Manual for Package: mathematics
Revision 2:5M

Karl Kästner

October 29, 2019

Contents

1 mathematics

```
mathematical functions of various kind
```

1.1 cast_byte_to_integer

```
cast byte to integer
```

2 complex-analysis

```
operations on complex numbers
```

2.1 complex_exp_product_im_im

```
 product of the imaginary part of two complex exponentials

the product has two frequency components

input :
```

```
      c : complex amplitudes
      o : frequencies
output :
      cp : amplitude of the product
      op : frequencies of the product
```

2.2 complex_exp_product_im_re

```
 product of the imaginary part of one and the real part of a second
 complex exponential

the product has two frequency components

input :
      c : complex amplitudes
      o : frequencies
output :
      cp : amplitude of the product
      op : frequencies of the product
```

2.3 complex_exp_product_re_im

```
the product has two frequency components

 product of the imaginary part of one and the real part of a second
 complex exponential

input :
      c : complex amplitudes
      o : frequencies
output :
      cp : amplitude of the product
      op : frequencies of the product
```

2.4 complex_exp_product_re_re

```
 product of the real part of two complex exponentials

re(c1 exp(io1x))*re(c2 exp(io2x)) =
      1/2*(    real(c1*c2*exp(i*(n1+n2)*o*x)) ...
            + real(conj(c1)*c2*exp(i*(n2-n1)*o*x)) )
```

```
the product has two frequency components

input :
      c : complex amplitudes
      o : frequencies
output :
      cp : amplitude of the product
      op : frequencies of the product
```

2.5 croots

```
nth-roots of a complex number

input:
c : complex number
n : order of root
   n must be rational, to obtain n solutions
   otherwise no finite set of solutions exists

r : roots of the complex number
```

2.6 root_complex

```
root of a complex number
```

2.7 test_imroots

3 derivation

derivation of several functions by means of symbolic computation

3.1 derive_acfar1

3.2 derive_ar2param

3.3 derive_arc_length

3.4 derive_fourier_power

3.5 derive_fourier_power_exp

3.6 derive_laplacian_curvilinear

3.7 derive_laplacian_fourier_piecewise_linear

3.8 derive_logtripdf

3.9 derive_smooth1d_parametric

3.10 simplify_atan

```
symbolic simplification of the arcus tangent
```

4 fourier/@STFT

4.1 STFT

```
class for short time fourier transform

Note: the interval Ti should be set to at leat 2*max(T), as
    otherwise coefficients
      tend to oscillate in the presence of noise
Note: for convenience, the independent variable is labeled as time
    (t),
      but the independent variable is arbitrary, so it works
          likewise in space
```

4.2 itransform

```
inverse of the short time fourier transform
```

4.3 stft_

```
static wrapper for STFT
```

4.4 stftmat

```
transformation matrix for the short time fourier transform
```

4.5 transform

```
short time fourier transform
```

5 fourier

```
support and analysis functions both for the discrete (fast) fourier
   transform (dft/fft)
and continuous fourier analysis (fourier series)
```

5.1 amplitude_from_peak

```
amplitude and standard deviation of the amplitude of a frequency
    component
represented by a peak in the fourier domain
input :
h : peak height
w : peak width at half height

output:
a : amplitude in real space
s : standard deviation of the frequency (!)
```

5.2 dftmