j+1} = f(x_j, p),
\]

(9.1)
a number of iterations equal to \( n0 + nt \), where the first \( nt \) iterations are discarded.

9.1.2 xcorr

Format: \( y = \text{xcorr}(x, y, \text{delay} /, \text{cnorm} /) \),

Arguments:

1. \( x \), real or complex vector, time series sampled at equidistant time points;
2 \( y \), real or complex vector, time series sampled at equidistant time points;

3 \( \text{delay} \), real vector containing delays of interest.

4 \( \text{cnorm} \), string, description of normalization

**Result:** \( y \), real vector of size \( \text{delay} \), containing the cross-correlation of the two time series for particular delay \( d \),

\[
xcorr(x, y, d) = \frac{1}{C} \sum_{i=1}^{\min(N_x, N_y)} x_i \cdot y^*_i + d
\]

If the series are not of equal length this is equivalent to having the shorter one padded with zeros to the length of the longer one. The constant \( C \) in Eq. (9.2) depends on the parameter \( \text{cnorm} \), where for “none” \( C = 1 \), for “biased” \( C = \min(N_x, N_y) \), for “unbiased” \( C = \min(N_x, N_y) - d \) and for “coef” \( C \) is a product of norms of respective vectors, so that when using \( xcorr \) to calculate the auto correlation function for \( d = 0 \) one obtains unity. Default value for \( \text{cnorm} \) in “none.”

9.1.3 \( \text{runavg} \)

**Format:** \( y = \text{runavg}(x, T) \),

**Arguments:**

1 \( x \), a real vector, time series sampled at equidistant time points;

2 \( T \), a real vector, the length of the averaging period.

**Result:** \( y \), a real vector of size of \( x \), containing the running average of the time series for the length of the period equal to \( T \).

**Note:** The first \( T - 1 \) entries in \( y \) are zeros.

9.1.4 \( \text{runvar} \)

**Format:** \( y = \text{runvar}(x, T) \),

**Arguments:**

1 \( x \), a real vector, time series sampled at equidistant time points;

2 \( T \), a real vector, the length of the averaging period.

**Result:** \( y \), a real vector of size of \( x \), containing the running variance of the time series for the length of the period equal to \( T \).

**Note:** The first \( T - 1 \) entries in \( y \) are zeros.
9.1.5 ami

Format: \( y = \text{ami}(x, \text{delay}, \text{part}) \),

Arguments:
1. \( x \), a real vector, time series sampled at equidistant time points;
2. \( \text{delay} \), a real vector containing delays of interest;
   • \( \text{part} \), an integer, number of partitions when counting the pair probability distribution (default, \( \text{part} = 16 \)).

Result: \( y \), a real vector of size \( \text{delay} \) containing the mutual information for each \( \text{delay} \).

9.1.6 falsenn

Format: \( y = \text{falsenn}(x, \text{emb}, \text{delay}, \text{stderr}, \text{escfac}, \text{theilerwin}) \),

Arguments:
1. \( x \), a real vector, time series sampled at equidistant time points;
2. \( \text{emb} \), a real vector, embedding dimensions of interest;
3. \( \text{delay} \), an integer, a delay used in constructing the vector of dimension \( \text{emb} \) of the time series;
4. \( \text{stderr} \), a konsole where the messages generated by the code are posted;
5. \( \text{escfac} \), a scalar, escape factor;
6. \( \text{theilerwin} \), the Theiler window.

Result: \( y \), a real vector, false-nearest-neighbor statistics \cite{Aberbanel1996} of a time series for given embedding dimension \( \text{emb} \) and time delay \( \text{delay} \).

9.1.7 poincare

Format: \( y = \text{poincare}(x, \text{emb}, \text{delay}, \text{comp}, \text{crossdir}, \text{crosspos}) \),

Arguments:
1. \( x \), a real vector, time series sampled at equidistant time points;
2. \( \text{emb} \), a real vector, embedding dimensions of interest;
3. \( \text{delay} \), an integer, a delay used in constructing the vector of dimension \( \text{emb} \) of the time series;
4. \( \text{comp} \), an integer, an index of a coordinate of a point in a reconstructed phase space;
5, 6 \( \text{crossdir} \), an integer, \( \text{crosspos} \), a scalar, a direction (1, for positive, 0 for negative) of crossing and a position of crossing of a hyperplane, \( x_{\text{comp}} = \text{crosspos} \), by the system trajectory in the reconstructed phase space.

Abstract: \textit{poincare} finds the poincare section of a \textit{embdim}-dimensional signal, constructed from time series \( x \) using \( \text{delay} \). A point is added to the output array \( y \) if the \( \text{comp} \) coordinate of the data vector (of size \( \text{embdim} \)) crosses the value \( \text{crosspos} \) in the positive \( (\text{crossdir} = 1) \) or negative \( (\text{crossdir} = 0) \) direction. For further reading check \cite{Aberbanel1996} or the TISEAN manuals.
Example: Henon map for $p$ for which the chaos appears. We find 100,000 points of the recursing after discarding first 500,000 points. Initial condition $x_0$ was chosen on random.

```plaintext
1 //
2 // rmap on henon mapping
3 //
4 henon = function( x , p )
5 {
6    // input:
7    // p[1] -> a
8    // p[2] -> b
9    // x[1] -> x(n)
10   // x[2] -> y(n)
11   // output:
12    // rx[1] <- x(n+1)
13    // rx[2] <- y(n+1)
14    rx = zeros(x.nr,1);
16    rx[2] = x[1];
17    return rx;
18 }
19
20 //
21 // parameters where chaos appears
22 //
23 // p[1] -> a
25 p = [];  
26 p[1] = 1.4;
27 p[2] = 0;
28
29 // initial condition
30 x0 = rand(2,1);
31
32 // iteration with
33 n0 = 100000;
34 nt = 500000;
35 y = rmap(henon, p, x0, n0, nt);
```
9.1. TIME SERIES ANALYSIS (TISEAN)

Basic Time Series Analysis - S&P500 Market Index (gspc) for period from 4-Dec-87 to 24-Mar-06

Figure 9.1: Results of functions from chaos toolkit applied to the data set consisting of a S&P500 market indicator in period April 1987 to August 2004. The stock market data was obtained by querying the server finance.yahoo.com.
9.2 The Gnu Linear Programming Kit (GLPK)

The Gnu Linear Programming Kit (GLPK) is released as a part of the GNU project by Makhorin (2006). The rlabplus is built for GLPK version 4.52 or higher.

The GLPK provides user with solvers to solve two type of problems: general linear programming (LP) problem, and mixed integer problem (MIP). For LP problem there are two methods: simplex method, and interior-point method, while there is only one method available for MIP.

In rlab3 the GLPK can be accessed through two interfaces, low-level one that comprises functions from _glpk_* below, or high-level one that involves classdef interface.

**rlabplus lp list**

In rlabplus, lp is list which contains all data needed to pose an LP or MI problem. Its entries are:

- **objective**, real vector, contains the objective of the linear programming problem, i.e., the linear functional that needs to be optimized;
- **c0**, real scalar, offset of the objective functional;
- **opt_direction**, string “min” or “max”, determines whether optimization problem is to find minimum or maximum of the objective functional;
- **problem**, string scalar, class of problem “lp” for general linear programming, or “mip” for mixed-integer problem;
- **constraints**, real matrix of size $m$-by-$n$, describes the relationship between the structural variables (columns) and the auxiliary variables (rows) in canonical formulation of the linear programming problem;
- **bounds_row**, real matrix of size $m$-by-2, first column (can be $-\infty$) gives lower while the second column (can be $+\infty$) gives upper bound on particular row (auxiliary) variable;
- **bounds_col**, real matrix of size $n$-by-2, first column (can be $-\infty$) gives lower while the second column (can be $+\infty$) gives upper bound on particular column (structural) variable;
- **problem**, string “lp” or “mip”, determines the type of problem: linear programming, or mixed-integer problem;
- **col_int, col_bin**, real vectors, relevant for MIP. It contains 1 if respective column variable is integer, or binary, in that order.

The functions operating on the lp list are available through a function list lpx.

9.2.1 _glpk_read

**Format:** $y = _glpk_read(filename[, fmt])$

**Arguments:**

1. **filename**, string, name of the file containing the specification of the linear programming problem;
2. **fmt**, string, format in which the data in the file is given if it cannot be determined from the extension of the filename:
“mps” for file in MPS format;
“lpt” for file in CPLEX LP format; and,
“mod” for file written in GNU MathProg modeling language;

Result: \( y \), list \( lp \), containing (presumably) a completely specified LP problem.

9.2.2 \_glpk\_write

Format: \( \text{status} = \_glpk\_write(\text{filename}[\text{, fmt}]) \)

1. \text{filename}, string, name of the file to which the linear programming problem will be written;

2. \text{fmt}, string, format in which the data in the file will be given if it cannot be determined from the extension of the filename:
   “mps” for file in MPS format;
   “lpt” for file in CPLEX LP format.

9.2.3 \_glpk\_solve

Format: \( y = \_glpk\_solve(l, \text{opts}) \)

1. \( l \), \( lp \) list;

2. \text{opts}, list, contains different options for the solver. Its entries can be grouped as follows:
   - \text{stdout}, string, file or terminal where solver will write messages. If given, the solver assumes maximum possible verbosity
   - \text{method}, string scalar, choice of solver for “lp” class of problem. Choices “primal”, “dual”, and “dualp” presume simplex method, while “interior” presumes interior point method. The parameter is irrelevant for mixed-integer solver.

   For simplex method, the following options are relevant:
   - \text{pricing}, string scalar, pricing technique: “std” for standard textbook, and “pse” for projected steepest edge;
   - \text{r\_test}, string scalar, ratio test technique: “std” for standard textbook, and “har” for Hariss’ two-pass ratio test;
   - \text{tol\_bnd}, real scalar, tolerance used to check if the basic solution is primal feasible;
   - \text{tol\_dj}, real scalar, tolerance used to check if the basic solution is dual feasible;
   - \text{tol\_piv}, real scalar, tolerance used to choose eligible pivot elements from the simplex table;
   - \text{obj\_min}, real scalar, lower limit of the objective function. If the objective function reaches this limit and goes beyond it, the solver terminates the search;
   - \text{obj\_max}, real scalar, upper limit of the objective function. See above;
   - \text{maxi}, integer scalar, iteration limit of the simplex solver;
   - \text{presolve}, integer scalar, 0 to disable LP presolver, 1 to enable.

   For interior point method, the following option is relevant:
   - \text{ordering}, string scalar, determines ordering algorithm prior to Cholesky factorization: “none”, “am01” for approximate minimum degree, “qmd” for quotient minimum degree, and “symamd” for another variation of approximate minimum degree algorithm.
For mixed-integer problem, the following options are relevant:

- **branch**, string scalar, determines the branching technique: “ffv” for first fractional variable, “lfv” for last fractional variable, “mfv” for most fractional variable, “dth” for heuristic by Driebeck and Tomlin, and “pch” for hybrid pseudo-cost heuristic;

- **backtrack**, string scalar, determines the backtracking technique: “dfs” for depth first search, “bfs” for breadth first search, “blb” for best local bound, “bph” for best projection heuristic;

- **preprocess**, string scalar, determines preprocessing: “none”, “root” for doing it only at the root level, and “all” for doing it at all levels;

- **feas_pump**, real scalar, determines whether to use (1) feasibility pump heuristic option or not (0);

- **proximity_search**, integer scalar, determines the duration in milliseconds of the duration of the proximity search heuristic. If 0 the search is disabled.

- **gmi_cuts**, integer scalar, determines if Gomory’s mixed integer cut option is enabled (1), or not (0);

- **mir_cuts**, integer scalar, determines if mixed integer rounding cut option is enabled (1), or not (0);

- **cov_cuts**, integer scalar, determines if mixed cover cut option is enabled (1), or not (0);

- **clq_cuts**, integer scalar, determines if clique cut option is enabled (1), or not (0);

- **tol_int**, real scalar, tolerance used to check if the current solution is integer feasible;

- **tol_obj**, real scalar, relative tolerance used to check if the value of the objective function is not better than the best known integer feasible solution;

- **mip_gap**, real scalar, the relative mip gap tolerance;

- **ch_size**, integer in range 0 through 255, number of extra bytes allocated for each node of the branch-and-bound tree to store application non-specific data;

- **presolve**, integer scalar, allows solver to use MIP presolver (1), or not (0);

- **binarize**, integer scalar, tells solver to replace general integer variables with binary ones.

**Result:** $y$, list which entries depend on the class of problem and method.

For simplex method:

$y = \langle \text{fail}; \text{status}; \text{primal\_status}; \text{row\_coef}; \text{col\_coef}; \text{dual\_status}; \text{dual\_row\_coef}; \text{dual\_col\_coef}; \text{objective}; \text{row\_unbound}; \text{col\_unbound} \rangle$

For interior point method:

$y = \langle \text{fail}; \text{status}; \text{row\_coef}; \text{col\_coef}; \text{dual\_row\_coef}; \text{dual\_col\_coef}; \text{objective} \rangle$ for interior point method;

and

$y = \langle \text{fail}; \text{status}; \text{row\_coef}; \text{col\_coef}; \text{objective} \rangle$ for mixed integer problem.

For all solvers:

- **fail**, integer scalar, 0 for success, otherwise contains a number describing type of error;

- **status, primal\_status, dual\_status**, string scalar, describes status of basic primal or dual simplex solution: “optimal”, “feasible”, “infeasible”, “unbounded” or “undef”.

- **row\_coef, col\_coef**, real vectors, solutions for row (auxiliary) and column (structural) variables;

- **objective**, real scalar, value of objective function.

Simplex and interior point solvers have:

- **dual\_row\_coef, dual\_col\_coef**, real vectors, solutions for dual row (auxiliary) and column (structural) variables.
Example: The sample problem from the reference manual (Makhorin, 2006), pp.9,
maximize
\[ Z = 10 x_1 + 6 x_2 + 4 x_3, \] (9.3)
subject to
\[ \begin{align*}
& x_1 + x_2 + x_3 \leq 100, \\
& 10 x_1 + 4 x_2 + 5 x_3 \leq 600, \\
& 2 x_1 + 2 x_2 + 6 x_3 \leq 300,
\end{align*} \] (9.4)
where all variables are non-negative,
\[ x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0. \] (9.5)
Solution: The problem is first transformed to a standard form,
maximize
\[ Z = c^T \cdot x \] (9.6)
subject to constraint
\[ r = Ax \] (9.7)
where the bounds for the entries of an auxiliary vector \( r \) are,
\[ auxB_{i,1} \leq r_i \leq auxB_{i,2} \] (9.8)
and the bounds for an entry of a structural vector \( x \) are,
\[ strB_{j,1} \leq x_j \leq strB_{j,2}. \] (9.9)
In our case, these are
\[ A = \begin{bmatrix} 1 & 1 & 1 \\ 10 & 4 & 5 \\ 2 & 2 & 6 \end{bmatrix}, \quad \text{and} \quad c = \begin{bmatrix} 10 \\ 6 \\ 4 \end{bmatrix}, \] (9.10)
and the bounds are
\[ auxB = \begin{bmatrix} -\inf() & 100 \\ -\inf() & 600 \\ -\inf() & 300 \end{bmatrix}, \] (9.11)
and
\[ strB = \begin{bmatrix} 0 & \inf() \\ 0 & \inf() \\ 0 & \inf() \end{bmatrix}, \] (9.12)
with
\[ x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \text{and} \quad r = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}. \] (9.13)

lpopts = <<>>;
lpopts.method = ‘interior’; // interior point method
lpopts.stdout = stderr();

y0 = <<>>;
y0.objective = [10,6,4]; // cost function
y0.constraints = [1,1,1; 10,4,5; 2,2,6]; // constraint matrix
y0.bounds_row = [-inf(),100; -inf(),600; -inf(),300]; // auxiliary (row) bounds
y0.bounds_col = [0,inf(); 0,inf(); 0,inf()]; // structural (col) bounds
y0.opt_direction = "max";
y0.problem = "lp";

s = _glpk_solve(y0, lpopts);

This produces output

```
s.col_coef = 
    33.3333333 66.6666667 0
>>
```
classdef interface

In $R3D/rclass directory, a classdef facility in rlab3 is demonstrated in file lp.r3.

Class constructor for the GLPK is called lp, and it can be used to create a linear programming problem class like this,

```matlab
>> y=lp()
ans =
    bounds_col    bounds_row    constraints    ncol    nrow
     objective    offset    opt_direction    problem    solve
       write
```

Each of the entries in the list y is a method for accessing (if no arguments are given) or for setting (if argument is provided) the appropriate private entry similar to the rlabplus list lp given earlier. If lp is provided with arguments, it assumes this is the file name containing problem definition (see _glpk_read earlier).

```matlab
//
//
lpopts = <<>>;
lpopts.method = 'interior'; // interior point method
lpopts.stdout = stderr();

y0 = lp();
y0.objective([10,6,4]); // cost function
y0.constraints([1,1,1; 10,4,5; 2,2,6]); // constraint matrix
y0.bounds_row([-inf(),100; -inf(),600; -inf(),300]); // auxiliary (row) bounds
y0.bounds_col([0,inf(); 0,inf(); 0,inf()]); // structural (col) bounds
y0.opt_direction("max");
y0.problem("lp");

s = y0.solve(lpopts);
```

This produces output

```matlab
s.col_coef =
    33.3333333  66.6666667  0
>>
```
9.3 Signal Processing Toolkit

9.3.1 filter

I. Format: \( y = \text{filter}(x, opts) \)

Arguments:

1. \( x \), real or complex vector, input time series;
2. \( opts \), list, with entries:
   - \( a \), real or complex array, the auto regression (AR) or infinite impulse response (IIR) array of the filter;
   - \( b \), real or complex array, the moving average (MA) of finite impulse response (FIR) array of the filter;
   - \( gain \), real scalar, the gain of the FIR filter, important if one chooses \( a[1] = 1 \);
   - \( y0 \), real or complex array, which length \( l \) matches the order of the IIR filter \( a \) as \( l = \text{deg}(a) - 1 \), these are initial conditions for \( y \);
   - \( zi \), real or complex array, similar to \( y0 \), provides initial conditions. Exclusive with \( y0 \). If none is given, the solver uses \( zi = [0, \cdots, 0] \);
   - \( periodic \), 0 or 1, select periodic boundary conditions on \( y \). Relevant if \texttt{filter} is going to be used within FFT.

or, for GSL greater than 2.4 installed, and \( x \) real array, additional filters are available through \( opts \), list, see GSL documentation for interpretation of entries:
   - \( smedian \), positive integer, order of the standard median filter to be used on input time series;
   - \( rmedian \), positive integer, order of the recursive median filter to be used on input time series;
   - \( mad \), positive real scalar, scale of the moving absolute difference to be used in specifying outliers for the impulse filter;
   - \( iqr \), positive real scalar, scale of the inter quartile range to be used in specifying outliers for the impulse filter;
   - \( sn \), positive real scalar, S-N scale to be used in specifying outliers for the impulse filter;
   - \( qn \), positive real scalar, Q-N scale to be used in specifying outliers for the impulse filter;
   - \( endt \), string, specify end point treatment for the filters, where "val" implies padding by values, "truncate" implies truncation, or leave default zero-padding.

This form can take the output of \texttt{mkfilter} solver as filter description.

II. Format: \( y = \text{filter}(x, b, /a/, /opts/) \)

Arguments:

1. \( x \), real or complex vector, input time series;
2. \( b \), real or complex array, the moving average (MA) of finite impulse response (FIR) array of the filter. Includes \( gain \);
3. a, real or complex array, the auto regression (AR) or infinite impulse response (IIR) array of the filter. If omitted, only FIR filter is considered.

4. opts, list, with entries:
   – y0, see above;
   – zi, see above;
   – periodic, see above.

Result: $y = \ll y, zf \gg$, where $y$ is real or complex array of the length of $x$. Output of the filter; $zf$ is the array of final differences computed for output $y$ and filter specified through $b$, $a$ and gain.

9.3.2 harminv

Format: $s = \text{harminv}(y,F/,\text{opts}/)$

Arguments:

1. $y$, real or complex vector, time signal recorded at equidistant time intervals $dt$ (see opts);
2. $F$, two-column real matrix, each row contains a range of frequencies to be fitted;
3. opts, list, with entries:
   – amp_abs, amp_rel, positive scalars, threshold for absolute (relative with respect to maximum) values of amplitudes. Smaller values will be discarded. Default values 0.1, $10^{30}$;
   – err_abs, err_rel, positive scalars, threshold for absolute (relative with respect to maximum) error as calculated by the code. Smaller values will be discarded. Default values 0, 0;
   – $Q$, positive scalar, threshold for $Q$-factor. Smaller values will be discarded; Default value 10.
   – $dt$, positive scalar, time interval at which signal $y$ is recorded. Default value 1;
   – density, positive scalar, denisty of modes. Default value 1;
   – $nf$, two-column integer vector, lower and upper bound on the number of frequencies sought in the interval given by $F$. Default value $[2, 50]$;
   – solve_mode, integer 0,1,2. Default value 0.

Result: $s$, list with entries

• freq, real vector, frequencies of the modes found by the code;
• decay, real vector, decay rates of the same modes;
• amp, complex vector, amplitudes of the same modes;
• err, real vector, errors in estimates of the amplitudes, as calculated by the code.

9.3.3 stl

**Format:** \( s = stl(x, Y, br, cc) \),

**Arguments:**

1. \( x \), real vector, abscissae of points;
2. \( Y \), real vector, ordinatae of points, or a two-column matrix, ordinatae and their tolerances;
3. \( br \), integer 0 or 1, indicates whether the points are band restricted (1) or not (0, default);
4. \( cc \), integer 0 or 1, indicates whether to look for a continuous line segment (1) or not (0).

**Result:** \( s = [x_j, y_j] \), two-column real matrix, the end points of the line segments.

**Abstract:** stl approximates the data set \( \{x_i, y_i\} \), possibly with \( y \)'s tolerances given as the second column in \( Y \), with a sequence of line segments. Parameter \( cc = 0 \) (default) indicates that the line segments need not be continuous. In that case each discontinuity appears two times, with left and right limits, respectively:

\[
\begin{bmatrix}
  x_1 & y_1 \\
  \vdots & \vdots \\
  x_k & y_k \\
  x_k & y_{k+1} \\
  \vdots & \vdots
\end{bmatrix}
\]  

(9.14)

If \( cc = 1 \) a continuous sequence of line segments is allowed:

\[
\begin{bmatrix}
  \vdots \\
  x_k & y_k \\
  \vdots
\end{bmatrix}
\]  

(9.15)

**Note:** The original FORTRAN code was written by D.G. Wilson, ACM TOMS 2 (1976) 388-391, and published on [www.netlib.org](http://www.netlib.org) as TOMS/510. The source code distributed with rlab carries the reference to the original work and was modified to have **double precision** array pointers (instead of the single precision arrays of fixed size).

**Note:** The table \( s \) given by \( s = stl(..) \) is suitable for use in linterp(..) function, as it can handle the discontinuities.

---

**Time-Frequency Operations**

9.3.4 window

**Format:** \( w = window(‘name’, length, p) \),

**Arguments:**

1. \( name \), string, name of the window. Admissible values are “rectangular”, “hamming”, “hanning”, “nuttall”, “blackman”, “harris”, “bartlett”, “barthann”, “papoulis”, “gauss” (1 optional parameter), “parzen”, “hanna” (1 optional parameter) and “spline” (1 necessary and 1 optional parameter);
2 length, integer, length of the filter;

3 p, real vector, contains optional or necessary parameters for the window function. For “gauss” log $p[1]$ is the multiplicative factor to the Gaussian centered at the mid-point of the window; for “hanna” $2 \cdot p[1]$ determines the exponent in the expansion of the window function; while for “spline” $p[1]$ is the number of frequencies while $p[2]$ is the relevant exponent.

### 9.3.5 stft

**Format:** $z = \text{stft}(\text{signal, window/}, \text{range/})$,

**Arguments:**

1 signal, real or complex vector, signal of interest that is sampled at uniform rate.

2 window, real vector of odd length, window function sampled at the same rate as signal, which is going to be used to limit the time signal before its Fourier transform;

3 range, two component vector of the form $[i_{\min}, i_{\max}]$. While the short-time fourier transform will be performed over a maximum possible signal length, only the frequency components in the range $f_{i_{\min}}$ to $f_{i_{\max}}$ will be recorded.

**Result:** $z = << \text{tfr; freq.bins; time.instants} >>$, list, where $1/\Delta t \cdot \text{freq.bins} = f_j$ are the frequencies, $\Delta t \cdot \text{time.instants} = t_i$ are the times, while $\text{tfr}[j; i] = \text{tfr}(f_j, t_i)$ is the short time Fourier transform of signal. Here, $\Delta t$ is the sampling time. The short time Fourier transform is given by

$$
stft(s, w) = \mathcal{F} \left( \int dt' w(t - t') s(t') \right),$$

(9.16)

where $w = w(\tau)$ is the window function, which, among other things is symmetric about the origin and non-negative, $w(\tau) = w(-\tau) \geq 0, \forall \tau$. In other words, stft is the Fourier transform of a windowed signal.

### 9.3.6 fftshift

**Format:** $z = \text{fftshift}(\text{signal})$

1 signal, real or complex vector, signal of interest that is sampled at uniform rate.

**Result:** The function splits the signal in two halves and than changes their order.

### 9.3.7 tfrd

**Format:** $z = \text{tfrd.dist}(\text{signal, winT, winF/, p1, p2, .../})$

1 signal, real or complex vector, signal of interest that is sampled at uniform rate.

2,3 winT, winF, real vector, window for time and frequency domain.

4 p, real vector, necessary parameters for calculating the time-frequency distribution of a given signal with respect to the two windows functions.
Result: \( z = \langle \theta, \text{freq bins}; \text{time instants} \rangle \), list, where \( 1/\Delta t \cdot \text{freq bins} = f_j \) are the frequencies, \( \Delta t \cdot \text{time instants} = t_i \) are the times, while \( tfr[j;i] = tfr(f_j,t_i) \) is the time-frequency distribution of signal. Here, \( \Delta t \) is the sampling time. The following distributions are available for given \( \text{dist} \): “bj”, for Born-Jordan; “butter”, for Butterworth with a positive scalar as the parameter; “cw”, for Choi-Williams with a positive scalar as a parameter; “gr”, for generalized rectangular with two positive scalars as the parameters \( p1=rs \) and \( p2=dsym \); “mh” for Margenau-Hill, which does not require the time nor the frequency window; “page” for Page, which also does not require the time nor the frequency window; “pmh” for pseudo Margenau-Hill, which requires the time window; “ppage”, for pseudo Page, which requires the time window; “pwv”, for pseudo Wigner-Ville, which requires the time window; “ri”, for ??, which does not require the time nor the frequency window; “spwv”, for smoothed pseudo Wigner-Ville; “wv”, for Wigner-Ville, which does not require the time nor the frequency window; “zam”, for ??.

\subsection{9.3.8 tfrid}

Format: \( z = \text{tfrid} \cdot \text{dist} \cdot (\text{signal}, \text{winT}, \text{winF}, /p1,p2,.../) \)

1 \( \text{signal} \), real or complex vector, signal of interest that is sampled at uniform rate.

2,3 \( \text{winT}, \text{winF} \), real vector, window for time and frequency domain.

4 \( p \), real vector, necessary parameters for calculating the time-frequency reduced interference distribution of a given \( \text{signal} \) with respect to the time, and possibly the frequency, window function.

\( z = \langle \theta, \text{freq bins}; \text{time instants} \rangle \), list, where \( 1/\Delta t \cdot \text{freq bins} = f_j \) are the frequencies, \( \Delta t \cdot \text{time instants} = t_i \) are the times, while \( tfr[j;i] = tfr(f_j,t_i) \) is the time-frequency reduced interference distribution of signal. Here, \( \Delta t \) is the sampling time. The following distributions are available for given \( \text{dist} \): “bessel”, “binom” for binomial, “tri” for trigonal, and “hann” for Hanning.

\section{B-spline smoothing}

\subsection{9.3.9 bsplinefit}

Format: \( z = \text{bsplinefit} \cdot (y,x, S, /, \text{options}/) \),

Arguments:

1 \( y \), real vector, ordinatae of points, or list, \( y = \langle \text{val}; \text{wgt} \rangle \), of ordinatae and their weights;

2 \( x \), real vector, abscissae of points, or list, \( x = \langle \text{val}; \text{wgt} \rangle \), of abscissae and their weights;

3 \( S \):

(a) real vector, position of the knots used to construct B-splines. Has to satisfy Schönberg-Whitney criterion (there has to be data points in the set inbetween the knots). Coefficients of the spline approximation are found by least-squares method;
(b) nonnegative scalar, smoothing parameter. Finds the minimum number of knots and their position so that the convergence criterion as specified by $S$ is satisfied;

3 options, list, its entries are

- $\text{degree}$, integer, degree of smoothing polynomial. Values 1, 3 (default) and 5;
- $\text{periodic}$, scalar, 0 (default) or 1 if the approximant is known to be periodic;
- $y'_{i}, y'_{f}$, scalars, values of first derivative at end point(s);
- $\text{maxi}$, integer, maximal number of iterations to reach desired tolerance. Default value 40;
- $\text{tol}$, positive scalar, desired tolerance of the smoothing spline. Default value 0.01;
- $\text{convex}$, real vector, of size $x$, or of size $S$ (when $S$ is a vector of knots). Convexity criterion for spline fitting, its entries are 0, ±1 and imply that the second derivative at the position of the knot in $S$ or at the abscise in $x$ is irrelevant, or $\mp$.

Result: $z$, B-spline compliant list, with entries

- $\text{coef}$, vector, coefficients of spline polynomial;
- $\text{knot}$, vector, position of knots;
- $\text{degree}$, integer, degree of spline polynomial (same as input);
- $\text{residual}$, real scalar, mean-squared residual of the fit;
- $\text{status}$, integer, nature of solution as follows:
  - 2 weighted least-squares polynomial curve;
  - 1 interpolating spline curve;
  - 0 meets the convergence criteria;
  - 2 theoretically impossible result was found during the iteration process for finding a smoothing spline curve with $\text{residual} = s$. Probable causes: $s$ is too small. There is an approximation returned but the corresponding weighted sum of squared residuals does not satisfy the condition $\text{abs}(\text{residual} - s)/s < \text{tol}$;
  - 3 the maximal number of iterations $\text{maxi}$ (default value 40) allowed for finding a smoothing curve with $\text{residual} = s$ has been reached. Probable cause: $s$ is too small. There is an approximation returned but the corresponding weighted sum of squared residuals does not satisfy the condition $\text{abs}(\text{residual} - s)/s < \text{tol}$.
  - 10 terrible internal error. Contact the developer (that would be me) and have your script ready.

Smooths the data set using natural B-splines as implemented in the package FITPACK by [Dierckx][1993].

9.3.10 bsplineval

Format: $y = \text{bsplineval}(x, z/, \text{idf}/)$,

Arguments:

1 $x$, real vector, abscissae of points at which $y$ is desired;
2 \( z \), B-spline compliant list, obtained by calling \texttt{bsplinefit};

3 \( ider \), integer vector, contains the derivatives of the approximation that are desired.

**Result:** \( y \), real matrix, contains values of spline approximation at \( x \) and possibly its derivatives (if \( ider \) is given).

### 9.3.11 bsplinefit2

**Format:** \( c = \texttt{bsplinefit2}(z, x, y, \text{/options/}) \),

**Arguments:**

1. \( z \), real vector or matrix, position of the points, or list, \( z = \text{//val;wgt//} \), of positions of the points and their weights. Please note: the weights are used only for scatter fit, and not for the grid fit (see below);
2. \( x \), real vector, abscissae of points, or list, \( x = \text{//val;wgt//} \), of abscissae and their weights;
3. \( y \), real vector, ordinatae of points, or list, \( y = \text{//val;wgt//} \), of ordinatae and their weights;
4. \( \text{options} \), list, the entries of which are the lists themselves, with entries \{//1;2//}, as follows
   - \( \text{degree} = \text{//1;2//} \), integer, degree of smoothing polynomial for coordinate 1 and 2, \( \text{degree.}[i] = 1, 3, 5 \), for \( i = 1, 2 \);
   - \( \text{range} = \text{//1;2//} \), two-element real vector, range for coordinate 1 and 2, \( \text{range.}[i] = [\min(x_{ii}), \max(x_{ii})] \), for \( i = 1, 2 \);
   - \( \text{max_knots} = \text{//1;2//} \), integer, maximal number of knots used along coordinate 1 and 2, \( \text{max_knot.}[i] \), for \( i = 1, 2 \);
   - \( \text{maxi} \), integer, maximal number of iterations to reach desired tolerance. Default value 40;
   - \( \text{tol} \), positive scalar, desired tolerance of the smoothing spline. Default value 0.001;
   - \( \text{rank_threshold} \), positive scalar, threshold for determining the effective rank of an over-determined linear system of equations, \( 0 < \text{rank_threshold} < 1 \);

**Result:** \( c \), B-spline compliant list, containing the following entries

- \( \text{coef} \), vector, coefficients of spline polynomial;
- \( \text{knot} \), list, \( \text{knot} = \text{//1;2//} \), real vector, position of knots along axes 1 and 2;
- \( \text{degree} \), list, \( \text{degree} = \text{//1;2//} \), integer, degree of spline polynomial (same as input) along axes 1 and 2;
- \( \text{residual} \), real scalar, mean-squared residual of the fit;
- \( \text{status} \), integer, nature of solution as follows:
  - \(-2\) weighted least-squares polynomial curve;
  - \(-1\) interpolating spline curve;
  - \(0\) meets the convergence criteria;
2 theoretically impossible result was found during the iteration process for finding a smoothing spline curve with residual = s. Probable causes: s is too small. There is an approximation returned but the corresponding weighted sum of squared residuals does not satisfy the condition \( \text{abs}(\text{residual} - s)/s < \text{tol} \);

3 the maximal number of iterations maxi (default value 40) allowed for finding a smoothing curve with residual = s has been reached. Probable cause: s is too small. There is an approximation returned but the corresponding weighted sum of squared residuals does not satisfy the condition \( \text{abs}(\text{residual} - s)/s < \text{tol} \).

10 terrible internal error. Contact the developer (that would be me) and have your script ready.

Smooths the 3-D data set using natural B-splines. Operates in two mode: scatter and grid, where the mode is determined by the size of matrix z. In scatter mode, z is a vector of the same length as x and y. In grid mode, z is a matrix where \( z.nr = \text{length}(y) \) and \( z.nc = \text{length}(x) \), and \( z[i,j] = z(x[j], y[i]) \).

### 9.3.12 bsplineval2

**Format:** \( z = \text{bsplineval2}(x, y, c/, iderx, idery/) \)

**Arguments:**

1. \( x \), real vector, first coordinate;
2. \( y \), real vector, second coordinate;
3. \( c \), B-spline compliant list, obtained by calling bsplinefit2;
4,5 \( iderx, idery \), integer, partial derivative to be calculated.

**Result:** \( z \), real matrix, contains values of spline approximation (or its partial derivative) given matrix-optimized coordinates \( x \) and \( y \).

### Generalized cross-validatory smoothing

### 9.3.13 gcvsplinefit

**Format:** \( c = \text{gcvsplinefit}(y, x/, options/) \)

**Arguments:**

1. \( y \), real vector, ordinata of points, or list, \( y =\langle< \text{val}; \text{wgt} >\rangle \), of ordinatae and their weights;
2. \( x \), real vector, abscissae of points, or list, \( x =\langle< \text{val}; \text{wgt} >\rangle \), of abscessae and their weights;
3. \( \text{options} =\langle< \text{degree}; \text{var}; \text{smooth}; \text{df} >\rangle \), list, the entries of which are
   - \( \text{degree} = 1, 3, 5, 7 \), a degree of smoothing polynomial (\( \text{spldeg} = 5 \), default);
   - \( \text{var} \), a positive scalar, the variance to be used in smoothing procedure;
   - \( \text{df} \), an integer, a number of degrees of freedom;
   - \( \text{smooth} \), a positive scalar, the smoothing parameter.
Please observe, it suffices to give only one of the parameters for `fit`.

**Result:** \( c \), gcv-compliant list, with entries

- \( m \), real two-column matrix, contains the abscissae of fitted points, \( x \), (first column) and coefficients of spline polynomial (second column);

- `degree`, integer, degree of spline polynomial (same as input);

- `info`, list, with entries
  1. `gcv_val`, real scalar, generalized cross validation value;
  2. `residual`, real scalar, mean-squared residual;
  3. `df`, integer, estimate of the number of degrees of freedom of the residual sum of squares;
  4. `smooth`, real scalar, smoothing parameter;
  5. `mse`, real scalar, estimate of the true mean squared error;
  6. `var`, real scalar, Gauss-Markov error variance.

Smooths data using the natural B-splines as implemented in the package GCVSPL by Woltring (1986).

### 9.3.14 `gcvsplineval`

**Format:** \( y = \text{gcvsplineval}(x,c,ider/) \),

**Arguments:**

1. \( x \), real vector, abscissae of points at which \( y \) is desired;
2. \( c \), gcv-compliant list, obtained by calling `gcvsplinefit`;
3. `ider`, integer vector, contains the derivatives of the splines that are desired.

**Result:** \( y \), real matrix, contains values of spline interpolant at \( x \) and possibly its derivatives (if `ider` is given).
Example: The following code creates a data set which noisely approximates the function \( y(x) = \exp(-x) \). In this case default settings suffice.

```
//
// file: main.r
//

haveplwin(2);

N = 10000;

dx = 1/N;
sy=0.02;

rand("normal",0,sy);

x = [0:N]'*dx;
y = exp(-x)+rand(N+1,1);

c = gcvsplinefit(y,x);
my = gcvsplineval(x,c);

plwin(1);
plstyle(["point","line"]);
plot( [x,y,my] );

plwin(2);
plot( [x, abs(y-my[;1])]);
```

Note: The package may hang occasionally or give nonsensical result. I find the following setting occasionally useful:

```
options = <<>>;
options.degree = 3; // use spline polynomials of third degree
options.var = 0.1; // assume variance of data points is given
..
gcvsplinefit( .. , .. , options);
..
9.4 Local Regression Models Fitting (loess)

This toolkit gives an user the access to the package *loess* by [Cleveland, Grosse, and Shyu](1992). The rlab interface closely follows that of the package.

The interface to *loess* is organized around a function list *loess* with the following elements,

<table>
<thead>
<tr>
<th>loess</th>
<th>anova</th>
<th>main</th>
<th>predict</th>
<th>setup</th>
</tr>
</thead>
</table>

Here, *loess.setup* is prefatory function by which a partial *loess* compatible list is created and filled with data and default values. On that list one can then perform a basic calculation using *loess.main* to obtain a full *loess* compatible list. Full lists can be mutually compared using *loess.anova*, that is, using analysis of variance, or one can make a prediction with or without calculation of confidence intervals.

**Note:** The original *loess* code from [www.netlib.org](http://www.netlib.org) was modified in order to compile without errors under gnu f77 compiler. See the README in flibs/loess directory of the rlab distribution. Some functions in the original code were slightly modified, and some completely removed for the sake of simplicity.

9.4.1 *loess*

**Format:** \( lo = \text{loess.setup}(Y, X) \),

**Arguments:**

1. \( Y \), a response variable, a real vector, or a list \( Y = \langle \text{val; wgt} \rangle \) of values and their weights;
2. \( X \), an explanatory variable, a real matrix, where the rows represent the values of the variable;

**Result:** a partial *loess* compatible list, \( lo = \langle \text{input; control; model} \rangle \), where each of the entries is a list itself. Its full structure is as follows,

- **input**, containing the following entries:
  - \( x \), an explanatory variable;
  - \( y \), a response variable;
  - weights, the weights (if not given, then default value of 1.0 is used for each \( y \) entry);
- **control**, containing the following entries:
  - cell, controls the size of \( k - d \) tree, default value 0.2;
  - iterations, default value 4, number of iterations for *loess* robust estimate;
  - statistics, default value “approximate”, also possible “exact”; 
  - surface, default value “interpolate”, also possible “direct”;
  - trace_hat, default value “wait.to.decide”;
- **model**, containing the following entries:
  - degree, default value 2 (quadratic), also possible 1 (linear);
  - drop_squares, default value \([0, 0, 0, 0, 0, 0, 0, 0]\);
  - family, default value “gaussian”, also possible is “symmetric”;
  - normalize, default value 1 (do the normalization), also possible 0 (do not normalize data);
parametric, default value \([0,0,0,0,0,0,0,0,0]\);

\(\text{span}\), neighborhood (smoothing) parameter, default value 0.5. Typically in \([0,1]\) range but can also be greater than unity;

Please see the loess manual for interpretation of the entries and their admissible values. I am too lazy to copy the loess manual. E.g., if one wants to change \(\text{span}\) from default value of 0.5 to 0.75, this is how it is done

```
1 //
2 // here I get x and y
3 //
4 ...
5
6 lo = loess.setup(y,x);
7 lo.model.span = 0.75;
8 ...
9 ...
```

**Abstract:** This function creates a loess compatible list from user data and sets default values for different entries of the list.

**II Format:** \(r = \text{loess.main}(lo)\),

**Arguments:** \(lo\), a loess compatible list. Typically one obtained by previous call to \(\text{loess.setup}\) and after some change in parameters.

**Result:** A full loess compatible list \(r = << \text{input}; \text{control}; \text{model}; \text{kd.tree}; \text{output} >>\).

**Abstract:** The full loess compatible list is a copy of the list \(lo\) with two additional sublists, namely,

- \(\text{kd.tree}\), with the elements:
  
  - \(a\);
  - \(\text{parameter}\);
  - \(\text{vert}\);
  - \(\text{vval}\);
  - \(\text{xi}\);

- \(\text{output}\), with the elements:
  
  - \(\text{diagonal}\);
  - \(\text{divisor}\);
  - \(\text{enp}\), the equivalent number of parameters;

\(\text{fitted.residuals}\);

\(\text{fitted.values}\);

\(\text{one.delta}\);

\(\text{two.delta}\);

\(\text{pseudovalues}\), used in inference if \(\text{family} = \text{“symmetric”}\);
robust;
\[s\] estimate of \(\sigma\), the standard error;
\[\text{trace\_hat}\];

\textbf{Note:} See the manual for more information.

\textbf{III Format:} \(yt = \text{loess.predict}(xt, r/, coverage/),\)

\textbf{Arguments:}

1. \(xt\), new values of the explanatory variable;
2. \(r\), a full \text{loess} compatible list, e.g., such as one obtained by the previous call to \text{loess.main};
3. \(coverage\), a number for the calculation of the confidence intervals. If \(coverage = 0\) the calculation is not performed;

\textbf{Result:} \(yt = << df; fit; \text{residual\_scale}; se_fit/; ci\_lower; ci\_upper >>\), an equivalent of a \text{predict} and \text{ci} structure in \text{loess}. Here,

\begin{itemize}
  \item \(df\), the degrees of freedom of the \(t\)-distribution upon which the confidence intervals are based;
  \item \(fit\), values of respond variable for \(xt\);
  \item \(se\_fit\), standard error of the \(fit\) values;
  \item \(\text{residual\_scale}\);
  \item \(ci\_lower, ci\_upper\), lower and upper bounds of the confidence interval of the given \(coverage\);
\end{itemize}

\textbf{Abstract:} This function performs the prediction of the response for the new values of the explanatory variable.

\textbf{IV Format:} \(a = \text{loess.anova}(r1, r2),\)

\textbf{Arguments:}

1, 2. \(r1, r2\), two full \text{loess} compatible lists. Typically obtained by calling \text{loess.main} for two different sets of parameters.

\textbf{Result:} \(a = << F\_value; Pr\_F; dfd; dfn >>\), a \text{loess} list of the type \text{anova}, containing analysis of variance for two calculated \text{loess} models, \(r1\) and \(r2\), namely,

\begin{itemize}
  \item \(F\_value\), value of \(F\)-statistics;
  \item \(Pr\_F\), its probability;
  \item \(dfd\), the denominator (degrees of freedom of an \(F\)-test);
  \item \(dfn\), the numerator (degrees of freedom of an \(F\)-test).
\end{itemize}
Example: Consider the example taken from the dloess package, file madeup.c.

```
//
// eg_loess1.r
//

// read in data from one_two.txt
one_two = [];
sx = reads("one_two.txt");
for (i in 1:sx.nr)
{
    one_two = [one_two, strtod(sx[i])];
}
one_two = reshape (one_two,2,100)';

// read in data from response.txt
response = [];
sy = reads("response.txt");
for (i in 1:sy.nr)
{
    response = [response, strtod(sy[i])];
}
response = response[:];

lo = loess.setup(response,one_two);
printf("loess list contains sublists:
	");
members(lo)
for (i in members(lo))
{
    printf(" sublist \"%s\" with members:\n\t", i);
members(lo.[i])
}

//
// first model has span = 0.5
//
lo.model.span = 0.5;
r1 = loess.main( lo );

//
// second model has span = 0.75
//
lo.model.span = 0.75;
r2 = loess.main( lo );
```
47 // compare the two
48 //
49 a = loess.anova(r1,r2);
50
51 // generic report
52 //
53 printf("Basic report:\n");
54 printf(" Number of observations = %g\n", one_two.nr);
55 printf(" Equivalent number of parameters = %g, %g\n", r1.output.enp, r2.output.enp);
56 printf(" Residual scale estimate = %g, %g\n", r1.output.s, r2.output.s);
57
58 //
59 // prediction
60 //
61 printf("Prediction for two points:\n");
62 xt = [ -0.5, 0.5; 0, 0]
63
64 // use 90% confidence interval
65 //
66 c = 0.9;
67 //
68 yt1 = loess.predict(xt,r1);
69 yt2 = loess.predict(xt,r2);
70 yt3 = loess.predict(xt,r1,c);
71
72 for (i in [1,2])
73 {
74   printf(" yt1[%g] = %g +- %g\n", i, yt1.fit[i], yt1.se_fit[i]);
75   printf(" yt2[%g] = %g +- %g\n", i, yt2.fit[i], yt2.se_fit[i]);
76 }
77
78 //
79 // plot the fitted model
80 //
81 haveplwin(1);
82 plwld ( [5,1] );
83 plstyle( ["point", "point"] );
84 plegend( ["raw data", "fit: span=0.5", "fit: span=0.75"] );
85 xlabel ("Data index");
86 ylabel ("Response");
87 plimits (0,100,0,25);
88 plot( [[1:100]',response, r1.output.fitted_values, r2.output.fitted_values] );
89
90
Chapter 10

Partial Differential Equations in 1-D

10.1 Introduction

As of now, rlab provides access to two numerical packages for solving systems of partial differential equations (PDE) in 1-D: a hyperbolic solver CLAWPACK by R. J. LeVeque et al. (2003) and a parabolic solver EPDCOL/PDECOL by Keast and Muir (1991). Some general theory on PDE can be found in, e.g. Evans (1998).

10.2 CLAWPACK

The solver CLAWPACK is intended for hyperbolic-type PDE,

$$K(x)\partial_t \vec{q} + \partial_x \vec{F}(t, x, \vec{q}) = \vec{S}(t, x, \vec{q}), \quad (10.1)$$

where the initial state of the system \(\vec{q}(t = 0, x)\) is given. The scalar function \(K = K(x)\) is called capacity, \(\vec{F}\) is called a flux function, while \(\vec{S}\) is the source function. The goal of the solver is to find \(\vec{q} = \vec{q}(t, x)\). One of the authors of CLAWPACK also wrote a book on numerical solutions of hyperbolic problems, see R. LeVeque (2002).

Note: The rlab implementation of CLAWPACK can deal with non-zero source functions. User may choose between the two time integration methods: an (explicite, for non-stiff problems) modified Euler method of Collatz (Stoer & Bulirsch 1992), and (default, explicite-implicite, for non-stiff to mildly stiff problems) Adams’ method by Shampine and Gordon (n.d.).

10.2.1 claw1

Format: \(Q2 = \text{claw1}(/K/, R, /S/, Q1, T)\)

Arguments:

1. \(K = \text{function}(x)\), a capacity function of the media;
2. \(R = \text{function}(t, x, q_l, q_r)\), a Riemann solver for Eq. (10.1);
3. \(S = \text{function}(t, x, q)\), a source function;

335
4 $Q_1 = [x, q(x,t_1)]$ a matrix containing mesh and the initial conditions $q(x,t_1)$;
5 $T = [t_1, t_2]$ is an integration interval.

**Result:** $Q_2 = [x, q(x,t_2)]$, a result of integration.

**Riemann Solver Function**

Theory of Riemann solvers and practical examples are described in [R. LeVeque (2002)](RiemannSolverFunction). In a nutshell: for a Riemann solver it is necessary to know the flux function $\vec{F} = \vec{F}(t, x, \vec{q})$ and its Jacobian $DF$ in matrix form,

$$DF_{i,j}(t, x, \vec{q}) = \frac{\partial F_i}{\partial q_j}. \quad (10.2)$$

The eigenvalues of $DF$ have to be real for the problem to be hyperbolic in the first place. These eigenvalues are called speeds, $c_j$, and their eigenvalues, $\hat{e}_j$, represent the directions of the waves. Once these are known, one has to numerically implement the Rankine-Hugoniot condition (relationship between the jump of the function $q$ and the function $F$ at the characteristics), and find the projections of the jump onto the directions of waves. E.g., a shock wave is made of a discontinuity in $\vec{q}$ at a cell boundary point $x_{i-1/2}$ at the time $t$, which is described as a pair $\vec{q}_{l}(i)$ (left limit at the cell wall) and $\vec{q}_{r}(i-1)$ (right limit at the same cell wall), onto the eigenvectors of $DF$. This is done by first calculating the projections, call it $\vec{\alpha}$, as

$$\vec{\alpha} = [\hat{e}_1, \hat{e}_2 \ldots]^{-1} \ast (\vec{q}_{l} - \vec{q}_{r}). \quad (10.3)$$

Observe, from a numerical viewpoint. If $DF$ is a matrix and $ql$ and $qr$ are the vectors, then in rlab $\vec{\alpha}$ is calculated as

1.. 
2 vDF = eig(DF).vec;
3 alfa = inv(vDF)*(ql-qr);
4..

This is because the rlab’s `eig` function returns a matrix the columns of which are the respective eigenvectors. Check for non-zero complex parts. What function $R$ has to return is a stacked matrix of waves and speeds. A wave, $\vec{w}_j$ is defined as $\vec{w}_j = \alpha_j \cdot \hat{e}_j$. In rlab script all this is written as

1 R = function(t,x,ql,qr){
2  //
3  // Riemann solver for RLaB/CLAWPACK
4  // Input:
5  // t -> time
6  // x -> x(i-1/2), the wall between the cells 'i-1' and 'i'
7  // ql -> ql(i), field 'q' in the cell 'i' on the left wall
8  // qr -> qr(i-1), 'q' on the right
9  // Return (1)
10  // y = [waves;speed];
11  //
12  ..
10.2. CLAWPACK

// calculate matrix DF(t, x, 0.5*ql+0.5*qr)

eDF = eig(DF);

vDF = real(eDF.vec); // real(..) is just to be sure
speed = real(eDF.val); // same

alfa = inv(vDF)*(ql-qr); // alfa is a column vector
waves = vDF .* alfa'; // multiply each column in vDF by a row in alfa

return [waves; speed];

Note: DF is assumed a continuous function of \( \vec{q}, x \) and \( t \). It is good approximation to assume that in a call to function \( R \), the matrix \( DF \) is calculated at \( 2\vec{q} = \vec{q}_l + \vec{q}_r \). If \( DF \) is a constant matrix than this is exact.

Note: The eigenvalue problem for \( DF \) and its inverse are calculated at every mesh point \( x \). This may be time consuming. Check if an expression for the vectors and velocities can be derived explicitly.

Note: Above procedure is roughly correct. In some cases additional tricks are required. E.g., for Burgers' equation under some conditions there exist an entropy fix, which has to be applied. In this case, the user has to calculate on its own the fluxes on the left and right between the cells defined by the mesh. If a basic Riemann solver cannot do the job, one may proceed to calculate the fluxes on its own. There are two fluxes that need to be calculated at the cell's boundary, \( amdq \), or left or minus, and \( apdq \) or right or plus. Under normal circumstances, rlab does it for you, where \( apdq = \sum_{c_j>0} c_j \cdot \vec{w}_j \) and \( amdq = \sum_{c_j<0} c_j \cdot \vec{w}_j \). However, when this does not suffice, you also have to provide \( apdq \) and \( amdq \) on your own. Then the Riemann solver function has to return these fluxes as follows,

R = function(t,x,ql,qr){
    //
    // Riemann solver for RLaB/CLAWPACK
    // Input:
    //  t -> time
    //  x -> x(i-1/2), the wall between the cells 'i-1' and 'i'
    //  ql -> ql(i), field 'q' in the cell 'i' on the left wall
    //  qr -> qr(i-1), 'q' on the left wall
    // Return (2):
    //  y = [waves,amdq,apdq;[speed,0,0]];
    //
    // calculate matrix DF(t, x, 0.5*ql+0.5*qr)
    eDF = eig(DF);
    vDF = real(eDF.vec); // real(..) is just to be sure
    speed = real(eDF.val); // same
    alfa = inv(vDF)*(ql-qr); // alfa is a column vector
    waves = vDF .* alfa'; // multiply each column in vDF by a row in alfa
    // this did not work and we have to calculate
CHAPTER 10. PARTIAL DIFFERENTIAL EQUATIONS IN 1-D

// fluxes 'amdq' and 'apdq' on our own
.. return [waves, amdq, apdq; [speed,0,0]]; // OBSERVE THE STRUCTURE OF THE RESULT !!!!
**Abstract:** order is used to set up an integration formula, while sourcesp determines how the splitting of integration between the spatial and the temporal part will be performed.

- **Format:** clawparams.cour(maxcour, cour, maxi)

**Arguments:**

1. maxcour, a maximal courant number allowed;
2. cour, a desired courant number;
3. maxi, a maximal number of iterations allowed to reach the desired courant number.
Example: Consider a rlab code that solves the acoustic problem in 1-D with a reflecting boundary on the right, and zero-order extrapolation (absorbing) boundary on the left.

```matlab
// test for clawpack 1-d solver:

// acoustic example with/without damping and capacity function
//
// K q_t + (F(q))_x = S
//
// where q = [ p(t,x); u(t,x) ],
// the flux function is F(q) = [0,1; 1,0] q,
// the source function is S(q) = -q, or S(q) = 0,
// and capacity function is K(x) = 1, or K(x) = 2-x.^2,
//
// with x on [-1,1]
// BC: Rigid wall on the right, free on the left

NITER = 2000; // test for segmentation faults, memory leaks etc.

xlo = -1;
xhi = 1;

nx=512;
dx = (xhi-xlo)/nx;
x = (xlo:xhi:dx)';
q0 = exp(-(x-.05).^2/0.1);
qu = [x,q0,zeros(q0)];

// Riemann solver: DF is a constant so these can be calculated
// only once

DF = [0,1;1,0];
eigDF = eig(DF);
speed = real(eigDF.val);
vedf = real(eigDF.vec);
iedf = inv(vedf);

// Riemann solver function: finds waves and speeds in a Hyperbolic problem

R = function(t,x,ql,qr)
{
    // DF
    global(iedf,vedf,speed);
    aDF = iedf * (ql-qr);
}
```
waves = vedi .* aDF';
return [ waves ; speed ];
};

// Source function
//
S = function(t,x,q)
{
  return -q;
};

// Capacity function
//
K = function(x)
{
  return 2.-x.^2;
};

// 2 layers, left absorbing, right reflecting
for( i in 1:NITER)
{
  qn = q;
  for(dt in [1/64:4:1/64])
  {
    spinner();
    //qn = claw1(R,S,qn,[dt-1/64,dt]);   // no capacity, with damping
    qn = claw1(K,R,,qn,[dt-1/64,dt]);   // capacity, no damping
    tl = "CLAWPACK 1-D: dt = "+text(dt); pltitle(tl);
    //sleep(0.1);
    data.[1] = q;
    data.[2] = qn;
    pllimits(,-1.5,1.5);
    plot( data );
  }
}
10.3 EPDCOL and BACOL

These are two parabolic solvers, *epdcol* by Keast and Muir (1991) and *bacol* by Wang, Keast, and Muir (2004), that both utilize B-splines. A parabolic problem is of the form,

\[ u_t = f(t, x, u, u_x, u_{xx}) \]

The two solvers differ in types of boundary conditions they can handle,

- *epdcol* can handle the boundary conditions in the form,
  \[ B(x_{L,R}, u, u_x) = Z(t) \]
  (10.5)
  where \( B \) and \( Z \) are known vector functions, while
- *bacol* handles the boundary conditions in the form,
  \[ B(t, x_{L,R}, u, u_x) = 0 \]
  (10.6)
  where \( B \) is a known vector function.

Method employed by the algorithm is the collocation method, by which a function is converted into set of coefficients of a specific polynomial. In *epdcol* a fixed mesh is employed, while *bacol* uses an adaptive mesh. These methods typically appear in the boundary value problems.

10.3.1 pdecol

**Format:** \( wn = pdecol(npde, f, Df, DbL, DbR, x, w0, T, options) \)

**Arguments:**

1. \( npde \) an integer, number of PDEs making the system (10.4);
2. \( f = function(t, x, u, u_x, u_{xx}) \) a vector function the output of which is a real matrix 1-by-\( npde \) representing time derivatives \((u_t)_i = f_i, i = 1 \ldots npde\);
3. \( Df = function(t, x, u, u_x, u_{xx}) \), is a list of jacobians of the function \( f \), \( Df = [ dfdu; dfdux; dfduxx ] \), where \( dfdu = \partial f/\partial u \), \( dfdux = \partial f/\partial u_x \), and \( dfduxx = \partial f/\partial u_{xx} \). Each matrix has to be of the size \( npde-by-npde \).
4. 4.5 \( DbL, R = function(t, u, u_x) \) is a differential of the boundary condition \( B(u, u_x) = Z(t) \), \( DbL, R = [ dbdu; dbdux; dzdt ] \), where \( dbdu = \partial B/\partial u \), \( dbdux = \partial B/\partial u_x \) and \( dzdt = dZ/dt \), each a matrix \( npde-by-npde \).
5. \( x \) is a column vector which determines the mesh, from which the collocation points will be chosen.
6. \( w0 = function(x) \) is the function determining the initial condition for the vector function \( u = u(x) \).
7. \( T = [t_i, \ldots t_f] \) is the time integration intervals. The solution at points \( t_j \) is going to be available to the user upon termination of the calculation.
8. \( options \) a list containing following elements that control the calculation, \( options = [ kord; erel; dt; ncc; imethod; maxder; nogauss; stdout ] \), where
   - \( kord \), the order of the piecewise polynomial space to be used. Default \( kord = 4 \);
10.3. EPDCOL AND BACOL

- `erel`, the relative error bound. Default `erel = 1e - 4`.
- `dt`, initial time step. Default `1e - 10`, may be too small for some problems. See example codes.
- `ncc`, the number of continuity conditions. Default `ncc = 2`.
- `imethod`, choice of the ODE integration method, 1 for the Adams method (generalization of Crank-Nicholson), 2 for the backward differentiation formula.
- `maxder`, maximum order of time integration formula allowed. Default `maxder = 5`.
- `nogauss`, use if Gauss-Legendre collocation points are not desired (`nogauss = 1`). Otherwise leave default `nogauss = 0`.
- `stdout`, a name where output messages generated by the solver are posted. Observe, the solver `EPDCAL` is not that much verbose.

**Result:** a matrix `wn` containing the blocks of `npde`-columns sections representing the values of the solution at the times given in `T`. The number of rows in `wn` is the same as in `x`.

**Note:** A speed of the solver can be increased in the case when any of the list-functions, `Df`, `DbL` or `DbR`, is a constant. To tell the solver that a particular function is a constant add an entry, `const` to the return values. The solver will check if such entry exists, and if so, will store the whole list locally. In consecutive calls to the particular list-function, solver will access the stored value rather than executing the appropriate script.

### 10.3.2 bacol

**Format:**

\[ wn =bacol(npde,f,Df,bL,bR,DbL,DbR,x,w0,T,options), \]

**Arguments:**

1. `npde` an integer, number of PDEs making the system \((\text{10.4})\);
2. `f = function(t,x,ux,uxx)` a vector function the output of which is a real matrix 1-by-`npde` representing time derivatives \((u_t)_i = f_i, i = 1 \ldots npde\);
3. `Df = function(t,x,ux,uxx)`, is a list of jacobians of the function \(f\), `Df =<<dfdu;dfdux;dfduxx>>`, where `dfdu = ∂f/∂u`, `dfdux = ∂f/∂ux`, and `dfduxx = ∂f/∂uxx`. Each matrix has to be of the size `npde`-by-`npde`;
4,5. `bL,R = function(t,u,ux)` is a function describing the boundary condition \(B(t,u,ux) = 0\), a matrix of size 1-by-`npde`;
6,7. `DbL,R = function(t,u,ux)` is a differential of the boundary condition \(B(t,u,ux) = 0\), `DbL,R =<<dbdu;dbdux;dbdt>>`, where `dbdu = ∂B/∂u`, `dbdux = ∂B/∂ux`, each a matrix `npde`-by-`npde`, and `dbdt = ∂B/∂t`, a matrix of size 1-by-`npde`;
8. `x` is a column vector which determines the mesh, from which the collocation points will be adaptively chosen.
9. `w0 = function(x)` is the function determining the initial condition for the vector function \(u = u(x)\).
10. `T = [t_i, \ldots t_f]` is the time integration intervals. The solution at points \(t_j\) is going to be available to the user upon termination of the calculation.
11 options, a list containing following elements that control the calculation,

- \( kord \), the order of the piecewise polynomial space to be used. Default \( kord = 4 \);
- \( atol \), the absolute tolerances for each component of the solution, or one value for all components.
  Default value, a scalar, \( atol = 1e^{-3} \);
- \( rtol \), the relative tolerances for each component of the solution, or one value for all components.
  Default value, a scalar, \( erel = 1e^{-2} \).
- \( idir = 0,1 \), specify whether at least one of the boundary conditions is a dirichlet one. Default value 0;
- \( imax \), a maximum number of subdivisions of the spatial domain. Default value 10,000;
- \( isover = 0,1 \), the way integration is to go about the internal time points. If 0, the algorithm
  performs integration up to the exact time point \( t_i \). If 1, the algorithm may go over \( t_i \) and
  interpolate the values for \( u(t_i) \). Default value 0;
- \( dt \), a initial time step in integration. Default value 0 (let the code decide);
- \( stdout \), a name where output messages generated by the solver are posted. Observe, the solver
  BACOL is not that verbose.

**Result:** A matrix \( wn \) containing the blocks of \( npde \)-columns sections representing the values of the solution
at the times given in \( T \). The number of rows in \( wn \) is the same as in \( x \).
Example: Consider the script `eg_pde2.r` which solves the PDE,

\[ u_t = u_{xx} + \pi^2 \cdot \sin(\pi \cdot x) \]  

(10.7)

with boundary conditions \( u(x_{L,R}) = 0 \) and \( u0 = 1 \).

```
1 //
2 // test for pdecol 1-d solver:
3 //
4 // u_t = u_xx + pi*pi*sin(pi*x)  (1)
5 //
6 // x in [0,1], with boundary conditions u(0,t) = 1 and u(1,t) = 1
7 // and initial condition u(x,0) = 1.
8
9 npde = 1;
10
11 //
12 // f
13 //
14 f = function(t,x,u,ux,uxx)
15 {
16   global(pi);
17   rval = uxx + pi*pi*sin(pi*x);
18   return rval;
19   
20   Df = function(t,x,u,ux,uxx)
21   {
22     rval<<>>;
23     rval.const = 1;  // tell solver Df is a constant.
24     rval.dfdu = 0;
25     rval.dfxux = 0;
26     rval.dfxuxx = 1;
27     return rval;
28   
29   //
30   // DbL,R, the boundary conditions on x=xL and x=xR
31   //
32   DbL = function(t,u,ux)
33   {
34     // Left boundary fixed: u = 0
35     // B(u) = u
36     // Z(t) = 0
37     rval<<>>;
38   
39   }`
CHAPTER 10. PARTIAL DIFFERENTIAL EQUATIONS IN 1-D

```plaintext
43    rval.const = 1; // tell solver DbL is a constant.
44    rval.dbdu = 1;
45    rval.dbdux = 0;
46    rval.dzdt = 0;
47    return rval;
48    }
49
50    DbR = function(t,u,ux)
51    {
52        // Right boundary fixed: u = 0
53        // B(u) = u
54        // Z(t) = 0
55        global(pi);
56        rval=<<>>;
57        rval.const = 1; // tell solver DbR is a constant.
58        rval.dbdu = 1;
59        rval.dbdux = 0;
60        rval.dzdt = 0;
61        return rval;
62    }
63
64    w0 = function(x)
65    {
66        global(pi);
67        rval = 1;
68        return rval;
69    }
70
71    //
72    // time interval for integration
73    //
74    dt = 1/128;
75    ti = 0;
76    tf = 1;
77    T = [ti:tf:dt];
78    options=<<>>;
79    options.stdout = rlab_initstderr();
80    options.dt = 1e-6;
81    options.imethod= 2;
82    options.erel = 1e-4;
83
84    //
85    // resolution of the functions
86    //
87    sizeS = [128];
88    NITER = 1;
```
10.3. **EPDCOL AND BACOL**

```matlab
xlo = 0;
xhi = 1;
tic();
nx = sizeS(1);
dx = (xhi-xlo)/nx;
x = (xlo:xhi:dx)';
wn = pdecol(npde,f,Df,DbL,DbR,x,w0,T,options);
exno = 1;
wmax = max(max(wn));
wmmin = min(min(wn));
```
// test for bacol 1-d solver:

//  u_t = u_xx + pi*pi*sin(pi*x)  (1)

// x in [0,1], with boundary conditions u(0,t) = 1 and u(1,t) = 1
// and initial condition u(x,0) = -x*(1-x).

npde = 1;
NITER = 100;

// f
f = function(t,x,u,ux,uxx)
{
  global(pi);
  rval = uxx + pi*pi*sin(pi*x);
  return rval;
};

// Df
Df = function(t,x,u,ux,uxx)
{
  rval = <<>>;
  rval.const = 1;
  rval.dfdx = 0;
  rval.dfdux = 0;
  rval.dfduxx = 1;
  return rval;
};

// bL,R, the boundary conditions on x=xL and x=xR
bL = function(t,u,ux)
{
  // Left boundary fixed: u = 0
  // B(u) = u
  rval = u;
  return rval;
};
DbL = function(t,u,ux)
{

// Left boundary fixed: u = 0
// B(u) = u
rval=<<>>;
rval.const = 1;
rval.dbdu = 1;
rval.dbdux = 0;
rval.dbdt = 0;
return rval;
);
bR = function(t,u,ux)
{
// Right boundary fixed: u = 0
// B(u) = u
rval = u;
return rval;
};
DbR = function(t,u,ux)
{
// Right boundary fixed: u = 0
// B(u) = u
global(pi);
rval=<<>>;
rval.dbdu = 1;
rval.dbdux = 0;
rval.dbdt = 0;
return rval;
};
w0 = function(x)
{
global(pi);
rval = -x*(1-x);
return rval;
};

// time interval for integration
//
dt = 1/256;
ti = 0;
tf = 1;
T = [ti:tf:dt];
options=<<>>;
options.stdout = rlab_initstderr();
options.erel = 1e-4;
options.eabs = 1e-4;
CHAPTER 10. PARTIAL DIFFERENTIAL EQUATIONS IN 1-D

```matlab
// resolution of the functions
sizeS = [128];
xlo = 0;
xhi = 1;
tic();

nx = sizeS[ 1 ];
dx = (xhi-xlo)/nx;
x = (xlo:xhi:dx)';

wn = bacol(npde,f,Df,bL,bR,DbL,DbR,x,w0,T,options);
```
Example: The following is the code used for plotting the result.

```plaintext
leg=blank(0,0);
for (i in 1:nPde)
{
  leg = [leg, "u(" + text(i) + ")"]; 
}

tstr = "Fig.1: pdecol/epdcol for Example No. " + text(exno);

if (!isempty(wn))
{
  plwin(1);
  plegend( leg );
  xlabel ( "x" );
  ylabel ( "u(x)" );
  plimits(xlo, xhi, wmin, wmax);
  pltitle( tstr );
  for (i in 1:(wn.nc-nPde+1):nPde)
  {
    plot( [x, wn[;i:(i+nPde-1)]]);
    sleep (0.1);
  }
}
```

Note: The boundary conditions need to be consistent with the initial value of the vector function $u$. 
CHAPTER 10. PARTIAL DIFFERENTIAL EQUATIONS IN 1-D
Chapter 11

Pattern Recognition

11.1 Patterns, Their Recognition, and Data Analysis

Since rlab 2.3.2.0 many functions have been changed internally to ignore NaN’s, e.g., \texttt{min}, \texttt{max}, \texttt{sort}, and so forth.

The solvers in this group assume that matrix comprises of rows of data.

Some of the pattern recognition solvers are converted from the C-cluster library by de Hoon et al. (2008), or rewritten from buggy, now extinct, SPRANLIB by Hoekstra, Kraaijveld, de Ridder, and Schmidt (1996).

11.1.1 \texttt{isrelmono}, \texttt{isabsmono}

\textbf{Format:} \texttt{i = isrelmono(x, \epsilon)}

1. \(x\), real vector;

2. \(\epsilon\), positive scalar. Some restrictions apply: for \texttt{isrelmono} \(0 < \epsilon < 1\);

\textbf{Result:} \(i\), integer scalar -1,0,1.

\textbf{Abstract:} Checks if \(x\) is absolute, or relative monotone increasing or decreasing function given the error \(\epsilon\). Returns 1 if it is monotone increasing, -1 monotone decreasing, or 0 if neither. \(x\) is absolute monotone increasing if

\[ x_{i+1} \geq x_i - \epsilon, \forall i. \tag{11.1} \]

and relative monotone increasing if

\[ x_{i+1}/x_i \geq 1 - \epsilon, \forall i. \tag{11.2} \]

For monotone decreasing, switch \(\geq\) to \(\leq\), and \(\epsilon \rightarrow -\epsilon\) in two expressions above. Here “decreasing” strictly speaking means “non-increasing,” and the same for “increasing.”

11.1.2 \texttt{locextri}

\textbf{Format:} \texttt{i = locextri(x, opts)}

1. \(x\), real vector, \(x = [x_i]_{i=1:N};\)
2. *opts*, list with entries "<eabs;erel;dir;edges>", where

- *dir*, string, “min” or “max” for the direction of search;
- *eabs*, *erel*, non-negative scalars, used to establish approximate relationships “less than” and “greater than,” as follows,

\[ x_i < x_{i+1} \equiv x_i < (1 - erel) \cdot x_{i+1} - eabs, \]  
and \[ x_i > x_{i+1} \equiv x_i > (1 + erel) \cdot x_{i+1} + eabs. \]

- *edges*, string containing “l” or “r” or both, to indicate whether left or right edge of the array should be ignored in search for the extremae.

**Abstract:** Finds indices of local extremae in an array (vector).

**Note:** The function is capable of recognizing the ranges of indices. To establish the range, the first element is used. Again, for the same example, the set of indices \( I' \) would be formed so that \( i^*_1 = \min I' \) and for all \( i^* \in I' \),

\[ x^*_i \approx x^*_{i^*_1}, \]

in the above sense of not being either “greater than” or “less than.” Then, the maximum would satisfy, \( x^*_{i^*_1} > x_{\max I' + 1} \) (first element after the range) and \( x^*_{i^*_1} > x^*_{i^*_1 - 1} \) (last element before the range). An example of calculation of local minimae of an image histogram is given in Fig. (11.1). For range, the function returns the midpoint of the range.

Also note, that the function ignores nans and infinities at the edge of the array irrespectively of the parameter *edges*. 

---

**Figure 11.1:** Thresholds calculated using *locextri*(.,<< .dir = “min”) >> over a kullback information measure \( H \) from a histogram of a black and white image.
11.1.3 distance

Format: \( c = \text{distance}(x_1, x_2, \text{metrics}, wgt) \)

1. \( x_1 \), real matrix of rows of data;
2. \( x_2 \), optional real matrix of rows of data;
3. \( \text{metrics} \), optional string scalar, choice of metric, “s” for spearman’s, “e” for euclid, “b” for cityblock, “m” for cityblock max, “p” for pearson, “a” for absolute pearson, “u” for pearson uncentered, “x” for absolute uncentered pearson and “k” for kendall. “e” is default.
4. \( wgt \), optional row-vector, has same number of entries as \( x_1 \) and \( x_2 \) columns, represents the arbitrary weights used in calculation of metric,

\[
d(x_1^i, x_2^j) = \sum_k w_k \mu(x_1^i,k, x_2^j,k),
\]

(11.6)

where \( \mu \) is the metrics function.

Please note: It is up to user to decide whether to normalize \( w \) or not. If \( w \) is omitted, then \( w_k = 1, \forall k \) is assumed for distance calculation. \( m \)-th component can be omitted from summation by setting \( w_m = \text{nan}(\) \).

Result: \( c \), real matrix of size \( \text{no-of-rows-in-x1} \) by \( \text{no-of-rows-in-x2} \) where the \((i, j)\) element represents the distance from the \( i \)-th element of dataset \( x_1 \), to the \( j \)-th element of dataset \( x_2 \). If matrix \( x_2 \) is omitted, then the distances are computed between the data in \( x_1 \).

11.1.4 cluster

I Format: \( c = \text{cluster.pao}(data, \text{thresh}/, \text{metrics}/) \),

Arguments:

1. \( data \), real matrix of rows of data;
2. \( \text{thresh} \), positive real scalar, maximal distance between the points in the cluster.
3. \( \text{metrics} \), string scalar, choice of metric. Default value is “e” for euclidean.

Result: \( c = << \text{feature; size; val} >> \).

Abstract: Finds minimum number of clusters through iterations such that the distance between the cluster center and a datum is less then \( \text{thresh} \). The solver performs one pass through the data. The solver does not tolerate \( \text{nan}(\)\)’s.

II Format: \( c = \text{cluster.knn}(data, ncl, \text{opts}) \)

III Format: \( c = \text{cluster.iso}(data, ncl, \text{opts}) \)

Arguments:

1. \( data \), data matrix;
2. \( ncl \), anticipated number of cluster centres;
3. opts is a list of options \(\ll metric; weight; maxi; method \gg\), with metric and weight as above, 
maxi, integer scalar for maximum number of attempts, and method is “m” or “M” for median, 
“a” or “A” for average, and “x” for medoid method for calculation of cluster centers.

Result: \(c = \ll feature; size; val \gg\).

Abstract: Finds \(ncl\) cluster centers through the \(k\)-nearest-neighbours (\(nn\)) or through isodata algorithm (\(iso\)) of R. Duda and P. Hart, “Pattern Classification and Scene Analysis,” John Wiley and Sons, 1973.

All cluster functions return a list \(\ll feature; size; val \gg\), where \(val\) is a matrix which rows are ordered cluster centers, \(size\) is a vector containing the sizes of each cluster from the input \(data\), while \(feature\) is a vector with the cluster assignment of \(data\).

11.1.5 classify

I Format: \(c = classify.parzen(x, dataset)\)

II Format: \(c = classify.knn(x, dataset, k)\)

III Format: \(c = classify.fisher(x, dataset/, class/)\)

IV Format: \(c = classify.fisherq(x, dataset/, class/)\)

Arguments:

1. \(x\), real matrix, row-wise data entries;
2. \(dataset = \ll data; feature \gg\), list where \(data\) and their \(feature\) comprise the learning set for classification.
   While \(data\) is strictly numerical matrix, \(feature\) vector can be numerical, or verbal.
   Obviously, \(x\) and \(data\) should have same dimension (number of columns), while \(feature\) should have the same length as \(data\) has rows.
3. \(knn\)
   \(k\), number of nearest neighbors upon which the decision should be made.
   \(fisher, fisherq\)
   \(class\), if \(data\) has more then two features, then \(class\) specifies which two features will be used in calculation of discriminant.

Result: \(c\), suggested features of the data \(x\) obtains either through \(Parzen\), or through \(k\)-nearest neighbour classification scheme.

Fisher and quadratic Fisher methods solve two-class problem using linear, or quadratic Fisher discriminant.

Arguments:

11.1.6 sammon

Format: \(map = sammon(x, dim, opts)\)

Arguments:
11.1. PATTERNS, THEIR RECOGNITION, AND DATA ANALYSIS

1. $x$, real matrix, row-wise data entries; or list $<<data; feature>>$.

2. $\text{dim}$, positive integer smaller than dimension of $data$.

3. $\text{opts}$, list of solver (minimizer) options $<< \text{imethod}; \text{lrate}; \text{momentum}; \text{maxi}; \text{map} >>$, where $\text{imethod}=0$ for steepest descent (uses first derivatives) and 1 for Newton method (uses second derivatives, as well), $\text{lrate}$ is the learning rate (0.001 is default) and $\text{momentum}$ (default value 0.1), $\text{maxi}$ is the maximal number of iterations used in finding the minimum of Sammon stress (default value 1000), and optional $\text{map}$ is the initial value of Sammon mapping that solver will try to improve upon (default value is a matrix filled with uniform random numbers from $[0, 1]$).

Result: $\text{map}$, dimension-reduced map of data $x$ find through minimization of Sammon stress.
Chapter 12

3-D Surfaces

12.1 GTS

GTS stands for The GNU Triangulated Surface Library. It provides workspace/environment for creating and manipulating surfaces, and their elemental parts: triangles (faces), line segments (edges) and points (vertices).

The functions in this library allow *rlab* to create and manipulate single surfaces and perform boolean operations between the surfaces.

The library is organized around surfaces as named objects in the GTS-managed workspace. User can create, delete or manipulate surface(s) as one could do that within the GTS. In addition there exist functions which export named GTS surfaces into *rlab* variables, or convert proper *rlab* variables into GTS surface objects. In that sense, *rlab* approach is different from, say python approach as in *pygts*, in which the GTS objects are tied to python variables.

All functions in this library are in a list, *gts.surface* to emphasize the *rlab* approach as named-surfaces centric.

In *rlab* parlance a GTS compliant surface \( s \) is a list \( s = \langle v; e; f; c \rangle \), which entries are

- \( v \), 3-col matrix of vertices comprising the surface.
- \( e \), 2-col integer matrix of edges comprising the surface, where the row entries are the indices of its vertices in \( v \).
- \( f \), 3-col integer matrix of the triangles comprising the faces of the surface, where the row entries are the indices of its edges in \( e \).
- \( c \), 3-col matrix containing the color of each or of all faces. If omitted the default value \([1,1,1]\) is used.
Utility Functions not in this Library

12.1.1 validgts

Format: \( u = \text{validgts}(s) \)

Arguments:

1. \( s = \langle v; e; f; c \rangle \), an rlab list containing GTS compliant surface.

Result: 0, 1 depending whether the rlab variable \( s \) contains GTS compliant surface or not.

Note: This function is a part of \texttt{libgts} library which is distributed with rlab irrespectively of shared object library.

12.1.2 gts_to_xyz

Format: \( \mathbf{A} = \text{gts} \_\text{to} \_\text{xyz}(s) \)

Arguments:

1. \( s = \langle v; e; f; c \rangle \), an rlab list containing GTS compliant surface.

Result: Internal function used for plotting programs to plot GTS surface in 3-D. Currently, Gnuplot is supported for both, plotting the 3-D meshes of surfaces (as chopped lines), and their sides (as 'polygons'). In Fig. (12.1) one can see a Gnuplot generated output of a few surfaces.

Note: This function is a part of \texttt{libgts} library which is distributed with rlab irrespectively of shared object library.

Geomview

About: Geomview is an interactive 3D viewing program. Geomview lets you view and manipulate three-dimensional objects: you use the mouse to rotate, translate, zoom in and out, and so on. Geomview can be used as a standalone viewer for static objects, or as a display engine for other programs which produce dynamically changing geometry. Geomview can display objects described in a variety of file formats. Geomview comes with a wide selection of example objects, and you can create your own objects too.

12.1.3 plot_gts_surface

Format: \texttt{geomview.plot_gts_surface(scene, obj)}

Arguments:

1. scene, string, GCL compliant name of the scene.

2. obj, string, name of the valid surface object from GTS workspace.

Abstract: Plot an GTS surface object on a scene using \texttt{geomview}. In Fig. (12.1) one can see a Gnuplot generated output of a few surfaces.
12.1.4 snapshot

Format: geomview.snapshot(fn/, fmt/)

Arguments:

1. fn, string, filename in which the current Camera view will be saved in post-script format.
2. fmt, string, “ps” or “ppm” for two image formats that geomview supports.

Abstract: Plot current camera view from geomview to a file in post-script format.

Note: In Fig. (12.1) one can see a Gnuplot and geomview generated outputs of few surfaces.

Administrative Functions

12.1.5 ls

Format: u = gts.surface.ls()

Arguments:

Result: u, string array, names of surfaces currently defined in the GTS workspace.

12.1.6 info

Format: u = gts.surface.info(sname)

Arguments:

1. sname, string, name of the surface.

Result: u, list ≪ nr_edge; nr_face; nr_vertex ≫, count of edges, faces and vertices of the surface which name is given by the variable sname in the GTS workspace.

12.1.7 stat

Format: u = gts.surface.stat(sname)

Arguments:

1. sname, string, name of the surface.

Result: u, list ≪ n_boundary_edges; n_duplicate_edges; n_duplicate_faces; n_incompatible_faces; n_non_manifold_edges; nr_edge; nr_face; nr_vertex ≫, which provides some statistical properties of the vertices, edges and faces comprising the surface which name is given by the variable sname in the GTS workspace.
12.1.8 qstat

**Format:** \( u = \text{gts.surface.qstat}(\text{sname}) \)

**Arguments:**

1. \textit{sname}, string, name of the surface.

**Result:** \( u \), list \( \langle \text{edge\_angle}; \text{face\_area}; \text{face\_length}; \text{face\_quality} \gg \), which provides some more statistical properties of the vertices, edges and faces comprising the surface which name is given by the variable \textit{sname} in the GTS workspace. Each entry in the list is another list with entries \textit{max}, \textit{min}, \textit{avg}, \textit{std} and \textit{n} with obvious interpretation.

## Functions for Definition of Surfaces

### 12.1.9 new

**1. Format:** \( u = \text{gts.surface.new}(s/,\text{prop}/) \)

**Arguments:**

1. \textit{s}, string, unique name of the surface to be created in the workspace. If the surface with that name already exists in the workspace, its elements are deleted and the surface with new properties put in its place.
2. \textit{prop}, list, properties that describe the surface. In its absence, and empty surface is created. The expected entries of the list depend on its most important entry \textit{desc}
   - \textit{desc} = “sphere”
     * \textit{level}, integer, level of detail of the sphere (default is 4).
   - \textit{desc} = “plane” or “rectangle”
     * \textit{bbox}_x, \textit{bbox}_y, and \textit{bbox}_z, of which two are pair of values, and one of them has to be single value. Creates rectangle in one of the planes \( x = \text{const}, y = \text{const} \) or \( z = \text{const} \), where the range is determined by the bounding box values.
   - \textit{desc} = “box”
     * \textit{bbox}_x, \textit{bbox}_y, and \textit{bbox}_z, all of are pairs of values. Creates box with respective bounding boxes.
     * \textit{remove\_face}, integer array of 0’s and 1’s for each of 12 triangles making the surface of the box. It explicitly includes or excludes each face. If omitted, all faces are included.
   - \textit{desc} = “mesh”
     * \textit{x}, \textit{y}, and \textit{z}, where \textit{x} is a real vector of length \textit{nx} containing the \textit{x-coordinate} of the mesh, \textit{y} is a real vector of length \textit{ny} containing the \textit{y-coordinate} of the mesh, and \textit{z} is a matrix of size \textit{nx-by-ny} where \( z[i; j] = z(x[i], y(j)) \).

**Note:** To exclude a rectangular area out of the surface - that is, to put a hole in the surface - choose the mesh appropriately, and set the values at the holes to \text{nan}() ;

In addition, all surfaces allow specifications of the point-wise transforms, which are all applied after the surface is constructed in \textit{x,y,z} coordinates.

- \textit{transl}, real 3-vector, translation of the surface after construction.
12.1. GTS

– center, real 3-vector, presumed center of surface, which is invariant under all transformations except translation.
– scale, real 3-vector, scaling of the surface in each of the tree directions.
– rot_x, rot_y, rot_z - real 3x3 matrix, presumably of rotations along each of the 3 axes.
– color, real 3-vector, single property of the entire surface, presumably its color.

II. Format: \( u = \text{gts.surface.new}(s\text{var}) \)

Arguments:

1. svar, variable containing GTS-compliant description of the surface in terms of a list with four entries:
   – v, 3-col matrix of vertices comprising the surface.
   – e, 2-col integer matrix of edges comprising the surface, where the row entries are the indices of its vertices in v.
   – f, 3-col integer matrix of the triangles comprising the faces of the surface, where the row entries are the indices of its edges in e.
   – c, 3-col matrix containing the color of each or of all faces. If omitted the default value \([1,1,1]\) is used.

Result: \( u \), an integer, 0 for success, 1 for failure.

Abstract: \( \text{gts.surface.new} \) creates new surface in the GTS workspace with the given properties. The list \( \ll v; e; f; c \gg \) can be created, e.g., by \( \text{svar=\text{gts.surface.export}(s)} \).

\[
\text{\Roman{12.1.10} \ from\_vertices}
\]

Format: \( \text{gts.surface.from\_vertices}(\text{sname}, v, f\text{v}/, c/) \)

Arguments:

1. sname, string, unique name of the surface to be created in the workspace. If the surface with that name already exists in the workspace, its elements are deleted and the surface with new properties put in its place.
2. v, 3-column real matrix of vertices comprising the surface.
3. f\text{v}, 3-column integer matrix of indices of vertices comprising surface. Notice \( f\text{v} \) is NOT a GTS compliant face.
   This is a helper function that makes it easier to construct a surface when only point information may be available.

Result: \( u \), an integer, 0 for success, 1 for failure.

Abstract: Creates a surface \( \text{sname} \) in GTS workspace from the vertices and ad hoc face data.
12.1.11  readm,writem

Format: \( u = \text{gts.surface.readm}(s, fn) \)
Format: \( u = \text{gts.surface.writem}(s, fn/, \text{opts}/) \)

Arguments:

1. \( s \), string, unique name of the surface to be created in the workspace. If the surface with that name already exists in the workspace, its elements are deleted and the surface with new properties put in its place.

2. \( fn \), string, file name containing GTS-compliant surface definition. If writing to file then from the file name the format is derived: oogl and gts.

Result: \( u \), an integer, 0 for success, 1 for failure.


12.1.12  export

Format: \( s = \text{gts.surface.export}(sname) \)

Arguments:

1. \( sname \), string, unique name of the surface to be created in the workspace.

Result: \( s = \langle v; e; f; c \rangle \), an \texttt{rlab} list containing GTS compliant surface.

Abstract: Exports GTS surface to \texttt{rlab}, where so defined surface can be re-imported to the GTS, e.g., in this example by using \texttt{gts.surface.new(newname, s)}. 
Figure 12.1: (Top row) Two examples of Gnuplot output of plotting, a rectangular surface with a rectangular hole (top left) and “cutter” surface (top right) provided with GTS source tree in directory test/boolean/surfaces. The mesh (black lines) is plotted as a 3-D scatter line plot, where user can specify line and point style in usual fashion, while the faces (gray surfaces) are plotted as Gnuplot objects 'polygons'.
(Bottom row) Two examples of use of geomview for rendering 3-D plots, shaken cube (bottom left); and horse4 (bottom right), also from test/boolean/surfaces.
Surface Manipulation Functions

12.1.13  foreach_vertex, foreach_edge, foreach_face

Format: $u = gts.surface.foreach_vertex(s, vertexfn, fnparams, /, /opts)$

Arguments:

1. $s$, string, name of the surface in the GTS workspace.
2. $vertexfn$, name of function that will be called for each vertex of the surface $s$. The function has to be defined as

   $vertexfn = function (vertex, fnparams) \{ \ldots \}$;

   where $vertex$ is a 3-vector, and $fnparams$ are the optional parameter passed to the function. If function returns a value, then the result of the last call of the function $vertexfn$ will be returned upon traversing all the vertices.
3. $fnparams$, parameter that is passed directly to the function $vertexfn$.
4. $opts$, list of options. Suported options are
   - $modify\_obj$, integer 0 or 1. If set to 1 then it expects that the $vertexfn$ returns a 3-vector which will then replace the function argument vertex in the surface (replaces the vertex or the points in the edge, or points in the triangle with the result of the function call on the vertex, or edge, or triangle.
   - $stack\_pts$, integer 0 or 1. If iterating over edges (triangles) choose if the function argument is going to be two points as row 6-vector (three points as row 9-vector), or two (three) points stacked on top of each other, that is a matrix with two (three) rows and three columns.

Result: $u$, depends whether $modify\_obj$ is set or not. If set, then returns 0 upon successful completion of iteration. Conversely, if $modify\_obj$ is unset then the result of the last function call is returned. For obvious reasons, one cannot combine the two, that is, the solver cannot modify vertices (edges, faces) and return value.

12.1.14  foreach_face_remove

Format: $gts.surface.foreach_face_remove(sname, decidefn, fnparams, opts)$

Arguments:

1. $sname$, string, name of the surface in the GTS workspace.
2. $decidefn$, name of the function that will be called on each face of the surface $s$. The function has to be defined as

   $decidefn = function (face, fnparams) \{ \ldots \}$;

   where $face$ can be a 9-vector containing the three vertices of the face, or a matrix 3-by-3 where each row is a vertex of the face. $fnparams$ are the optional parameter passed to the function. The function returns 0 if the face stays with the surface, or 1 if the face is to be removed from the surface.
3. _fnparams_, parameter that is passed directly to the function _vertexfn_.

4. _opts_, list of options. Suported options are

   - _stack_pts_, integer 0 or 1. If iterating over triangles choose if the function argument is going to be three points as row 9-vector, or three points stacked on top of each other, that is a matrix with three rows and three columns.

Abstract: The solver iterates over all faces of the named surface _sname_, and for each face it executes the _rlab_-function _decidefn_. If _decidefn_ returns 1 for particular face, this face is removed from the surface.

12.1.15  _add_face_

Format: \( i = \text{gts.surface.add\_face}(\text{sname}, \text{tri}/, \text{opts}/) \)

Arguments:

1. _sname_, string, name of the surface.

2. _tri_, 9-vector or 3x3 matrix, vertices of a face to be added to the surface.

3. _opts_, list of options, Currently supported is entry _color_, 3-vector of the color associated with the face.

Result: \( i \), integer 0 or 1, if surface existed and the new face has been added, or if surface did not exist, and was created comprising the single face.

Abstract: Adds face to the surface: a surface is created comprising single face from three provided vertices, This surface is then merged with the surface named in _sname_.

12.1.16  _remove_face_

Format: \( i = \text{gts.surface.remove\_face}(\text{sname}, \text{tri}) \)

Arguments:

1. _sname_, string, name of the surface.

2. _tri_, 9-vector or 3x3 matrix, vertices of a face to be added to the surface.

3. _opts_, list of options, Currently supported is entry _color_, 3-vector of the color associated with the face.

Result: \( i \), integer 0 or 1, if surface existed and the new face has been added, or if surface did not exist, and was created comprising the single face.

Abstract: Remove face from the surface by first adding the vertices provided in the array _tri_.

Note: This operation is 50% or better likely to succeed. Sometimes it just silently fails.
12.1.17 remove_surface

Format: gts.surface.remove_face(sname, rsname)

Arguments:

1. sname, string, name of the surface.
2. rsname, string, name of the surface.

Abstract: Remove second surface from the first surface.

Note: This operation is 50% or better likely to succeed. Sometimes it just silently fails.

Example:
Chapter 13

Rlab2 Rel.3 Internals

13.1 Sparse Matrices

rlab2 Rel. 2 is statically linked to the following sparse matrix packages:

- UMFPACK by Davis (2005),
- SuperLU by Demmel et al. (2003), and
- SPARSKIT by Saad (1994).

UMFPACK and SuperLU each provides a LU-factorization type solver for sparse problems, while SPARSKIT provide a number of iterative solvers and the necessary preconditioners. SPARSKIT also provides some general sparse read/write functions, which can be accessed through spwrite and spread commands.

UMFPACK and SuperLU are C-libraries, and so their sources are located in subdirectory /clibs in rlab source tree, while SPARSKIT is a FORTRAN library and such are located in flibs.

But first, legal stuff...

SuperLU

What is SuperLU

Copyright (c) 2003, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from U.S. Dept. of Energy)

SuperLU [Demmel et al., 2003] contains a set of subroutines to solve a sparse linear system $A \cdot x = b$. It uses Gaussian elimination with partial pivoting (GEPP). The columns of $A$ may be preordered before factorization; the preordering for sparsity is completely separate from the factorization.

Changes to SuperLU source code

There were two minor changes: the library uses Boehm’s garbage collector library for allocation/deallocation of memory, and the doublecomplex type in superlu is replaced by rlab_complex structure. For that reason rlab2 Rel. 2 comes with its own, stripped, distribution of superlu-3.0 which provides archive clibs/superlu/superlu.a for static linking.
CHAPTER 13. RLAB2 REL.3 INTERNALS

UMFPACK

What is UMFPACK?

Copyright (c) 2005, by Timothy A. Davis.

UMFPACK is a set of routines for solving unsymmetric sparse linear systems, \( A \cdot x = b \), using the unsymmetric multi-frontal method and direct sparse LU factorization.

The original source code is available from http://www.cise.ufl.edu/research/sparse/umfpack

SPARSKIT

What is SPARSKIT?

Copyright (c) 2005, Yousef Saad, University of Minnesota.

SPARSKIT is a collection of iterative solvers, preconditioners, different matrix/vector routines etc. http://www-users.cs.umn.edu/ saad/software/SPARSKIT/sparskit.html

Sparse solve, spsolve and related functions

13.1.1 sparams

Format: sparams.realsolv(“method”),
Format: sparams.iterator(iter, m),
Format: sparams.precond(ipre, stat),
Format: sparams.tol(abserr, relerr, convcrit, maxit),
Format: sparams.dpt(dpt, preo),
Format: sparams.compsolv(“method”).

Abstract: sparams is a list of function which allows an user to choose different parameters for the RLaB sparse solver.

- sparams.realsolv(“package”) sets a sparse real solver which will be used when function solve(a, b) is called with a real sparse matrix a (, and a dense matrix b, the right hand side of the linear problem). The choices for “package” is
  - “umfpack”, “UMFPACK”, “umf”, for UMFPACK solver (default);
  - “SuperLU”, “superlu”, “slu”, for SuperLU solver; and
  - “SPARSKIT”, “sparskit”, “spk”, for SPARSKIT iterative solvers.

If user chooses an SPARSKIT than she also has to provide additional parameters: preconditioner, using command sparams.precond(ipre), where

- ipre = 1, for ILUT;
- ipre = 2, for ILUTP;
- ipre = 3, for ILUD (default);
- ipre = 4, for ILUDP;
- ipre = 5, for ILUK;

where all rely on some form of incomplete LU-factorization or the other. The user also can choose iterative solver using command sparams.realsolv(iter), where
– iter = 1, for CG;
– iter = 2, for CGNR;
– iter = 3, for BCG;
– iter = 4, for BCGSTAB;
– iter = 5, for TFQMR;
– iter = 6, for GMRES (default);
– iter = 7, for FGMRES;
– iter = 8, for DQGMRES;
– iter = 9, for DBCG.

Finally, absolute and relative error entering the stopping criterion have to be given. This is done with \texttt{sparams.tol(abserr,relerr)}. Default values are abserr = $1e^{-6}$ and relerr = 0.

\begin{itemize}
\item \textbf{Note:} Each of the functions $x = \texttt{solve}(a,b)$ or $x = a\backslash b$ refer to, are optimized for memory and speed. This means that internally a minimum number of steps and copying is involved to get the final result. This required substantial rewriting of the sparse interface compared to RLaB2 Rel.1.
\item \textbf{Note:} Generally speaking, umfpack is the fastest (speed, memory consumption) for linear problems containing unstructured sparse matrices. SuperLU is slower, and iterative solvers typically can’t make it (issues with preconditioner, hard to decide on memory consumption for incomplete LU factorization). For banded sparse matrices iterative solvers are as quick as superlu (which is somewhat faster than umfpack) but consume less memory. Choose your method carefully, and if you don’t have a clue what I am talking about leave defaults (umfpack).
\item \textbf{Note:} My opinions are subject to change without any notice.
\item \textbf{Note:} If you forget how to use above commands, just type the command name without any parameters in parenthesis. The error messages should give you the clue how to proceed.
\end{itemize}

### 13.1.2 \texttt{spsolve}

\begin{itemize}
\item \textbf{I Format:} $x = \texttt{spsolve}(a,b)$
\item \textbf{Arguments:}
\begin{itemize}
\item 1 $a$, sparse square matrix;
\item 2 $b$, dense vector.
\end{itemize}
\item \textbf{Result:} $x$, dense vector.
\end{itemize}

\textbf{Abstract:} Solves sparse linear problem $a \cdot x = b$ where $a$ is sparse square matrix but where $b$ is a dense vector.

\begin{itemize}
\item \textbf{II Format:} $\texttt{spsolve}(a); x = \texttt{spsolve}(b)$
\item 1 $a$, on first call to the function, sparse square matrix;
\item 1 $b$, on subsequent calls to the function, dense vector.
\end{itemize}
Result: $x$, dense vector.

Abstract: Solves a series of sparse linear problems $a \cdot x = b_j$ where $a$ is sparse square matrix but where $b_j$ is a number of different dense vectors.

Abstract: In many situations it will happen that a real linear problem $a x = b$ will have to be solved multiple times, where the matrix $a$ will be the same, while the right-hand-side $b$ will change. In those situations it is costly to recalculate LU factorization of $a$ each and every time. Function $\text{spsolve}$ exists exactly with this situation in mind. When called with a sparse matrix as an argument, e.g., $\text{sparse}(a)$ or $\text{sparse}(a,b)$, it first calculates the LU decomposition of $a$ and stores the result internally, and then applies it to any right-hand side either submitted as $\text{sparse}(b)$ or as $\text{sparse}(a,b)$.

Note: “SuperLU” in $\text{spsolve}$ mode is much faster than “UMFPACK”, compared to their pretty even $\text{solve}$ performance. For multiple right-hand-sides “UMFPACK” is overall $\sim 10$ or so times faster than $\text{spsolve}$ than $\text{solve}$. “SuperLU,” on the other hand, is $\sim 20$ and more times faster as $\text{spsolve}$ than $\text{solve}$. The improvement comes from “SuperLU” having the full LU decomposition stored internally, and not referring to the original matrix during backsubstitution computations. “UMFPACK” stores only partial decomposition information in its internal symbolic and numeric structures, and refers to the original matrix (which thus also has to be stored) while doing the backsubstitution.

Note: Do not use $\text{factor}$, $\text{spfactor}$ or $\text{backsub}$ for the tasks for which $\text{spsolve}$ is designed. This will not work because “SuperLU” and “UMFPACK” use compressed column format while rlab uses compressed row format for internal storage of sparse matrices. Doing factorization requires rlab matrix to be transposed first before calling “SuperLU” or “UMFPACK”, then the result of factorization (as it is actually a factorization of transposed matrix) needs to be converted in the right format, which is again time consuming. Finally, when calling $\text{backsub}$, the LU matrices have to be converted back to the original column-compressed format to be used for internal backsubstitution routines. Conclusion: use $\text{factor}/\text{spfactor}$ to inspect left-hand-side matrix, otherwise forget about it.

13.2 Changes in Linear Algebra Routines

13.2.1 eig

I Format: $y = \text{eig}(a/\,\text{flag}/)$,

Arguments:

1 $a$, dense square matrix;
2 $\text{flag}$, string, “S” or “G” designating whether $a$ is symmetric or general matrix.

II Format: $y = \text{eig}(a,b/\,\text{flag}/)$,

Arguments:

1 $a$, dense square matrix;
2 $b$, dense square matrix;
3 $\text{flag}$, string, “S”, “G” or “SP” designating whether $a$ is symmetric or general, and $a$ is symmetric and $b$ is positive-definite matrix. matrix.

Result: $y$, list with entries $\text{val}$ and $\text{vec}$, for eigen-values and eigen-vectors, respectively.
13.2. CHANGES IN LINEAR ALGEBRA ROUTINES

13.2.2 eigs

Format: \( y = eigs(a/, b/, k/, s/) \)

Arguments:

1. \( a \), square matrix, dense or sparse;
2. \( b \), square matrix, dense or sparse;
3,4 \( k \), integer, number of eigenvalues and eigenvectors of desired properties;
3,4 \( s \), positive scalar \((\sigma)\), or string \((\text{which})\) or list \(<< \sigma; \text{which} >>\).

Result: \( y \), list with entries \( \text{val} \) and \( \text{vec} \), containing the first \( k \) eigen-values and eigen-vectors, respectively

Abstract: \( eigs \) implements ARPACK\([\text{Lehoucq et al.} 1997]\) for finding selected eigenvalues and eigenvectors of a matrix \( a \) (standard eigenvalue problem) or two matrices \( a \) and \( b \) (general eigenvalue problem). For a single matrix, there are two methods that are implemented: \textit{regular}, for \( \sigma = 0 \), and \textit{shift-invert}, for \( \sigma \neq 0 \). The string \textit{which} determines which eigenvalues are computed (regarding the value of \( \sigma \)), and can be

- “LM”, “SM”, for largest, smallest in magnitude;
- “LA”, “SA”, for largest, smallest algabreic, (real symmetric \( a \), real \( \sigma \));
- “LR”, “SR”, for largest, smallest real part (complex \( a \), or complex \( \sigma \));
- “LI”, “SI”, for largest, smallest imaginary part (complex \( a \), or complex \( \sigma \)),
- “BE”, for computing eigenvalues from both ends of the spectrum (real symmetric \( a \), real \( \sigma \)).

Details: when using \textit{shift-invert} mode for computation of eigenvalues, the code finds and stores internally LU decomposition of the matrix \( a - \sigma \cdot I \), either using LAPACK routines (for dense \( a \), or \textit{spsolve} (for real sparse \( a \), and real \( \sigma \)). For complex sparse matrix, instead of LU decomposition, we use UMFPACK sparse solver (speed and memory consumption for unstructured sparse matrices), to solve a linear system \((a - \sigma \cdot I)x' = x\) for \( x' \). This is slower than \textit{spsolve}, but at this moment \textit{spsolve} does not work for complex sparse matrices.

Note: a generalized eigenvalue problem where \( a \) is real-symmetric and \( b \) is positive definite is not yet implemented (ARPACK provides 2 additional routines for invertible \( b \)'s).

Note: (from the authors of ARPACK) If \( b \) is easily invertible, convert the general eigenvalue problem to a standard one.

Note: (from me, the writer of the code and this manual) Do not upgrade the ARPACK from the author’s sources if such become available in the future. I stripped the original code from the report generating routines and variables as these made the whole code not work properly in the dynamically allocated memory environment of \texttt{rlab}. I informed the authors about that. Here, ARPACK works with the latest LAPACK (for original, version 2 was recommended) and BLAS.
Chapter 14

New Data Types

14.1 Integers

As of rlab version 2.4.2.0 the parser recognizes integer types in decimal (number ends with “L” or “l”, all digits in range 0-9), hexadecimal (number starts with “0x” or “0X,” all digits in range 0-9 with letters “a,” “b,” “c,” “d,” and “f”), and binary format (number ends with “B” or “b,” all digits in range 0-1):

```plaintext
1 >> x=0xaade
2 ans =
3     43742
4 >> class(x)
5 num
6 >> type(x)
7 int
8 >> y=0101001B
9 ans =
10     41
11 >> type(y)
12 int
13 >> z=4724839L
14 ans =
15     4724839
16 >> type(z)
17 int
```

This complements int function present since rlab version 2.2.8.

The integers are equivalent to c-type int, that is, signed 4-bytes integers (int32).

Most of the functions work with integers seamlessly, while the functions that do not, complain about it. For multiargument operators the rule about the result is: If both arguments are integers, and the result of the operation is integer (e.g., addition or subtraction), then the result is integer, as well. Otherwise the result is non-integer: E.g., addition of an integer and a double gives a double.

Some operations act differently when applied to integers than when applied to doubles: logical operators and (&&), or (||), not (!) act bit-wise on integer entries:
The integer matrices are properly read and saved in binary files, and can be properly saved when using generic ASCII write format. rlab can now read in m*lab's integer matrices properly from binary files. For now integer random number generator has a type mismatch. By the GSL definition its output are unsigned integers in range $0 \ldots 2^{32} - 1$, while they are assigned to the signed integers in range $-2^{31} + 1 \ldots 2^{31} - 1$. So bitwise, they both store the same information, it is just a question of interpretation.

### 14.2 Ordered Lists - Experimental Feature

As of rlab version 2.5.0 there is a new data type, called in rlab parlance Dense Entity Matrix (or MDE). The data type is very similar to the existing (unordered) lists (somewhat similar to python dictionaries) with that difference that instead of key, the element is accessed through its one or two indices. Internally, this means that the data access is faster: offset of few pointers for MDE's is easier to calculate, then traversing a binary tree to the right node.

This matrix has as entries other data types (in rlab parlance, these are called entities). rlab's MDEs are equivalent to cells, or cell-arrays in m*lab or octave; lists in python, or struct's in c-language.

Matrices of entities can be stacked, and referenced just like regular matrices, with that exception that once they are created certain way they become immutable: E.g., if $c$ is an MDE, where its entry $c[1]$ is a numerical matrix, then later $c[1]$ cannot be changed to, say, string matrix.

For now, the MDEs are experimental feature in that one should use them carefully, because segmentation faults may occur.

#### 14.2.1 cell

**Format:** $c = \text{cell}(nr, nc)$,

**Arguments:**

1. $nr, nc$, non-negative integers, number of rows and columns in an MDE to create.

**Result:** $c$, entity matrix of size nr-by-nc, containing undefined entities.

#### 14.2.2 vcell

**Format:** $c = \text{vcell}(e1, e2, e3, \cdots)$,
14.2. ORDERED LISTS - EXPERIMENTAL FEATURE

Arguments:

1. \( e_1, \ldots, rlab \) entities.

Result: \( c \), column-vector of entities, containing content of the entities \( e_1, \ldots \).

14.2.3  hcell

Format: \( c = \text{hcell}(e_1, e_2, e_3, \cdots) \),

Arguments:

1. \( e_1, \ldots, rlab \) entities.

Result: \( c \), row-vector of entities, containing content of the entities \( e_1, \ldots \).

Example: In the following example we create an entity matrix of size 1-by-4, and assign different dense matrices as its entries:

```plaintext
>> c = cell(1,4)
ans =
Matrix_Dense_Entity 1-by-4:
[1;1] -> Undefined
[1;2] -> Undefined
[1;3] -> Undefined
[1;4] -> Undefined
```

```plaintext
>> c[1] = rand(1,6); c[2] = rand(2,1); c[3] = randchar(3,3,18)
ans =
Matrix_Dense_Entity 1-by-4:
[1;1] -> Matrix_Dense_Real
 0.920833526 0.247582271 0.673870914 0.198899527 0.995438771 0.0177593464
[1;2] -> Matrix_Dense_Real
 0.689719571
[1;3] -> Matrix_Dense_String
lJTG8gSgFqCqLgPcqGygxb Ybu3KrTEJiFCyEGVkrH 6rqoqzTi2NLwG0E5YP
pXpnGTP0RA90kxxPa wM21F2rUEPVo1TsNk eR0EfVHZVbLPd7b
XJHoq8Nrd07SiLs1U bqWxxHOTQSKeCn3D 1C9lfcZJxB6chOBbYy
[1;4] -> Undefined
```

Example: We continue previous example, where to the \( c[4] \) we attach another entity matrix. However, because of the limitations of the parser, this cannot be done directly, but indirectly:

```plaintext
>> d=cell(1,3); d[1] = rand(1,2); d[2] = rand(2,1)
ans =
Matrix_Dense_Entity 1-by-3:
[1;1] -> Matrix_Dense_Real
 0.869975524 0.682346592
[1;2] -> Matrix_Dense_Real
```

0.92481232
0.678735826
[1;3] -> Undefined
>> c[4] = d
ans =
Matrix_Dense_Entity 1-by-4:
[1;1] -> Matrix_Dense_Real
0.920833526 0.247582271 0.673870914 0.198899527 0.995438771 0.0177593464
[1;2] -> Matrix_Dense_Real
0.689719571
0.71271324
[1;3] -> Matrix_Dense_String
1JTG8GsLgPcqJgYgxb Ybu3KrTEJiFCYEgvkH 6rqqzTi2NLwG0E5YP
pXpGTPORAO90kxxPa wM21Fd2rUPEVP1TsNK eROEfVHZyVbLpGd7b
XHJog8Nrd075l5s1U bqWxxH0TQ9S4kpecn3D 1C9lfecZxB5chO8bYy
[1;4] -> Matrix_Dense_Entity
Matrix_Dense_Entity 1-by-3:
[1;1] -> Matrix_Dense_Real
0.869975524 0.682346592
[1;2] -> Matrix_Dense_Real
0.92481232
0.678735826
[1;3] -> Undefined

Example: We continue by retrieving a few entries from the structure:

>> c[1]
anst =
0.920833526 0.247582271 0.673870914 0.198899527 0.995438771 0.0177593464
>> c[4][2]
anst =
0.92481232
0.678735826

Example: We finish these examples, by modifying entries in the dense matrices that are part of the structure:

>> c[1]
anst =
0.920833526 0.247582271 0.673870914 0.198899527 0.995438771 0.0177593464
>> c[1][1]
anst =
0.920833526
>> c[1][1] = 1
ans =
1 0.247582271 0.673870914 0.198899527 0.995438771 0.0177593464
14.2. ORDERED LISTS - EXPERIMENTAL FEATURE

>> c[1] = randchar(1,1,23)
ans =
Matrix_Dense_Entity 1-by-4:
[1;1] -> Matrix_Dense_String
qmy1137qsbwrRYkqdAw1HH
[1;2] -> Matrix_Dense_Real
0.689719571
0.71271324
[1;3] -> Matrix_Dense_String
1JTg8GsLgPcqJgYgxb Ybu3KrTEj1FCYEGvkH 6rqoqzTii2NLwGOE5YP
pxpGTP0RAD90kxxPa Wm21Fd2rUPEVP1TsNK eR0EFvHZYVbLpGd7b
XHJog8NpdG7SiLs1U bqwxxH0TQ9SKpecn3D IC91fcZJxB6ch0BbYy
[1;4] -> Matrix_Dense_Entity
Matrix_Dense_Entity 1-by-3:
[1;1] -> Matrix_Dense_Real
0.869975524 0.682346592
[1;2] -> Matrix_Dense_Real
0.92481232
0.678735826
[1;3] -> Undefined
>>

14.2.4 iscell

Format: \( i = \text{iscell}(c) \),

Arguments:

1. \( c \), any entity.

Result: \( c \), integer in range 0 to maximum depth of the entity matrix. Under the depth we assume an entity matrix (depth=1), which as an entry has another entity matrix (depth=2), which has as an entry another entity matrix (depth=3), and so forth, where as the depth, the maximal depth is reported.

Example: For the previous example,

>> iscell(c)
2
>>

This is because entity matrix \( c \) has as one entry another entity matrix, but that entity matrix has as entries all non-entity matrices, thus the depth of \( c \) is 2.

Stacking Entity Matrices

What changes compared to regular matrices is the following. A matrix has to be initially declared entity matrix, even if of zero-size,
Now, we can add entries to \( c \) using regular stacking operations, e.g.,

\[
\begin{align*}
\text{>> } a &= \text{rand}(3,3) \\
\text{ans} &= \\
&\begin{bmatrix}
0.0887320207 & 0.293112792 & 0.857965352 \\
0.238220698 & 0.952758355 & 0.995495549 \\
0.693303563 & 0.463039492 & 0.580193552
\end{bmatrix}
\text{>> } c &= [c, a] \\
\text{ans} &= \\
\begin{bmatrix}
\text{Matrix}_\text{Dense}_\text{Entity} & \text{1-by-1:} \\
0.0887320207 & 0.293112792 & 0.857965352 \\
0.238220698 & 0.952758355 & 0.995495549 \\
0.693303563 & 0.463039492 & 0.580193552
\end{bmatrix}
\text{>> } a &= \text{randchar}(2,4,6) \\
\text{ans} &= \\
\begin{bmatrix}
\text{Cu9HU8} & \text{hMty3i} & \text{fLPUCX} & \text{IKDU34} \\
\text{McgLYM} & \text{Sllkbh} & \text{oAt83Q} & \text{GSyWOW}
\end{bmatrix}
\text{>> } c &= [c, a] \\
\text{ans} &= \\
\begin{bmatrix}
\text{Matrix}_\text{Dense}_\text{Entity} & \text{1-by-2:} \\
0.0887320207 & 0.293112792 & 0.857965352 \\
0.238220698 & 0.952758355 & 0.995495549 \\
0.693303563 & 0.463039492 & 0.580193552 \\
\text{Cu9HU8} & \text{hMty3i} & \text{fLPUCX} & \text{IKDU34} \\
\text{McgLYM} & \text{Sllkbh} & \text{oAt83Q} & \text{GSyWOW}
\end{bmatrix}
\text{>> }
\end{align*}
\]

...
14.3 Protected Lists

As of rlab version 2.2.8 a list can be protected/released. This change was introduced to allow save command not to try to save the lists of functions and built-in numerical constants such as lists const (containing mathematical constants) and mks (containing unit conversion factors).

The problem in older version of rlab can be described as following. Create a function list of user defined functions. Save the workspace using save command. Now read it in using load. The list of functions should be gone. This is because the rlab saves the name of the list to the file (but cannot save its content, as the list contains only functions). When loading the workspace file, the list content is overwritten by the empty list from file.

The way it works now is the following. If the list is protected it cannot be saved using save and it cannot be overwritten, using assignment statement, or destroyed, using clear/clearall.

14.3.1 protect

Format: protect(L),

Arguments: list *L,

Abstract: This protect the list-variable L from being destroyed, cleared, overwritten or saved to a file using binary-write.

14.3.2 release

Format: release(L),

Arguments: list *L,

Abstract: This releases the protection of the list-variable L so it can be destroyed, cleared, overwritten or saved to a file using binary-write.

14.3.3 isprot

Format: x = isprot(L),

Arguments: list *L,

Abstract: This function checks whether the list L is protected (x = 1) or not (x = 0). It is used in the functions save, who and whos.

Note: Use protect if you are creating lists of functions. rlab has a few, do not unprotect them.

Note: Having a list protected prevents an user from accidentally destroying the list as a variable. Its entries still can be modified, e.g.,

```plaintext
>> const
   e    euler    ln10    ln2    lnpi
   log10e log2e    pi    pihalf    piquarter
   rpi    sqrt2    sqrt2r    sqrt3     sqrtpi
>> const.pi
   3.14159265
```
>> const = 1
ERROR: /usr/local/bin/rlab-2.2.8: cannot assign to a protected variable!
>> const.pi = 1
   e        euler    ln10    ln2    lnpi
  log10e  log2e      pi     pihalf    piquarter
   rpi      sqrt2     sqrt2r    sqrt3    sqrtpi
  tworpi
>> const.pi
   1
14.4 Subroutines

Rlab does not support subroutines directly, because it allows only functions to be defined. What can be done in a function is to change the values of its input arguments.

14.4.1 assign

Format: \texttt{assign(x,val)},

Arguments:

1. \texttt{x}, variable, presumably one of the input arguments of a function.

2. \texttt{val}, variable or value which content is to be assigned to the variable \texttt{x}.

Abstract: The function assigns value to a variable (that may be an input argument of a function). This value is then available to the same variable upon return from the function.

Example: This is how subroutine might be implemented:

```rhl
subroutine = function (a,b,c)
{
    printf("SUBR : x = \%g,\%s\n", a, entinfo(a).addr);
    rval = a + b + c;
    assign(a,rval);
    assign(b,2*rval);
    assign(c,3*rval);
    return 0;
}

x = rand();
y = rand();
z = rand();
printf("before: x = \%g,\%s\n", x, entinfo(x).addr);
subroutine(x,y,z);
printf("after : x = \%g\n\n", x);
```

Which produces the following output

```bash
>> rfile test/general/eg_assign.r
before: x = 0.186492,0x13abde0
SUBR : x = 0.186492,0x13abde0
after : x = 0.590712
>>
```
14.5 System Constants

Since rlab 2.2.8 the following lists of constants are implemented, mks and const.

14.5.1 mks - a list of physical constants and physical unit conversion factors

Abstract: The list mks contains the following physical constants, conveniently grouped.

Fundamental physical constants

- \( c \) speed of light in vacuum;
- \( \mu_0 \) permeability of free space, \( \mu_0 \);
- \( \varepsilon_0 \) permittivity of free space, \( \varepsilon_0 \);
- \( N_a \) Avogadro’s number, \( N_a \);
- \( k_B \) Boltzmann constant, \( k_B \);
- \( R_0 \) the molar gas constant, \( R_0 \);
- \( V_0 \) the standard gas volume, \( V_0 \);
- \( \text{gauss} \) the magnetic field of 1 Gauss;
- \( \text{mikron} \) the length of 1 micron;
- \( \text{ha} \) the area of 1 hectare;
- \( \text{mph} \) the speed of 1 mile per hour;
- \( \text{kmh} \) the speed of 1 kilometer per hour.

Astronomical units and constants

- \( \text{au} \) astronomical unit;
- \( G \) the gravitational constant;
- \( \text{ly} \) the distance of 1 light year;
- \( \text{parsec} \) the distance of 1 parsec;
- \( g \) the standard gravitational acceleration on Earth;
- \( mSun \) the mass of the Sun.
14.5. SYSTEM CONSTANTS

Atomic and Nuclear Physics units and constants

19 \( e \), the charge of the electron;
20 \( eV \), the energy of 1 electron-volt;
21 \( amu \), the atomic mass unit;
22 \( mmu \), the mass of the muon, \( m_\mu \);
23 \( mp \), the mass of the proton, \( m_p \);
24 \( mn \), the mass of the neutron, \( m_n \);
25 \( \alpha \), the electromagnetic fine structure constant, \( \alpha \);
26 \( Ry \), the Rydberg constant;
27 \( a_0 \), the Bohr radius, \( a_0 \);
28 \( a \), the length of 1 angstrom;
29 \( barn \), the area of 1 barn;
30 \( \mu B \), the Bohr magneton, \( \mu_B \);
31 \( \mu N \), the nuclear magneton, \( \mu_N \);
32 \( \mu e \), the electron magnetic moment, \( \mu_e \);
33 \( \mu P \), the magnetic moment of proton, \( \mu_p \).

Time

34 \( min \), the number of seconds in 1 minute;
35 \( hour \), the number of seconds in 1 hour;
36 \( day \), the number of seconds in 1 day;
37 \( week \), the number of seconds in 1 week.

Imperial units

38 \( inch \), the length of 1 inch;
39 \( foot \), the length of 1 foot;
40 \( yard \), the length of 1 yard;
41 \( mile \), the length of 1 mile;
42 \( mil \), the length of 1 mil (\( = 1/1000^{th} \) of one inch).
Nautical units
43  nmi, the length of 1 nautical mile;
44  fathom, the length of 1 fathom;
45  knot, the speed of 1 knot.

Volume
46  acre, the area of 1 acre;
47  L, the volume of 1 liter of H₂O;
48  gal, the volume of 1 U.S. gallon;
49  cgal, the volume of 1 Canadian gallon;
50  ukgal, the volume of 1 U.K. gallon;
51  cup, the volume of 1 cup.

Mass and weight
52  lb, the mass of 1 pound;
53  oz, the mass of 1 ounce;
54  ton, the mass of 1 metric ton;
55  ukton, the mass of 1 U.K. ton;
56  toz, the mass of 1 troy ounce;
57  ct, the mass of 1 carat;
58  gf, the weight of 1 gram;
59  pf, the weight of 1 pound;
60  kpf, the weight of 1000 pounds;
61  pal, the weight of 1 poundal.

Thermal energy and power
62  cal, the energy of 1 calorie;
63  btu, the energy of 1 British Thermal Unit;
64  therm, the energy of 1 therm;
65  hp, the power of 1 horse(power).
14.5. SYSTEM CONSTANTS

Pressure
66 \textit{bar}, the pressure of 1 bar;
67 \textit{atm}, the pressure of 1 atmosphere;
68 \textit{torr}, the pressure of 1 torr;
69 \textit{mHg}, the pressure of 1 meter of Hg;
70 \textit{inHg}, the pressure of 1 inch of Hg;
71 \textit{inH2O}, the pressure of 1 inch of H2O;
72 \textit{psi}, the pressure of 1 pound per square inch.

Viscosity
73 \textit{poise}, the dynamic viscosity of 1 poise;
74 \textit{stokes}, the kinematic viscosity of 1 stokes.

Light and illumination
75 \textit{stilb}, the luminance of 1 stilb;
76 \textit{lumen}, the luminous flux of 1 lumen;
77 \textit{lux}, the illuminance of 1 lux;
78 \textit{phot}, the illuminance of 1 phot;
79 \textit{ftcan}, the illuminance of 1 foot candle;
80 \textit{lam}, the luminance of 1 lambert;
81 \textit{ftlam}, luminance of 1 footlambert.

Radioactivity
82 \textit{curie}, the activity of 1 curie;
83 \textit{roe}, the exposure of 1 roentgen;
84 \textit{rad}, the absorbed dose of 1 rad.

Force and energy
85 \textit{dyne}, the force of 1 dyne;
86 \textit{erg}, the energy of 1 erg.
14.5.2 const - a list of mathematical constants

1. $e$, the base of natural logarithm;
2. $\log_2 e$, the base-2 logarithm of $e$;
3. $\log_{10} e$, the base-10 logarithm of $e$;
4. $\sqrt{2}$, the square root of 2;
5. $\frac{1}{\sqrt{2}}$, the reciprocal of the square root of 2;
6. $\sqrt{3}$, the square root of 3;
7. $\pi$, one and only;
8. $\frac{\pi}{2}$, one half of one and only;
9. $\frac{\pi}{4}$, one quarter of one and only;
10. $\frac{1}{\pi}$, the reciprocal of one and only;
11. $\sqrt{\pi}$, the square root of one and only;
12. $2\frac{1}{\pi}$, twice the reciprocal of one and only;
13. $\ln 10$, the natural logarithm of 10;
14. $\ln 2$, the natural logarithm of 2;
15. $\ln \pi$, the natural logarithm of one and only;
16. $\pi$, his constant.
Part IV

Miscellaneous Topics
Chapter 15

Embedded Interpreters

15.1 ngspice - Electronic Circuit Simulator

The function list 

spice

provides rlab with access to ngspice [NGspice team 2013]. The communication with ngspice can be done through

1. File exchange:

rlab creates circuit file, then starts a ngspice process that reads the circuit file, and (presumably) following successful computation, writes the results to a “raw” file. rlab then reads the content of the “raw” file. This requires only ngspice executable to be installed on the linux system.

2. Shared object library:

This requires ngspice to be compiled as a shared object library on the host system. While this is not difficult to do, there are some steps that user has to perform in order for the build to be successful. Once that is done, rlab can be built to support it: It requires rebuilding rlab using the --with-ngspice flag, that is,

./configure --with-ngspice [...other flags follow..]

15.1.1 exec

Format: \n
\ny = spice.exec([scriptckt], [fn_ckt], [fn_log], [fn_err])\n
Format: \n
\ny = spice.fork([scriptckt], [fn_ckt], [fn_log], [fn_err])\n
Arguments:

1. scriptckt, string vector, a sequence of ngspice-readable instructions. If omitted then the file fn_ckt that provides the circuit must be given.

2. fn_ckt, string, name of the file that ngspice will execute. If scriptckt is given, then this script will be saved to fn_ckt, otherwise it is assumed that fn_ckt exists and contains the circuit to be analyzed by ngspice.

3. fn_log, string, name of the file to which ngspice will write the log of computation, e.g., progress, warnings and such.
4. \texttt{fn.err}, string, name of the file to which ngspice will write any errors during the execution of the script.

The difference between \texttt{.fork} and \texttt{.exec} is that in the former case \texttt{rlab} will start the calculation and then fork it to background, while in the latter it will stick around until the computation is completed, and then return to the user.

\subsection*{15.1.2 raw}

\textbf{Format:} \texttt{s = spice.raw(fn\_raw)}

\textbf{Arguments:}

1. \texttt{fn\_raw}, string, name of the file that contains the output of \texttt{ngspice} analysis. If omitted, then value “rawspice.raw” is used.

\textbf{Result:} \texttt{s}, list with entries \texttt{≪“variables”; “points”; “data” ≫}, where \texttt{variables} gives the number of variables in the output, \texttt{points} the length of the vectors, while \texttt{data} is a list, which entries are named \texttt{v\_nn}, and so forth, for voltages of all circuit nodes \texttt{nn}, and \texttt{i\_xx} for currents of all voltage sources \texttt{xx}. Additionally, there is a column “time” (from transient analysis) or “v\_sweep” (from DC analysis), and so forth.

This function is loosely based on octave function \texttt{spice\_readfile} by Werner Hoch, 2005.

\textbf{Note:} One recognizes that these three functions in the list allow one to start a number of ngspice processes which will write their results to files.

For large number of shorter computations writing and reading files may become time consuming, so then using ngspice as a shared library becomes an option.

\subsection*{15.1.3 init}

\textbf{Format:} \texttt{spice.init(/fn\_log/)}

\textbf{Arguments:}

1. \texttt{fn\_log}, string, name of the file that contains the log of \texttt{ngspice} analysis.

\texttt{spice.init()} opens a ngspice session.

\subsection*{15.1.4 cmd}

\textbf{Format:} \texttt{spice.cmd(scriptcmds)}

1. \texttt{scriptcmds}, string vector, a sequence of commands, that one would have entered on command line directly in ngspice.

This is a direct call to \texttt{ngSpice\_Command\_handle} function in ngspice library.

\texttt{spice.cmd()} sends commands to an open ngspice session.

\subsection*{15.1.5 circuit}

\textbf{Format:} \texttt{spice.circuit(scriptckt)}

1. \texttt{scriptckt}, string vector, a content of circuit file for ngspice entered here line by line.

This is a direct call to \texttt{ngSpice\_Circ\_handle} function in ngspice library. You probably don’t need to use it.
15.1.6 runckt

**Format:** `spice.runckt(scriptckt)`

1. `scriptckt`, string vector, a content of circuit file for ngspice entered here line by line.
   It is different from the function `spice.circuit` in that it first clears the ngspice memory (issues “destroy all”), and upon loading the script issues a command to do the calculation “bg_run”.

15.1.7 getvals

**Format:** `s = spice.getvals()`

**Result:** `s`, list with entries `< “variables”; “points”; “data” >`, same as returned by the command `spice.raw()`.

15.1.8 isrunning

**Format:** `i = spice.isrunning()`

**Result:** `i`, integer, 0 if ngspice is not running (calculation completed), and 1 if it is running (still calculating).

15.1.9 kill

**Format:** `spice.kill()`

**Abstract:** `spice.kill()` closes an open ngspice session.
Example: In the following circuit we find the charge time of the capacitor $C_2$ as a function of its capacity for $C_2$ in the range 0.1-200 pF, with fixed $V_1 = 5$ V, $R_1 = 1$ kΩ, $C_1 = 125$ pF and $R_2 = 100$ Ω.

![Electric circuit used in discussion of transient analysis. We look at the charging of $C_2$ after the switch SW is activated.](image)

We use rlab to create a script with desired values of $C_2$, and collect results, namely, voltage across $C_2$ (between nodes 5 and 0) as a function of time, $v_5 = v_5(t)$. This is shown in Fig. (15.2), top panel. We fit $v_5 = v_5(t)$ to

$$v_5 \equiv V_{C_2}(t) = V_0 \cdot (1 - \exp\left(\frac{t - t_0}{\tau}\right)),$$

(15.1)

and find the fit parameters \{\(V_0, t_0, \tau\}%2C. Finally, in Fig. (15.2), bottom panel, we plot $\tau$ and $V_0$ as the functions of $C_2$. 
// rlab and ngspice

// rising edge formula
vfunc = function(t,p)
{
  // p:
  // p[1] -> barV
  // p[2] -> t0
  // model
  // V(t) = barV * (1 - exp(-(t-t0)/tau))
  //
  res = p[1]*(1 - exp(-(t-p[2])./p[3]));
  return res;
};

// constants
ns = 1e-9;
pF = 1e-12;
mohm = 1e6;

// circuit parameters - do not interfere with switching circuit
// unless, of course, you know what you are doing
// all units are SI
V1 = 5;
C1 = 125 * pF;
R1 = 100 * mohm;
R2 = 1200;
range_C2 = [0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200];
y = <<>>;
lgd_y = blank(0,0);
coef_fit = [];

for (i in 1:length(range_C2))
{
  // label to avoid lexicographic ordering
  li = text(i,"%02.0f");
  C2 = range_C2[i] * pF;
  // create appropriate ngspice script
spicecrkt = [ ...
    "* rc circuit: 2 rc circuits connected by a switch"; ...
    "v1 1 0 dc " + text(V1); ...
    "r1 1 2 " + text(R1); ...
    "c1 2 0 " + text(C1); ...
    "r2 4 5 " + text(R2); ...
    "c2 5 0 " + text(C2); ...
    "vc 3 0 0 pulse(0 1 0 ln 1n 5u 1s)"; ...
    "s1 2 4 3 0 switch"; ...
    ".model switch sw(vt=0.09 ron=0.01)"; ...
    ".ic v(2)=" + text(V1) + " v(4)=0 v(5)=0"; ...
    ".control"; ...
    ".tran 0.05ns 200ns"; ...
    "write"; ...
    ".endc"; ...
    "end" ...
];

// call ngspice and wait until it is finished, then load the data
spice.exec(spicecrkt);
s = spice.raw();

lgd_y = [lgd_y, ...
    text(R2, "R_2 = %g {/Symbol W}, ") + text(range_C2[i], "C_2 = %4.1f pf") ...
];
y.[ii] = [s.data.time, s.data.v_5] ./ [ns, 1];

// fit this to the rising edge formula
// find vmax
_vmax = lastr(y.[ii][2]);
_dt = max(diff(y.[ii][:][1]));
_ta = min(findroot(y.[ii], 0.1 * _vmax));
_tb = min(findroot(y.[ii], 0.9 * _vmax));
i4a = min(find(abs(y.[ii][:][1] - _ta) < _dt));
i4b = min(find(abs(y.[ii][:][1] - _tb) < _dt));
tf = y.[ii][i4a:i4b;1];
vf = y.[ii][i4a:i4b;2];

pf = zeros(1,3);

pf[1] = _vmax;

pf[2] = _ta;

pf[3] = range_C2[i];

//
cf = odrfit (vf, tf, pf, vfunc, odropts);
coef_fit = [coef_fit; [range_C2[i], cf.coef]];
]

// plot the quantity of interest
gnuwins (2);

gnuwin (1);

gnulimits (0, 50, 0, 5);

gnuxtics (5, 5);

gnuxlabel ("Time / ns");

gnuylimits (0, 5, 5);

gnuylabel ("Voltage (V)");

gnulegend (lgd_y);

gnuplot (y, "r2b_vt.eps");

gnuwin (2);

gnulimits (0.1, 200, 0.1, 200);

gnulimits2 (,,0, 1);

gnuscale ("log", "log");

gnuxtics ([0.1,0.2,0.5,1,2,5,10,20,50,100,200]);

gnuxlabel ("Capacitor C_2 (pF)");

gnuylimits ([0.1,0.2,0.5,1,2,5,10,20,50,100,200],,,"nomirror");

gnuylabel ("Time / ns", "V/V_{max}");

gnuy2tics (0.1,1);

gnucmd ("unset grid; set grid xtics ytics y2tics");

gnulegend (["Time constant for charging C_2 (ns)", "Final voltage ratio V(C_2)/V_{max}" ]);

gnuformat ("[with lines lw 2 lc rgb 'green' axes x1y1", "with lines lw 2 lc rgb 'red' axes x1y2"])n;

gnuplot (<<a=coef_fit[;1,4];b=coef_fit[;1,2]./[1,V1]>>, "r2b_tc.eps");
Example: Same as the previous example, but the calculation is performed using shared library.

```c
//
// ngspice script:
//
spicecrkt= [ ... "* rc circuit: 2 rc circuits connected by a switch"; ...
"v1 1 0 dc " + text(V1); ...
"r1 1 2 " + text(R1); ...
"c1 2 0 " + text(C1); ...
"r2 4 5 " + text(R2); ...
"c2 5 0 " + text(C2); ...
"vc 3 0 0 pulse(0 1 0 1n 1n 5u 1s)"; ...
"s1 2 4 3 0 switch"; ...
".model switch sw(vt=0.09 ron=0.01)"; ...
".ic v(2)=" + text(V1) + " v(4)=0 v(5)=0"; ...
".tran 0.05ns 200ns"; ...
".end" ... ];

// call ngspice and execute the script
spice.runckt(spicecrkt);

while (spice.isrunning())
{ sleep(0.01); }

// get the output:
s = spice.getvals();
```
Figure 15.2: (Top panel) Voltage across $C_2$ as a function of time as a function of capacitance $C_2$. (Bottom panel) Charge time of $C_2$ as a function of its capacitance.
15.2 Python

Introduction - Why Python?

The main web page from http://python.org reads,

Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.

Python runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines.

Python is free to use, even for commercial products, because of its OSI-approved open source license.

The Python Software Foundation holds the intellectual property rights behind Python, underwrites the PyCon conference, and funds other projects in the Python community.

Python has enormous number of wrapper libraries, and libraries written in native python. The purpose of the functions in this section is to allow rlab access to python functions. As of now, wrappers for numpy and scipy objects are not available.

rlab supports python through a function list py. If one does not want to use python with their rlab installation, reconfigure and recompile rlab as follows,

```bash
./configure --disable-python [...follow with other flags here..]
```

The functions in the list are organized in three groups: Administrative, Executive, and Assignment functions.

Administrative start and stop the interpreter, and provide information about the version of the interpreter.

Executive evaluate chunks of python code, either by loading the code from files, or from rlab through string variables.

Assignment transfer data between the interpreter and the rlab environment.

Administrative Functions

15.2.1 init

**Format:** istatus = py.init(/exec, sighand/)

**Arguments:**

1. `exec`, string scalar, name of the python executable;

2. `sighand`, integer scalar, signal handler;

**Result:** istatus, integer scalar, 0 on success, 1 on failure.

**Abstract:** py.init() initializes the interpreter. Without it, other functions fail.
15.2.2 kill

Format: \texttt{istatus = py.kill()}

Arguments:

Result: \texttt{istatus}, integer scalar, 0 on success, 1 on failure.

Abstract: \texttt{py.kill()} kills the interpreter. In order to use the interpreter again it has to be reinitialized.

15.2.3 ver

Format: \texttt{s = py.ver()}

Result: \texttt{s}, string scalar, version of the python interpreter accessed by the \texttt{rlab} library.

Abstract: Finds the version of the python interpreter.

Executive Functions

15.2.4 file

Format: \texttt{istatus = py.file(fn)}

Arguments:

1 \texttt{fn}, string scalar, filename of a python file to be executed by the interpreter.

Result: \texttt{istatus}, integer scalar, 0 on success, 1 on failure.

Abstract: Executes a python file within the interpreter.

15.2.5 cmd

Format: \texttt{istatus = py.cmd(c)}

Arguments:

1 \texttt{c}, string vector, sequence of commands to be executed by python interpreter.

Result: \texttt{istatus}, integer scalar, 0 on success, \texttt{-lineno} on failure, where \texttt{lineno} is the index of entry in \texttt{c} that caused the failure.

Abstract: Executes a sequence of commands within python interpreter.

Assignment and Executive Functions

15.2.6 var

1. Format: \texttt{v = py.var(/m/, name)}

Arguments:

1 \texttt{m}, string scalar, module in context of which the variable is to be accessed.
2 *name*, string scalar, name of the variable to be accessed.

**Result:** \(v\), scalar, vector or matrix, with the content of the python variable *name*.

**Abstract:** Retrieves a content of a python variable, which name is *name*.

II. **Format:** \(istatus = py.var(/m/, name, val)\)

**Arguments:**

1. *m*, string scalar, module in context of which the variable is to be created.
2. *name*, string scalar, name of the variable to be created.
3. *val*, scalar, vector or matrix, numeric or string.

**Result:** \(istatus\), integer scalar, 0 on success, 1 on failure.

**Abstract:** Creates a python variable named *name*, and assigns to it value *val*. If the variable already exists, it is destroyed first.

The correspondence is as follows:

(rlab) scalar \(\equiv\) (python) scalar,
(rlab) vector \(\equiv\) (python) list,
(rlab) matrix \(\equiv\) (python) nested lists, where a list represents a row-vector.

### 15.2.7 eval

**Format:** \(x = py.eval( c )\)

**Arguments:**

1. *c*, string scalar, command to be evaluated by python interpreter, which result is exportable (results in a python object, not necessarily named).

**Result:** \(x\), scalar, vector or matrix, that is a result of python interpreter executing the command in \(c\).

**Abstract:** Executes a commands within python interpreter and returns the result to rlab, iff it conforms to python\(\equiv\)rlab equivalence above.
Example: This is an example of how we let python do our bidding.

```python
py.init();

x1 = rand();
py.var("x", x1);
py.cmd("print 'x=',x");
y1 = py.var("x");
d1 = max(max(abs(x1-y1)));
if (d1 != 0)
{
  error("test failed!");
else
  printf("Test passed for real scalars\n");
}

x2 = rand(1,6);
py.var("x",x2);
py.cmd("print 'x=',x");
y2 = py.var("x");
d2 = max(max(abs(x2-y2)));
if (d2 != 0)
{
  error("test failed!");
else
  printf("Test passed for real vectors\n");
}

x3 = rand(4,6);
py.var("x",x3);
py.cmd("print 'x=',x");
y3 = py.var("x");
d3 = max(max(abs(x3-y3)));
if (d3 != 0)
{
  error("test failed!");
else
  printf("Test passed for real matrices\n");
}

py.cmd(["x=1.0", "y=2.0"]);  
py.cmd("print '(python): x+y=', x+y");
c=py.eval("x+y");
printf("(rlab) : x+y= %g\n", c);```
15.3 JVM

Java™ Native Interface

The functions from this group allow user to integrate Java™ (for now only) methods in rlab. The functions are accessible through a list jvm, which comprise of the entries create, destroy and call_method. The functions extend the earlier work by Ian Searle, and follow (Liang, 1999).

In order to use JVM rlab has to be built with support for java. First, one has to choose JVM shared object library (eg., in /usr/lib64/gcj-4.6-12 on opensuse 12.1) and install its development files. Only then, rlab should be configured to use it,

```
./configure --with-jvm=/usr/lib64/gcj-4.6-12 
[..follow with other flags here..]
```

By default, jvm is disabled.

Note: This is experimental set of functions. Use them at your own risk.

15.3.1 create

**Format:** istatus = jvm.create(/flag1, flag2, ...) 

**Arguments:**  

1. ... flag1, string scalar, command line parameters passed to the Java™ virtual machine (JVM) to be initialized.

**Result:** istatus, integer scalar, 0 if new JVM is successfully created, 1 if it already exists.

15.3.2 destroy

**Format:** jvm.destroy() 

**Abstract:** The function tries to destroy a JVM. As of now, it does not really work, see comment in (Liang, 1999), p.86.

15.3.3 call_method

**Format:** x = jvm.call_method jm, arg1, ... /

**Arguments:**  

1. jm, list with entries method and descriptor, where
   - method, string, class and name of the method that follows JNI format.
   - descriptor, string, description of input and output arguments after the JNI format.

arg1, ... , string, integer or double scalar or vector, that is passed to the method in the order they appear in the argument list of the function.  
As of now, the code does not check that the argument types follow the description provided in jm.descriptor.

**Result:** x, string, integer or double scalar or vector, or none, that is constructed from jm.descriptor that contains the result of calling jm.method through JVM.
Example: Call a method from java class *Prog*, see example in rlab/test/java.

```java
//
//
// jvm.create(); //

j_method0 = <<>>; // a method in the class Prog.java
j_method0.method = "Prog.gout0"; // method
j_method0.descriptor = "(Ljava/lang/String;)V"; // parameters inside brackets, followed by return type

jvm.call_method(j_method0, ["1: from RLaB"]);

j_method1 = <<>>;
j_method1.method = "Prog.main";
j_method1.descriptor = "([Ljava/lang/String;);V";

jvm.call_method(j_method1, ["1: from RLaB"]);
jvm.call_method(j_method1, ["2: from RLaB", "2: more from RLaB", "2: even more from RLaB"]);

j_method2 = <<>>;
j_method2.method = "Prog.gout1";
j_method2.descriptor = "([Ljava/lang/String;);V";

jvm.call_method(j_method2, ["1: from RLaB"]);
jvm.call_method(j_method2, ["2: from RLaB", "2: more from RLaB", "2: even more from RLaB"]);

j_method3 = <<>>;
j_method3.method = "Prog.gout2";
j_method3.descriptor = "([D)V";

jvm.call_method(j_method3, rand(1,2));
jvm.call_method(j_method3, rand());
```
TODO

1. Pass matrix arguments, get matrices from java methods.

2. Parse input arguments for the method from descriptor. Currently code uses native rlab types of data and converts to equivalent java objects, integers, doubles and strings.

3. Parse fields in the method description.

4. Create JVM objects from rlab data and make them available to java methods.

5. Pass rlab functions as arguments to java methods.
Chapter 16
Cameras and Images

16.1  GPhoto2

GPhoto2 Interface

In rlab there is a list of functions gp that allows user full control over a tethered camera through the GPhoto2 library.

By default, rlab supports the library, but user can disable that support by reconfiguring rlab through

```
./configure --disable-gphoto2 [..follow with other flags here..]
```

This requires the library to be installed on the linux system. If one builds their own rlab then the library development files have to be installed as well.

This is experimental set of functions. Use them at your own risk. Acquiring a really long sequence of images seems to cause some memory leaks that prevent the library from functioning normally: After some 400 images, the communication with the camera fails, and rlab produces bunch of “Bad parameters” messages.

The list gp comprises of the following functions.

16.1.1  supported

**Format:**  \( s = gp\text{.}\text{supported}() \)

**Result:**  \( s \), string vector, contains the names of all the camera models currently supported by the library. These names are to be used when initializing a camera for use with the library.

E.g., I have a Nikon D40 camera, and am wondering if it is supported by the library. I do this

```r
>> grep(grep(gp\text{.}\text{supported}(),"Nikon"),"D40")
Nikon DSC D40 (PTP mode)
Nikon DSC D40x (PTP mode)
>>
```

and yes, it is supported, and the camera name in the library is “Nikon DSC D40 (PTP mode)”.

407
16.1.2 init

Format: \texttt{gp.init(/name/)}

Arguments:

1. \texttt{name}, string, name of the camera under which the device appears in the list of the supported cameras. If unsure leave empty, which reverts to default value “USB PTP Class Camera”.

Abstract: The function initialize the library for work with the camera “name”. Please note, the library supports one camera at the time, so if one wants to use more then one camera then, before next camera is initialized the previous one has to be freed.

16.1.3 exit

Format: \texttt{gp.exit()}

Abstract: The function closes the currently active camera, and releases all the library resources associated with the camera.

16.1.4 info

Format: \texttt{s = gp.info()}

Result: \texttt{s=} \begin{tabular}{@{}lllll@{}}
file\_op & folder\_op & op & status & summary \\
\end{tabular}. Each entry is a string vector with description of implemented operations on camera, on its files and folders, and the status of the camera driver.

Abstract: The function provides the information about the camera driver features.

E.g., I want to configure Nikon Coolpix L22 (generic USB PTP class camera), and see what I can do with it within the library,

\begin{verbatim}
>> gp.exit()?
0
>> gp.init()?
4
>> x=gp.info()?
file_op folder_op op status summary

>> x.summary
Manufacturer: Nikon Corporation
Model: L22
  Version: COOLPIX L22 V1.0
  Serial Number: A0A5ML057722
Vendor Extension ID: 0xa (1.0)
Vendor Extension Description: microsoft.com: 1.0;

Capture Formats: JPEG
Display Formats: Association/Directory, Defined Type, JPEG, DPOF, MS AVI, Apple Quicktime

Device Capabilities:
  File Download, File Deletion, File Upload
  Generic Image Capture, No Open Capture, No vendor specific capture
  Nikon Wifi support
\end{verbatim}
Storage Devices Summary:
store_00010001:
  StorageDescription: L22
  VolumeLabel: None
  Storage Type: Removable RAM (memory card)
  Filesystemtype: Digital Camera Layout (DCIM)
  Access Capability: Read-Write
  Maximum Capability: 16005464064 (15264 MB)
  Free Space (Bytes): 16002711552 (15261 MB)
  Free Space (Images): 15649

Device Property Summary:
Property 0xd407:(read only) (type=0x6) 1
Property 0xd406:(readwrite) (type=0xffff) ''
Property 0xd002:(readwrite) (type=0x6) Enumeration [1,2,3,4,5,6,7] value: 3
Date & Time(0x5011):(readwrite) (type=0xffff) '20181005T101954'
Flash Mode(0x500c):(readwrite) (type=0x4) Enumeration [1,2,3,4] value: Automatic flash (1)
Focus Mode(0x500a):(readwrite) (type=0x4) Enumeration [2,3] value: Automatic (2)
Focal Length(0x5008):(read only) (type=0x6) Enumeration [3500,4600,5300,6100,7300, 8600,10500] value: 46 mm (4600)
Battery Level(0x5001):(read only) (type=0x2) Enumeration [2,5,25,50,65,80,100] value: 50% (50)
Property 0xd303:(read only) (type=0x2) 1

This means that through the driver (and thus through rlab) one can capture images and configure camera. However, changing the focal length of the zoom lens, parameter 0x5008, is not possible through the interface because it is readonly.

16.1.5 config_options

Format: gp.config_options()

Result: s, list, which entries are the options or parameters of the camera that are accessible. If more then one value is listed for some parameter, these represent the available choices not necessarily changeable by the user, e.g., focal length of the zoom lens (has to be set manually on the body of the camera and not through the interface). Nikon Coolpix L22 has very few configurable options. Some other cameras, e.g., Nikon D40 may have many more, e.g., aperture (f-number),

```bash
>> a=gp.config_options();
>> a.['f-number']?
/4.2 /4.5 /5. /5.6 /6.3 /7.1 /8 /9 /10 /11
/13 /14 /16 /18 /20 /22 /25 /29
>>
```
16.1.6 config

Format: \( s = \text{gp.config}(\text{feature},/\text{value}/) \)

1. feature, string, the camera feature which value we want to read, or change.

2. value, string, number or range. If provided, then this value is set for the “feature”. If feature is of type radio, then value can be either one of the available values obtained using config_options or its index.

Result: \( s \), integer, or string, or range.

Abstract: The function reads or changes a camera feature. E.g., to find out current “f-number” on the camera,

\[
\begin{align*}
\text{>> } \text{gp.config}("f\text{-number}") \\
f/10 \\
\text{>> }
\end{align*}
\]

The following two commands do the same thing

\[
\begin{align*}
\text{>> } \text{gp.config}("f\text{-number","f/5")} \\
\text{>> } \text{gp.config}("f\text{-number", 3)}
\end{align*}
\]

because the third entry in the string vector, which is the output of \text{gp.config-options()}, is “f/5”.

16.1.7 capture

Format: \( \text{gp.capture}(/\text{filename}/) \)

1. filename, string, file under which the captured image is saved on local file system. If not provided the image stays in the camera memory only.

Abstract: The function makes the camera to capture the image. If filename is not given, then the image remains on the camera file system. Otherwise, the image is transferred to the local file system and deleted on the camera.
Example: Flip through all f-numbers and all exposure times (except indefinite) settings of a Nikon D40 camera, and for each setting capture an image to local file system.

```plaintext
mycamera = "Nikon DSC D40 (PTP mode)";

gp.init(mycamera)

xopt = "shutterspeed";
yopt = "f-number";
c_speed = gp.config(xopt); // remember initial setting of the camera
c_fnum = gp.config(yopt); // remember initial setting of the camera

"current value for '" + xopt + "' is '" + c_speed + "\n"
"values for '" + xopt + "' are"
x=gp.config_options(xopt)

"current value for '" + yopt + "' is '" + c_fnum + "\n"
"values for '" + yopt + "' are"
y=gp.config_options(yopt)

mkdir("photo");
for (i in range(x))
{
    if (i==len(x)) // skip last 'shutterspeed' - indefinite time
        { break; }
    for (j in range(y))
    {
        speed = x[i];
        fnum = y[j];
        gp.config(xopt,speed);
        gp.config(yopt,fnum);
        fnum2 = gsub("f","f/",fnum).string;
        if (!strindex(fnum2,".\n")
            { fnum2 = fnum2 + ".0"; }
        fn = "photo/snap_" + speed + "."n + fnum2 + ".jpg"
        gp.capture(fn); // capture image
    }
}

gp.config(xopt,c_speed);
gp.config(yopt,c_fnum);
gp.exit();
```
16.2 ImageMagick

Introduction

On the web site http://www.imagemagick.org one can read:

ImageMagick\(\textregistered\) is a software suite to create, edit, compose, or convert bitmap images. It can read and write images in a variety of formats (over 100) including DPX, EXR, GIF, JPEG, JPEG-2000, PDF, PhotoCD, PNG, Postscript, SVG, and TIFF. Use ImageMagick to resize, flip, mirror, rotate, distort, shear and transform images, adjust image colors, apply various special effects, or draw text, lines, polygons, ellipses and Bzier curves.

rlab implements selection of functions from MagickWand C API. By default, rlab supports the library. This requires the library and its development files to be installed on the linux system. User can disable that support by reconfiguring rlab through

```
./configure --disable-im [...] (follow with other flags here..)
```

and then rebuilding rlab (linux mantra: make clean; ./configure ...; make rlab)

Note: ImageMagick support is in experimental stage, that is, use it at personal risk.

rlab support follows the internal organization of ImageMagick: There exist a collection of (magick) wands, where each (magick) wand comprises of a linked list of images. All (magick) wand operations act on a default (magick) wand, while all image operations act on the currently active image on the default (magick) wand.

Current limits are: 32 as a maximal number of (magick) wands, and 512 as a maximal number of images in a (magick) wand.

The library is implemented through a function list image, which comprises of the following functions in alphabetical order:

```plaintext
>> image
  append  art  clear  clone  close
  clut   combine  deconstruct  diff  disp
  distort  exit  f  init  iter
  join  method  opt  pixels  prop
  read  specs  wand  write
>>
```
Library and (magick) wand related functions

16.2.1 init
Format: image.init()
Abstract: Initializes ImageMagick library and all its internal structures.

16.2.2 exit
Format: image.exit()
Abstract: Closes ImageMagick library and releases all its resources.
Note: It is a good practice to start all scripts with image.exit() to close previous instance of the library.

16.2.3 wand
Format: l = image.wand([n])
Arguments:
1. n, integer scalar, index of (new) (magick) wand.
Result: l, list with entries ≪ wand; image; images≫. where
   1. wand, integer scalar, index of a default (magick) wand.
   2. image, integer scalar, index of currently active image on the default (magick) wand.
   3. images, integer vector, indices of all images on the default (magick) wand.
Abstract: If index n is provided: If (magick) wand n does not exist, it initiates the library, and then creates a new (magick) wand which is accessed through this number. Finally, it makes (magick) wand n a default one. If index n is not provided, then returns information about the default (magick) wand. If the default (magick) wand does not contain any images, image and images are empty arrays. If library is not initialized and no arguments are provided, the function returns an empty list and does not initialize the library.

Functions related to images on a default (magick) wand

16.2.4 write
Format: image.write([filename][,[flag]])
Arguments:
1. filename, string.
2. flag, integer.
Abstract: Saves active image from default (magick) wand to file. If flag all is present then all images on default (magick) wand are saved. If filename is not provided then image property filename is accessed, and used for saving the file. This property is also accessible through function image.specs, see later.
16.2.5 read

Format: `image.read(filename[, pos])`

Arguments:

1. `filename`, string.
2. `pos`, string: “first”, “last”, or “current”. A position in default (magick) wand where the image is to be red. Default value is “last”, meaning that the red image becomes the last image in default (magick) wand.

Reads image from the file `filename` into the desired position in default (magick) wand. Makes the image active.

Note: Repeated call to the function will load the same image same number of times in default (magick) wand.

16.2.6 iter

Format: `r = image.iter([d])`

Arguments:

1. `d`, string, “first”, “next”, “last”, or “previous”, or integer in range of the available images in default (magick) wand.

Result: `r`, integer. If argument `d` is omitted, then `r` contains the index of the active image in default (magick) wand. Otherwise it contains 0, if the iteration along default (magick) wand was successful, or 1 if it failed. The failure occurs if one tries to iterate past the end of default (magick) wand.

16.2.7 clone

Format: `image.clone(j)`

Arguments:

1. `j`, integer scalar, index of new (magick) wand that will be (i), an exact replica of default (magick) wand, and (ii), new default (magick) wand.

Abstract: Clones default (magick) wand to a new (magick) wand, and makes it default.

16.2.8 clear

Format: `r = image.clear()`

Result: `r`, integer scalar, index of default (magick) wand.

Abstract: Clears all images in default (magick) wand.
16.2.9  close

Format:  \texttt{r = image.close()}

Result:  \( r \), integer scalar, index of default (magick) wand.

Abstract: Clears all images in default (magick) wand then closes it. If there are open wands, then returns the index of new default (magick) wand. If this was the last open (magick) wand, then closes the library, as well.

16.2.10  join

Format:  \texttt{image.join(j)}

Arguments:

1.  \( j \), integer scalar, index of (magick) wand which images will be added to default (magick) wand.

Abstract: Adds all images from (magick) wand \( j \) to default (magick) wand.

16.2.11  append

Format:  \texttt{image.append(j[,d])}

Arguments:

1.  \( j \), integer scalar, index of (magick) wand which will contain the appended images from active onward in default (magick) wand.

2.  \( d \), string, “l” for left-to-right, or “t” for top-to-bottom, for the direction of operation. Default is “l”.

Abstract: Appends a selection of images in default (magick) wand to a single image in given direction in new (magick) wand. Makes new (magick) wand default.

16.2.12  combine

Format:  \texttt{image.combine(j[,channel])}

Arguments:

1.  \( j \), integer scalar, index of (magick) wand which will contain the appended images from active onward in default (magick) wand.

2.  \( channel \), string, contains not more than 5 characters from the following selection, “r/g/b” for red/green/blue channels, “c/m/y” for cyan/magenta/yellow channels, “a” or “o” for alpha or opacity. “B” or “k” for black, “M” or “t” for matte. E.g., “rgba” or “cmyk”.

Abstract: Combines a selection of images with respect to the channels in default (magick) wand to a single image in new (magick) wand. Makes new (magick) wand default.
16.2.13  diff

Format: \( n = \text{image.diff}(j, \text{metric}[, \text{channel}]) \)

Arguments:

1. \( j \), integer scalar, index of (magick) wand which will contain the appended images from active onward in default (magick) wand.

2. \( \text{metric} \), string scalar, metric used in establishing a difference between images: “abs” for absolute error, “meanabs” mean absolute error, “meannpp” for mean error per pixel, “square” for mean squared error metric, “peakabs” for peak absolute error metric, “peaksnr” for peak signal to noise ratio metric, “rms” for root mean squared error metric, “normcc” for normalized cross-correlation error metric and “fuzz” for fuzz error metric.

3. \( \text{channel} \), string, contains not more than 5 characters from the following selection, “r/g/b” for red/green/blue channels, “c/m/y” for cyan/magenta/yellow channels, “a” or “o” for alpha or opacity. “B” or “k” for black, “M” or “t” for matte. E.g., “rgba” or “cmyk”.

Result: \( n \), integer scalar, index of new default (magick) wand which contains the differences between images from default (magick) wand and (magick) wand \( j \).

Note: Unintended results occur if default (magick) wand prior to call to this function contains two identical images (their difference turns out to be a null pointer).

16.2.14  composite

Format: \( \text{image.composite}(j, \text{op}[, \text{channel}]) \)

Arguments:

1. \( j \), integer scalar, index of (magick) wand which will contain the appended images from active onward in default (magick) wand.


3. \( \text{channel} \), string, contains not more than 5 characters from the following selection, “r/g/b” for red/green/blue channels, “c/m/y” for cyan/magenta/yellow channels, “a” or “o” for alpha or opacity. “B” or “k” for black, “M” or “t” for matte.

E.g., “rgba” or “cmyk”.

Abstract: Applies composite operator to active image in default (magick) wand. Creates new default (magick) wand, and its active image contains the result.
16.2.15  deconstruct

Format: \texttt{image.deconstruct()}

Abstract: Replaces images in default (magick) wand with their deconstructed version.

16.2.16  distort

Format: \texttt{image.distort\(\text{dist, [\text{params}, bestfit]}\)\rangle}

Arguments:


2. \texttt{params}, real vector, parameters for the distortion.

3. \texttt{bestfit}, integer scalar, 1 to use bestfit option of the distortion, 0 not to.

Abstract: Replace active image in the default (magick) wand, with its destorted version, which is also made active.

16.2.17  clut

Format: \texttt{image.clut\(\text{clutwand[,channel]}\)\rangle}

Arguments:

1. \texttt{clutwand}, integer scalar, index of (magick) wand with a single image that represents a color-lookup table;

2. \texttt{channel}, string.

Abstract: Converts colors in active image in default (magick) wand using the color lookup table from \texttt{clutwand}.

16.2.18  pixels

\texttt{1 Format: pixel\_data = image.pixels\(w, h, pixel\_map[, qrange]\)\rangle}

Arguments:

1. \texttt{w}, integer pair \(y_1, y_2\), representing pixel range \(y_1 : (y_1 + y_2 - 1)\);

2. \texttt{h}, integer pair \(x_1, x_2\), representing pixel range \(x_1 : (x_1 + x_2 - 1)\);

3. \texttt{pixel\_map}, string scalar, contains names of the channels to be extracted;

4. \texttt{qrange}, integer scalar, provides quantum range for extracted pixels.

Result: \texttt{pixel\_data}, list \([\text{pixel\_map}; \text{pixel\_x}; \text{pixel\_y}; \text{height}; \text{width}]\rangle\), where
CHAPTER 16. CAMERAS AND IMAGES

- \textit{pixel\_map}, string, first letters of extracted channels, e.g., \textquotedblright rgba\textquotedblright;
- \textit{pixel}, list which entries are labeled after individual letters from \textit{pixel\_map}. E.g., pixel.a, pixel.b, pixel.g, pixel.r, where each is an integer matrix of dimensions height-times-width containing respective color values of pixels from the image;
- \textit{x}, \textit{height}, integer scalars, x-range of extracted pixels, that is \(x : (x + \textit{height} - 1)\). Note that \(x\) could be called an offset;
- \textit{y}, \textit{width}, integer scalars, y-range of extracted pixels, that is \(y : (y + \textit{width} - 1)\). Note that \(y\) could be called an offset.

\textbf{Abstract:} Extracts pixels from active image of default (magick) wand to integer matrices, which entries are in range 0 to \textit{qrange}.

\textbf{II Format:} \texttt{image.pixels(pixel\_data)}

\textbf{Arguments:}

1. \textit{pixel\_data}, list, see above.

\textbf{Abstract:} Inserts pixels from list \textit{pixel\_data} into active image of default (magick) wand. If active image does not exist, say, because default (magick) wand is empty, then it creates an image that fits all of its dimensions (offset with size).

\textbf{16.2.19 f}

\textbf{Format:} \texttt{image.f(oper, param[,channel])}

\textbf{Arguments:}

1. \textit{oper}, string scalar, name of the operator or function that is applied to the active image in default (magick) wand. Available operators are (with the alternative names inside the brackets): \textquotedblright and\textquotedblright, \textquotedblright divide\textquotedblright (\texttt{\slash}), \textquotedblright shiftleft\textquotedblright (\texttt{\textless{}\textless{}}), \textquotedblright max\textquotedblright, \textquotedblright min\textquotedblright, \textquotedblright multiply\textquotedblright (\texttt{\ast}), \textquotedblright or\textquotedblright, \textquotedblright shiftright\textquotedblright (\texttt{\textgreater{}\textgreater{}}), \textquotedblright set\textquotedblright, \textquotedblright subtract\textquotedblright (\texttt{\textminus{}}, or \texttt{-}), \textquotedblright xor\textquotedblright, \textquotedblright pow\textquotedblright, \textquotedblright log\textquotedblright, \textquotedblright threshold\textquotedblright, \textquotedblright thresholdblack\textquotedblright, \textquotedblright thresholdwhite\textquotedblright, \textquotedblright gaussian\textquotedblright (\texttt{noisegaussian}), \textquotedblright noiseimpulse\textquotedblright (\texttt{impulse}), \textquotedblright noiselaplace\textquotedblright (\texttt{laplace}), \textquotedblright noisemultiplicative\textquotedblright (\texttt{multiplicative}), \textquotedblright noisepoisson\textquotedblright (\texttt{poisson}), \textquotedblright noiseuniform\textquotedblright (\texttt{uniform}), \textquotedblright cos\textquotedblright, \textquotedblright sin\textquotedblright, \textquotedblright addmodulus\textquotedblright (\texttt{+mod}), \textquotedblright mean\textquotedblright, \textquotedblright abs\textquotedblright, \textquotedblright exp\textquotedblright, and \textquotedblright median\textquotedblright.

Available functions are \texttt{poly} for polynomial, \texttt{asin} for arc-sine, and \texttt{atan} for arc-tangent.

2. \textit{param}, real scalar or vector, parameter for the operation or function.

3. \textit{channel}, string.

\textbf{Abstract:} Replace active image in the default (magick) wand, with a result of applying the operation or function and its parameter(s) on the active image or a channel of it.

\textbf{16.2.20 method}

\textbf{Format:} \texttt{image.method(method, [param1,...]})

\textbf{Arguments:}

Methods in great details

- **“annotate”**
  **Format:** `image.method(“annotate”, text, pos, params)`

  **Arguments:**
  1. **text**, string to be placed;
  2. **pos**, vector [x,y], position where to place the string (right justified);
  3. **params**, list with entries:
     - **font**, string, IM compliant description of the font e.g., “Times-Roman-Bold”;
     - **font_color**, string, IM compliant description of the color, e.g., “white” or “#aabbcc” for r-channel 0xaa, g-channel 0xbb and b-channel 0xcc;
     - **font_size**, integer, size of fonts in pixels;
     - **angle**, real scalar, angle in degrees at which the **text** will be placed (0° is default).

- **“rotate”**
  **Format:** `image.method(“rotate”, bkg, angle)`

  **Arguments:**
  1. **bkg**, vector [r,g,b], the color that will be used for background as size of the image changes due to rotation;
  2. **angle**, real scalar, angle by which the image will be rotated around its center.

Input parser is kind enough that if it cannot recognize the method, it will inform user about it. Affine method is implemented, as well, but it doesn’t do anything except adds a layer of pixels around the image with each call. The issue may have been fixed in more recent updates to the library. Or one can try `image.distort(…)`.

2. **param1, …**, parameters for the method. Here, I direct the reader to exhaustive descriptions on ImageMagick web site. The parameters follow the convention and notation listed therein, all in small letters.

Some distinctive methods and their parameters are, in no particular order:

- **“addnoise[]”**: First parameter is “noisetype”, string scalar: “u” for uniform, “g” or “n” for gaussian, “m” for multiplicative gaussian, “i” for impulse, “l” for laplacian, “p” for poissonian; Second parameter
is integer scalar: the maximal magnitude of the noise.
I wonder if providing a negative number turns the operation to “subtractnoise[]”?  


- Methods that require pixel wand for their arguments, are implemented so that user has to provide three- or four-component vector for rgb or rgba values of the pixel. In most cases the vector should be real, in range 0 to 1, which is internally multiplied with the image quantum range (256 for 8-bit depth, or 65535 for 16-bit depth).

16.2.21 specs

I **Format:** \textit{image.specs(s, val)}

**Arguments:**


2. \textit{val}, string or real scalar, or real vector, value of the property \textit{s}.

II **Format:** \textit{s = image.specs()}

**Result:** \textit{s}, list \texttt{≪alpha, redprimary, greenprimary, blueprimary, colorspace, compression, compressionquality, depth, ...≫}

**Abstract:** Retrieve all properties, specifications et c., associated with active image in default (magick) wand, or change some of them.

16.2.22 art

**Format:** \textit{image.art([ar],[val])}

1. \textit{ar}, string scalar, name for the image artifact;

2. \textit{val}, string or real scalar, value of the artifact.

**Abstract:** Adds an artifact \textit{ar} to the active image on default (magick) wand, or access its value if it exists.

16.2.23 opt

**Format:** \textit{image.opt([option],[val])}

1. \textit{option}, string scalar, name for the image option;

2. \textit{val}, string or real scalar, value of the option.

**Abstract:** Adds an \textit{option} to the active image on default (magick) wand, or access its value if it exists.
16.2.24 disp

Format: \texttt{image.disp([xserv])}

1. \texttt{xserv}, string scalar, X-server to be used in displaying the image;

Abstract: Displays active image on default (magick) wand on X-server.

Note: \texttt{image.disp} transfers control to display, and it does not return to \texttt{rlab} until one closes the display window. This is bad, but I don’t think there is any other way with embedded viewers. Alternative would be to write image to a file which separate application reads and displays, and rereads when the image is updated.
Example: Consider script that creates image in Fig. 16.1.

```c
fn = "./photo/randomnoise.jpg";
fdir = "/.
fn = "randomnoise";
fn = "./photo/";
```

```c
image.wand(1);
```

```c
image.method("annotate", "R", [220, 80], <font_size=36;font_color=_c;font="Times-Roman-Bold">); image.method("annotate", "L", [10, 80], <font_size=36;font_color=_c;font="Times-Roman-Bold">); image.method("annotate", "U", [116, 30], <font_size=36;font_color=_c;font="Times-Roman-Bold">); image.method("annotate", "D", [116, 120], <font_size=36;font_color=_c;font="Times-Roman-Bold">);
```

```c
// get pixel map
pic = image.pixels([1, 256], [1, 128], "rgba", 255);
image.clear();
```

```c
// draw lines at 60, 15 to 241
idx_row = 62;
for (idx_col in 15:241)
{
  for (k in -1:1)
  {
    pic.pixel.a[idx_row+k;idx_col] = 255L;
    pic.pixel.r[idx_row+k;idx_col] = 255L;
    pic.pixel.g[idx_row+k;idx_col] = 255L;
    pic.pixel.b[idx_row+k;idx_col] = 255L;
  }
}
```

```c
idx_col = 129;
for (idx_row in 15:96)
{
  for (k in -1:1)
  {
    pic.pixel.a[idx_row;idx_col+k] = 255L;
    pic.pixel.r[idx_row;idx_col+k] = 255L;
    pic.pixel.g[idx_row;idx_col+k] = 255L;
    pic.pixel.b[idx_row;idx_col+k] = 255L;
  }
}
```

```c
image.pixels(pic);
image.specs("resolution", [120, 120]);
image.write("./photo/randomnoise_annotated.jpg");
```
Figure 16.1: An image that was created using script on previous page. The file was imported to this
document with `\includegraphics[scale=1.5]`. 
Example: Consider script that creates image in Fig. 16.2.

```plaintext
fdir = "./photo/";
fn = "oriented_circle";
fext = ".png";

image.wand(1);
image.read(fdir + fn + fext);
s_0 = image.specs().dim;
r_0 = image.specs().resolution;

d_angle = 30;
end_angle = 90;
for (angle in 0:end_angle:d_angle)
{
    // wand 1: image at 0
    image.wand(1); // set default to 1
    // wand 2: clone 1 here and rotate it
    image.clone(2);
s_angle = num2str(angle,"rot\%03.0fdeg");
    image.method("rotate", [255L,255L,255L], angle);
s_new = image.specs().dim;
p_1 = int(0.5*(s_new[1] - s_0[1])); // padding on each side
    // remove padding
    pic = image.pixels([p_1,s_new[1]-p_1],[p_1,s_new[2]-p_1],"rgba", 255);
    image.clear(); // kill 2

    // manipulate: forget offset
    image.wand(2);
    pic.x = 1;
    pic.y = 1;
    image.pixels(pic); // drop just extracted pixels into the wand
    image.specs("resolution",r_0);
    image.write(fdir + fn + s_angle + fext);
    image.clear(); // kill 2
}
```
Figure 16.2: A sequence of four images. The original was created using Xfig - unit circle with XY-coordinate system (first on the left). Subsequent images are created using “rotate” method of IM by angles of 30, 60 and 90°, and written in PNG format. Note: using other formats introduces frame around the original image that is rotated with the image.
16.3 Random topics unfit for other sections

Source Extraction and Photometry of Astronomical Images

The solvers in this group are based on the SEP library [Barbary 2016], which provides access to function from Source Extractor by [Bertin & Arnouts 1996]. The names of the solvers and their input and output parameters mostly follow their python implementation from [Barbary 2016].

16.3.1 sep

- **Format:** \( b = \text{sep.background}(x, \text{opts}) \)

  **Arguments:**
  
  1. \( x \), integer or double matrix, image to be processed.
  2. \( \text{opts} = << \text{tile}; \text{filter \_thresh}; \text{mask}; \text{mask \_thresh} >> \), list of options for the solver:
     
     - \text{tile} = [bw,bh]: size of sub-frames used in extraction of image-varying background;
     - \text{mask}, \text{mask \_thresh}: double matrix of the same size as \( x \), scalar. If entry in the matrix \text{mask} is greater than \text{mask \_thresh} the corresponding pixel in the image is masked (ignored).
     - \text{filter} = [fw,fh]: pair of integers, size of the median filter used for detecting the background ([1,1] means no filtering)
     - \text{filter \_thresh}: double scalar, threshold for median filter when such is needed, see discussion of background estimation in [Bertin & Arnouts 1996].

  **Result:** \( b = << \text{back}; \text{rms}; \text{globalback}; \text{globalrms} >> \), list, containing the global background properties median and standard deviation, and matrices of size \( x \) containing background and standard deviations per pixel locations.

- **Format:** \( b = \text{sep.extract}(d, \text{opts}) \)

  **Arguments:**
  
  1. \( d \), integer or double matrix, image to be processed. If this is done in continuation from background analysis, then \( d = x - b.\text{back} \) is original image with background already subtracted.
  2. \( \text{opts} = << \text{noise}; \text{noise \_var}; \text{thresh} >> \), list of options for the solver:
     
     - \text{noise}, \text{noise \_var}: representation of the noise in the image, either standard deviation global (scalar) or local (matrix of per pixel values), or globar variance (scalar).
     - \text{thresh}, double scalar, relative threshold for source extraction that combines with the noise parameter for global threshold \( \text{thresh} \times \text{noise} /\text{noise \_var}^{0.5} \), or local threshold \( \text{thresh} \times \text{noise}[i;j] \),

  **Result:** \( b = << \text{back}; \text{rms}; \text{globalback}; \text{globalrms} >> \), list, containing the global background properties median and standard deviation, and matrices of size \( x \) containing background and standard deviations per pixel locations.
Chapter 17

Other Hardware Interfaces

17.1 General Purpose Interface Bus (GPIB)

Introduction

General Purpose Interface Bus (GPIB, IEEE-488, HP-IB) is a communication protocol for scientific instrumentation. The shared object library gpib.so allows user access to supported instruments within rlab. The library comprises wrapper functions for functions from the GPIB-Linux (Hess 2006) shared object library.

The source code for the library gpib.so can be obtained from rlabplus.sourceforge.net.

Control Functions

17.1.1 ibdev

Format: \( ud = \text{ibdev}(\text{minor}, \text{pad}, \text{sad}/, \text{options}/) \)

Arguments:

1. \text{minor}, index of the board;
2. \text{pad}, primary address of a device;
3. \text{sad}, secondary address of a device;
4. \text{options} = \langle \langle \text{timeout}; \text{send_coi}; \text{eos} \rangle \rangle, the list the entries of which are
   - \text{timeout}, string, acceptable values are “TNONE”, “T10us”, “T30us”, “T100us”, “T300us”, “T1ms”, “T3ms”, “T10ms”, “T30ms”, “T100ms”, “T300ms”, “T1s”, “T3s”, “T10s”, “T30s”, “T100s”, “T300s”, “T1000s”.
   - \text{send_coi}, integer. If non-zero then the EOI line is asserted with the last byte sent during writes.
   - \text{eos}, character or integer, specifies the end-of-string (EOS) character or its ASCII code. Non-zero value implies that the reception of EOS character should terminate the reads.
   - \text{eos_mode}, string, acceptable values are “REOS” that enables termination of reads when \text{eos} is received; “XEOS” that asserts EOI line whenever \text{eos} character is send during writes; “BIN” that matches the \text{eos} character using all 8 bites. \text{eos_mode} is read only if \text{eos} is provided.
CHAPTER 17. OTHER HARDWARE INTERFACES

**Result:** $ud$, an integer, descriptor of a device.

**Abstract:** ibdev initializes the instrument and returns the descriptor $ud$ which should be used in consequent communications with the device. Default values for the device is what ever is set in `/etc/gpib.conf`, with the exception of $eos = "\n"$ (new line) and $eos\_mode = "REOS|BIN$).

### 17.1.2 ibfind

**Format:** $ud = ibfind(s)$

**Arguments:**

1. $s$, character descriptor of a device, as given in `/etc/gpib.conf`.

**Result:** $ud$, an integer, descriptor of a device.

**Abstract:** ibfind initializes the instrument and returns the descriptor $ud$ which should be used in consequent communications with the device. If using ibfind then the details of the communications should be set in the file `/etc/gpib.conf` rather than through ibdev.

### 17.1.3 ibonl

**Format:** $i = ibonl(ud,\,on/)$$

**Arguments:**

1. $ud$, an integer, descriptor of a device.
2. $on$, an integer, if given than its value should be unity.

**Result:** $i$, the status of the device/board as reported by ibsta byte.

**Abstract:** ibonl without the second argument releases the resources associated with the device the descriptor of which is $ud$. If the second argument is given than the values for that device are set to defaults (either from the file or as set by ibdev).

### 17.1.4 ibclr

**Format:** $i = ibclr(ud)$

**Arguments:**

1. $ud$, an integer, descriptor of a device.

**Result:** $i$, the status of the device/board as reported by ibsta byte.

**Abstract:** ibclr clears the device.

### 17.1.5 ibsre

**Format:** $i = ibsre(ud,\,on/)$$

**Arguments:**
1. *ud*, an integer, descriptor of a device.

2. *on*, an integer, if given than its value should be unity.

**Result:** *i*, the status of the device/board as reported by *ibsta* byte.

**Abstract:** If *on* is not given than the REN (Remote Enable) line is unasserted, if it is given and is unity than the REN line is asserted. The board has to be a system controller.

### 17.1.6 *ibsic*

**Format:**

\[ i = ibsic(ud) \]

**Arguments:**

1. *ud*, an integer, descriptor of a device.

**Result:** *i*, the status of the device/board as reported by *ibsta* byte.

**Abstract:** *ibsic* performs the interface clear, i.e., it resets the GPIB bus. The board *ud* becomes controller-in-charge.

### 17.1.7 *ibln*

**Format:**

\[ i = ibln(ud, pad, sad) \]

**Arguments:**

1. *ud*, an integer, descriptor of the board;
2. *pad*, primary address of the device being queried;
3. *sad*, secondary address of the device being queried.

**Result:** *i*, an integer.

**Abstract:** *ibln* checks whether the device with primary and secondary address *pad* and *sad* is present or not. In the former case *i* is nonzero, otherwise it is set to zero.

### I/O Functions

### 17.1.8 *ibwrt*

**Format:**

\[ i = ibwrt(ud, cmds) \]

**Arguments:**

1. *ud*, an integer, descriptor of a device;
2. *cmds*, a string vector, a sequence of commands passed to the device *ud*.

**Result:** *i*, the status of the device/board as reported by *ibsta* byte, following writing of each command line from *cmds*.

**Abstract:** *ibwrt* writes the character information contained in the string vector *cmds* to the device *ud*. 
17.1.9 ibrd

Format: \( c = \text{ibrd}(ud, nbytes) \)

Arguments:

1. \( ud \), integer, descriptor of a device;
2. \( nbytes \), integer (optional), number of bytes to be read from the device \( ud \).

Result: \( c \), integer array of length \( nbytes \); or string if \( nbytes \) was not provided.

Abstract: Function receives the response of the device \( ud \). If \( nbytes \) is given then these are returned in an integer row vector. Conversely, if \( nbytes \) is omitted than the response of the device \( ud \) is returned in a single string, where the transfer of the response terminates when the first '\0' is encountered.

Note: The functions \( \text{char} \) and \( \text{ascii} \) can be used to efficiently manipulate the response of the instrument.

Note: One could conceivably use character read (\( nbytes \) omitted) for processing a response of an instrument to a query pertaining to its settings, e.g., for oscilloscope Tektronix TDS210 (\( ud3 \)) to find out its horizontal scale.

\[
\begin{align*}
\text{>> ibwrt(ud3, "HOR:MAI:SCA?"); ibrd(ud3)} \\
5.0E-9 \\
\text{>> ibwrt(ud3, "HOR:MAI:SCA?"); s=ibrd(ud3,10)} \\
\text{matrix columns 1 thru 6} \\
| \text{53} | \text{46} | \text{48} | \text{69} | \text{45} | \text{57} | \\
\text{matrix columns 7 thru 10} \\
| \text{10} | \text{0} | \text{0} | \text{0} | \\
\text{>> char(s)} \\
5.0E-9
\end{align*}
\]

In both cases, a postprocessing of the reply is necessary. Here, the function \( \text{strtod()} \) suffices to get the numeric value.

17.1.10 ibqrd

Format: \( c = \text{ibqrd}(ud, nbytes, query, n) \)

Arguments:

1. \( ud \), integer, descriptor of a device;
2. \( nbytes \), integer (optional), number of bytes to be read from the device \( ud \), in a single query;
3. \( query \), string vector, a sequence of commands to be passed to the device \( ud \);
4. \( n \), integer (optional), number of times the device will be queried.
Result: $c$, integer matrix of size $n$ rows and $nbytes$ columns containing the reply of the instrument to query; or, vector of $n$ string entries, where each string represents a reply of the device to the query.

Abstract: $ibqr{d}$ performs queried read of the device $ud$ where the repetition is performed at the lower level.

Note: The purpose of this command is to increase the speed of querying the device as doing it through script may not be fast enough for some applications.
**Example:** Communication with Keithley instrument to retrieve the measured values (VDC) at maximal rate.

```c
1  //
2  // eg_keithley.r: short the terminals for voltage measurement
3  //
4  rfile libgpib.so
5  // initialize gpib devices
6  if (!exist(ud1))
7    {
8      ud1 = ibdev(0,0,0); // gpib main board, here gpib-usb-hs, set to 0
9    }
10   if (!exist(ud2))
11    {
12      ud2 = ibdev(0,7,0); // set keithley’s gpib-address to 7
13    }
14  // number of readings
15  N = 100;
16  // data list for plotting
17  mydata = <<>>;
18  // use for storing the numerical values
19  x = zeros(N,2);
20  // initialize keithley:
21  // 1: SYST:PRES  'Continuous measurement mode (INIT:CONT ON)'
22  // 2: FUNC 'VOLT:DC'  'Select DCV function'
23  // 3: VOLT:DC:NPLC 0.1 'Set maximum read rate (FAST)'
24  mycode = [ "SYST:PRES", "FUNC 'VOLT:DC'", "VOLT:DC:NPLC 0.1"];
25  ibwrt(ud2,mycode);
26  //
27  // read N data points through internal loop
28  //
29  r = ibqrd(ud2, ["SENS:DATA:FRES?"], N);
30  // To above query Keithley replies with
31  # -1.10276960E-06VDC,+6606.482SECS,+66586RDNG#
32  # this is to be extracted into a matrix with columns
33  # [time, voltage]
34  datarange = [1:strindex(r[1],"VDC")-1];
35  timerange = [strindex(r[1],"VDC")+4:strindex(r[1],"SECS")-1];
36  for (i in 1:r.nr)
```
17.1. GENERAL PURPOSE INTERFACE BUS (GPIB)

```c
{ 
    x[i;1] = strtod( substr(r[i], timerange) );
    x[i;2] = 1e6 * strtod( substr(r[i], datarange) );
}
mydata.a = x;

// single query reading within rlab loop
//
for (i in 1:N)
{
    r = ibqrd(ud2, ["SENS:DATA:FRES?"]);
    x[i;1] = strtod( substr(r, timerange) );
    x[i;2] = 1e6 * strtod( substr(r, datarange) );
}
mydata.b = x;

// read/write query within rlab loop
//
for (i in 1:N)
{
    ibwrt(ud2, "SENS:DATA:FRES?");
    r = ibrd(ud2);
    x[i;1] = strtod( substr(r, timerange) );
    x[i;2] = 1e6 * strtod( substr(r, datarange) );
}
mydata.c = x;

// plot
xlabel ( "Instrument Time (sec)" );
ylabel ( "Instrument Reading (\gm\V)" );
plegend( ["N queries", "1 query", "ibwrt/ibrd"] );
plot ( mydata );

// reading times difference internal/external
ta = last(mydata.a)[1] - mydata.a[1;1];
tb = last(mydata.b)[1] - mydata.b[1;1];
tc = last(mydata.c)[1] - mydata.c[1;1];
printf("Times for %g measurements\n", N);
printf("N queries : %g sec\n", ta);
printf("1 query : %g sec\n", tb);
printf("ibwrt/ibrd: %g sec\n", tc);
```
Example: Communication with Textronic TDS-210. This requires binary manipulation functions as implemented in rlab using functions ascii, char, and internal integer representation of the values for which logical operators act bitwise.

```c
// eg_tds210.r: ch1 terminal on probe compensation
//
rfil libgpib.so

// initialize gpib devices
if (!exist(ud1))
{
  ud1 = ibdev(0,0,0); // init gpib main board, here gpib-usb-hs, pid,sid = 0,0
}
if (!exist(ud3))
{
  ud3 = ibdev(0,6,0); // init tds210, pid,sid = 0,6
}

// get the responses of the device
//
reply = []; for (i in 1:255)
{
  r = ibask(ud3, i);
  if (isempty(r)){ continue; }
  reply = [reply; i, r];
}
printf("Configuration bits for the device tds210 (ud3):
");
reply

// basic communication to set-up waveform parameters
//
ibclr( ud3 ); // clear the osciloscope
mycode = [ ... 
  "DATA:SOURCE CH1": ... 
  "DATA:ENCODG RIBINARY;WIDTH 1": ... 
  "HORIZONTAL:RECORDLENGTH 1000": ... 
  "DATA:START 1": ... 
  "DATA:STOP 1000": ... 
  "HEADER OFF": ... 
  "ACQUIRE:STATE RUN": ... 
  "CURVE?" ];
ibwrt ( ud3, mycode );
```

//
//
c = ibrd( ud3, 1 );  // read first byte, '#'
c = ibrd( ud3, 1 );  // read second byte, the length of a string containing the length
n = strtod( c );    // convert it to number
L = ibrd( ud3, n ); // read string containing the number of bytes to transfer
N = strtod( L );    // convert the string to the number
s = ibrd( ud3, N ); // read in N bytes, each representing a data point
x = [1:N]';
y = ascii(s)' +0;
plot([x,y]);

Note: I have written support libraries for large number of instruments. Let me know what you need...

Note: Ideally, rather than having a shared object library that user has to compile themselves, I would like to incorporate GPIB as another url, something of sort, e.g.,

gpib_url = 'gpib://0:7:0'

which is then accessed through rlab's open/readm/writem/close.
This would resolve one problem of the GPIB-linux library, which is that repeated call to ibdev for a single device creates new descriptor each time.
Part V

References
References


Part VI

Appendices
17.2 Pattern Matching
Pattern matching is a powerful tool for syntax analysis. The main idea of pattern matching comes from the SNOBOL4 language (see the wonderful book THE SNOBOL4 PROGRAMMING LANGUAGE by R. E. Griswold, J. F. Poage and I. P. Polonsky). Some of the pattern expression atoms and statements were taken from there. One can find that patterns are very similar to the Backus-Naur forms. Comparing with the regular expressions (used by grep and egrep in UNIX) patterns are more powerful, but slower in matching.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

As a special exception, if other files instantiate generics from this unit, or you link this unit with other files to produce an executable, this unit does not by itself cause the resulting executable to be covered by the GNU General Public License. This exception does not however invalidate any other reasons why the executable file might be covered by the GNU Public License.

The current version works under Windows and UNIX.

Download (match_1_1.tar.gz + gzip, Windows users may use WinZip).

1. Pattern expression

A pattern is a character string. All keywords can be written in both upper and lower cases. A pattern expression consists of atoms bound by unary and binary operators. Spaces and tabs can be used to separate keywords.

Atoms (terms of the expression):
17.2. PATTERN MATCHING

1.1. Atoms

Atoms are terms of the pattern expression. Each atom is a very simple pattern that matches something.

1.1.1. Substring atom

This atom is a substring that matches itself. Such a substring must be put in quotes ‘ ‘ or " ".
Very often it is necessary to match case independent substring. In this case the substring
must be put in <> quotes. The case of letters in <> does not matter. Empty substring is
allowed too. It matches empty chain of characters. This atom is assumed to be in any place
where a pattern must be on a syntactical reason. Examples of substring atom:

**Pattern** | **Matches**
---|---
"ABC" | ABC
'ABC' | ABC
<ABC> | ABC, ABc, AbC, Abc, aBC, aBc, abC, abc

The character circumflex (`) in the substring has a special meaning. It converts the following character by inverting bit 6 of its ASCII code. For example, ^@ means NUL, ^M means CR and so on. This allows you to include control characters and quotes in the substring body. The following table shows how circumflex works. Characters from pair columns give each other being specified with the circumflex.

| How circumflex works |
|---|---|---|---|---|---|---|
| Character | Code | Character | Code | Character | Code | Character | Code |
| @ | 0x0064 | N | 0x0652 | u | 0x0108 | 5 |
| A | 0x0065 | K | 0x010b | 6 |
| B | 0x0066 | L | 0x010c | 7 |
| C | 0x0067 | M | 0x010d | 8 |
| D | 0x0068 | N | 0x010e | 9 |
| E | 0x0069 | O | 0x010f | = |
| F | 0x006a | P | 0x0110 | > |
| G | 0x006b | Q | 0x0111 | ? |
| H | 0x006c | R | 0x0112 | A |
| I | 0x006d | S | 0x0113 | B |
| J | 0x006e | T | 0x0114 | C |
| K | 0x006f | U | 0x0115 | D |
| L | 0x0070 | V | 0x0116 | E |
| M | 0x0071 | W | 0x0117 | F |
| N | 0x0072 | X | 0x0118 | G |
| O | 0x0073 | Y | 0x0119 | H |
| P | 0x0074 | Z | 0x011a | I |
| Q | 0x0075 | | 0x011b | J |
| R | 0x0076 | | 0x011c | K |
| S | 0x0077 | | 0x011d | L |
| T | 0x0078 | | 0x011e | M |
| U | 0x0079 | | 0x011f | N |
| V | 0x007a | | 0x0120 | O |
| W | 0x007b | | 0x0121 | P |
| X | 0x007c | | 0x0122 | Q |
| Y | 0x007d | | 0x0123 | R |
| Z | 0x007e | | 0x0124 | S |

Circumflex at the end of a literal denotes itself. For instance, `^` specifies character literal containing the single character - circumflex.

**1.1.2. Atom AnyOf**

The atom matches any character from the given character set. Notation for this atom is a character string put in `{}` brackets. Order of characters in the string does not affect on the
result of matching. Circumflex within the string body works the same way as for substring atom. For example, the pattern \{.,;?!\} matches one of punctuation marks.

1.1.3. Atom END

The atom **END** (end or .) matches the line end. It fails when the rest of the current line is not empty. On success it matches empty chain of bytes.

1.1.4. Atom BREAK

The atom **BREAK** (break or _) matches non empty chain of spaces and tabs. It can match empty chain at a keyword boundary. Keyword is assumed to be a chain of letters and digits. Thus **BREAK** will fail if previous and current characters are letters or digits. This pattern is very useful to separate multiword keywords. For example: the pattern `key_' will match key in `key(12)` and key is, but never in `keyword`.

1.1.5. Other atoms

Some other atoms are described in the following table:

<table>
<thead>
<tr>
<th>Atom (notation)</th>
<th>Abbr.</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY</td>
<td>%</td>
<td>Any character</td>
</tr>
<tr>
<td>BLANK</td>
<td>+</td>
<td>Any non empty chain of spaces and tabs</td>
</tr>
<tr>
<td>DIGIT</td>
<td>#</td>
<td>One of characters 0, 1, ..., 9 (see also <strong>UserBase</strong>)</td>
</tr>
<tr>
<td>UPPER_CASE_LETTER</td>
<td>U</td>
<td>One of A, B, ..., Z</td>
</tr>
<tr>
<td>LOWER_CASE_LETTER</td>
<td>L</td>
<td>One of a, b, ..., z</td>
</tr>
<tr>
<td>LETTER</td>
<td>L</td>
<td>One of A, B, ..., Z, a, b, ..., z</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>C</td>
<td>One digit or letter</td>
</tr>
<tr>
<td>NL</td>
<td>/</td>
<td>Empty chain and skips the line (see)</td>
</tr>
<tr>
<td>FAILURE</td>
<td>F</td>
<td>Nothing and ends matching with failure</td>
</tr>
<tr>
<td>FENCE</td>
<td>:</td>
<td>Empty chain and prevents search for an alternative (see)</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>S</td>
<td>Empty chain and ends matching with success</td>
</tr>
</tbody>
</table>

1.2. Binary (infix) operations

A pattern expression is constructed with the aid of atoms and operators. There are three binary operators catenation, alternation and assignment.

1.2.1. Catenation

There is no special notation for catenation of two patterns. You should just put them in a
Concatenated patterns are matched sequentially. The whole matching is successful if and only if each of them is matched. For example:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY LETTER</td>
<td>Any character followed by a letter</td>
</tr>
</tbody>
</table>

1.2.2. Alternation

Alternatives are separated by the alternation operator, denoted as \( \text{OR} \), \( ! \) or \( | \). For example:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>'A'</td>
<td>'B'</td>
</tr>
</tbody>
</table>

Note, that the order of alternatives is very important. For instance, the pattern \( 'A' | 'AA' | 'AAA' \) will never match AAA because A is a prefix of AAA. The rule is that the pattern which is a prefix of another one must follow it in the alternation list. In our example \( 'AAA' | 'AA' | 'A' \) would be correct. Note also, that in all cases, when the AnyOf atom can be used instead of the alternation, it is preferable, because it is matched faster and the time of matching depends on neither the number of alternated characters nor their order.

1.2.3. Assignment

Assignment (=) allows one to set the value of a variable to the part of a line (lines) matched by the pattern specified as the right argument of the assignment. This feature is based on interface between the caller and the match processor (see GetVariableId, AssignVariable and DeAssignVariable).

1.3. Grouping

Round and square brackets can be used for grouping patterns. Brackets can be nested. They allow us to make more complex expressions.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>('A'</td>
<td>'B')</td>
</tr>
<tr>
<td>('A'</td>
<td>'')</td>
</tr>
<tr>
<td>('A' )</td>
<td>Same as above</td>
</tr>
<tr>
<td>('A' )</td>
<td>Same as above</td>
</tr>
</tbody>
</table>

1.4. Unary operators

There are several unary operators. All of them have prefix form, i.e. they must be specified before a pattern. If the pattern is a list of concatenated or alternated patterns it must be put
in round braces because priority of any unary operator is greater than one of \textit{catenation} or \textit{alternation}, but less than one of \textit{assignment}.

1.4.1. Repeaters

Sometimes it is necessary to repeat matching of a pattern. There are three kinds of repeaters.

The simplest repeater is \textit{finite repeater}. To repeat a pattern desired number of times, one should place the number of repetitions before the pattern. For example, \texttt{5 DIGIT} matches five digits. Note that \texttt{5('A' | 'B')} indeed matches five \texttt{A} or \texttt{B} letters, because finite repeater tries to match all available alternatives.

A more complex case is when it is unknown, how many times a pattern must be repeated. There are two indefinite repeaters \texttt{* (little repeater)} and \texttt{\$ (big repeater)}.

The \texttt{little repeater} tries to match the repeated pattern as little times as possible. So first time it does not try to match the pattern at all. Then, if a failure occurs while matching of the following pattern expression, the repeater tries to match the pattern. On success the matching process will be continued from the pattern following the repeated pattern. So each time when matching fails, the \texttt{little repeater} makes an attempt to match the repeated pattern one more time. If it is impossible to match the repeated pattern, the \texttt{little repeater} tries to diminish the number of repetitions, so that if the pattern has alternatives the repeater will try to match all of them. And only if the repetition count is zero and it is impossible to match the repeated pattern the \texttt{little repeater} fails. Therefore the expression \texttt{*P}, where \texttt{P} is a pattern may be outlined as \texttt{(|P)(|P)(|P)...} (One may say that matching of an indefinite repeater is led by its right context).

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>*LETTER '.'</td>
<td>A chain of letters followed by dot</td>
</tr>
</tbody>
</table>

In contrary to the \texttt{little repeater} the \texttt{big repeater} tries to match the repeated pattern as many times as possible. It starts with matching the repeated pattern so many times as possible. Then matching process continues to the rest of pattern expression. If it fails, the \texttt{big repeater} tries to match other alternative of the repeated pattern and then the pattern itself (as many times as possible). Only if it is impossible to find an alternative the repeater diminishes the number of repetitions. Therefore the expression \texttt{\$P}, where \texttt{P} is a pattern is an equivalent to \texttt{(P)(P)(P)...}. For example, an arbitrary length string of letters can be specified as \texttt{$LETTER}$.

1.4.2. Using atom FENCE with repeaters

All repeaters try to match all possible alternatives before failure. In many cases it is not necessary. Sometimes it is possible to optimize matching process with the aid of the atom \texttt{FENCE} (\texttt{FENCE} or \texttt{:}). In a given list of concatenated patterns \texttt{FENCE} disables any return to unmatched alternatives.
Suppose we want to build a pattern which will match a string containing a substring beginning with \texttt{A} and ending by \texttt{B}. We can write this pattern using \texttt{little repeater} and the atom \texttt{ANY} by the following way \texttt{*%'A'*%'B'}. With the string \texttt{AAACA} this pattern will work as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Unmatched</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AAACA</td>
<td>1st % matches empty substring</td>
</tr>
<tr>
<td>2</td>
<td>AAACA</td>
<td>‘A’ matches \texttt{A}</td>
</tr>
<tr>
<td>3</td>
<td>AACA</td>
<td>2nd % matches empty substring</td>
</tr>
<tr>
<td>4</td>
<td>AACA</td>
<td>‘B’ fails to match \texttt{A}</td>
</tr>
<tr>
<td>5</td>
<td>AACA</td>
<td>2nd % matches one character (A2)</td>
</tr>
<tr>
<td>6</td>
<td>ACA</td>
<td>‘B’ fails to match \texttt{A}</td>
</tr>
<tr>
<td>7</td>
<td>ACA</td>
<td>2nd % matches one more character (A3)</td>
</tr>
<tr>
<td>8</td>
<td>CA</td>
<td>‘B’ fails to match \texttt{C}</td>
</tr>
<tr>
<td>9</td>
<td>CA</td>
<td>2nd % matches one more character (C4)</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>‘B’ fails to match \texttt{A}</td>
</tr>
<tr>
<td>11</td>
<td>A</td>
<td>2nd % matches one more character (A5)</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>‘B’ fails to match nothing (end of line)</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>2nd % fails and returns AAACA back. ‘A’ returns \texttt{A}</td>
</tr>
<tr>
<td>14</td>
<td>AAACA</td>
<td>1st % matches one character (A1)</td>
</tr>
<tr>
<td>15</td>
<td>AAACA</td>
<td>‘A’ matches \texttt{A}</td>
</tr>
<tr>
<td>16</td>
<td>ACA</td>
<td>2nd % matches empty substring</td>
</tr>
<tr>
<td>17</td>
<td>ACA</td>
<td>‘B’ fails to match \texttt{A}</td>
</tr>
</tbody>
</table>

It is obvious that once \texttt{*%'B'} failed (step 13), there is no necessity to match other samples and the whole matching must fail. So we can optimize the pattern by placing \texttt{FENCE} after the first indefinite repeater \texttt{*%'A'*%'B'}. This pattern will fail after step 13.

Let us consider another example. One way of matching a chain of hexadecimal letters is $\texttt{(DIGIT|'A'|'B'|'C'|'D'|'E|')}:$. Here \texttt{FENCE} allows to avoid useless attempts to match \texttt{A} instead of a digit, \texttt{B} instead of \texttt{A} and so on. In this case the alternatives of the repeated pattern do not overlap. Hence, once an alternative fails there is no need to try others.

### 1.4.3. Ellipsis

The pattern \texttt{*%:} is used so often that it was given a special notation - ellipsis (\ldots or \ldots). Ellipsis are used to match anything till something. For example, pattern \texttt{‘A’\ldots‘B’} will match any chain which starts with \texttt{A} and ends by \texttt{B}.

### 1.4.4. NOT

The unary operator \texttt{NOT} (\texttt{not} or \texttt{^}) inverts the result of matching the pattern it precedes. Where the pattern \texttt{P} is successfully matched, \texttt{^P} pattern fails. Where \texttt{P} fails, \texttt{^P} successfully matches empty substring. So \texttt{^P} means \textit{empty if not P}. Very often doubled \texttt{NOT} is used to test if a pattern matches something doing that without side effects. For example, \texttt{LETTER:^^'*'} matches a chain of letters if and only if it is followed by \texttt{*}. But \texttt{*} itself remains unmatched.
1.4.5. NOEMPTY

The statement NOEMPTY (NOEMPTY or ?) does not allow an empty substring to be matched. For example, \$LETTER: matches only non-empty words.

1.4.6. Label

Each pattern may have one or more labels. The syntax is label>pattern. A reference to the label works as if all the patterns following the label till an unbalanced bracket ( or |) or the end of the pattern expression, would be placed instead of the reference. Note that | is just an abbreviation for |). Let the label Bool was somewhere defined as Bool>'0'|'1'. Then a reference to Bool will work exactly as '{'0'|'1'}'.

Using labels one can build recursive patterns. For instance, we can build a pattern to match a balanced chain of {} braces: balanced=('{'[balanced]'}').

1.5. Special patterns

1.5.1. NL

The atom NL (nl or /) is used to leave the current line for the next one. For example, to match an item that occupies several lines. Note the difference between END atom and NL. The atom END just matches the end of a line but does not leave it. While the atom NL leaves the current line even if it is not completely matched. This feature is based on the interface between a user written caller and the match processor (see GetNextLine and GetPreviousLine for more information).

1.5.2. User defined patterns

The match processor interface allows one to use user defined patterns as well as embedded ones. The user defined patterns are pre translated patterns which addresses are returned by the GetExternalPattern function, which binds unknown identifiers with user defined patterns.

1.6. Example

Let us build a pattern that matches an expression of C language. A C expression is something limited by semicolon, an unbalanced bracket or comma. We should take into account C comments and literals. We will build the pattern by pieces. Let us start from the pattern for string literals:

""" *( '\' | '"' | 'END/ | %): """
Note that simply `'...` would be wrong, because in C a string literal may occupy several lines. We should also recognize digraphs `\` and `\` as well.

The next pattern is for character literals (we recognize `'` literal as a separate case):

```markdown
^"|"..."$
```

The pattern for C and C++ comments looks like:

```markdown
^'/\*(\|\%/\|\%):'/\'|'/\*...\%/\$
```

The following pattern is not really necessary. It quickly skips spaces, tabs, numbers and identifiers:

```markdown
BLANK|CHARACTER$CHARACTER:
```

Now let us put all these patterns together and add **ANY** and **NL** patterns as alternatives to skip special characters and line ends:

```markdown
(Blank|Character$Character
|"|"..."
|"\"|"...\"
|"\n|"...\n"
|\*\*(END/|\%):\*/|'/\*...\%
|\%
|/
)
```

Let us repeat this pattern using **little repeater**, mark this as **Item** and add brackets recognition:

```markdown
(Item>\*

(Blank|Character$Character
|\>('Item')
|\?('Item')
|\?(Item)
|"|"..."
|\*\*(\|\%/\|\%):\*\*/\'|'/\*...\%
|\%
|/
)
```

Let us repeat this pattern using **little repeater**, mark this as **Item** and add brackets recognition:

```markdown
(Item>\*

(Blank|Character$Character
|\>('Item')
|\?('Item')
|\?(Item)
|"|"..."
|\*\*(\|\%/\|\%):\*\*/\'|'/\*...\%
|\%
|/
)
```

The condition of repetition termination is: stop if one of `,;\)\} characters appears. Here `)` is specified using **circumflex**. Doubled **NOT** is added to disable matching of the terminating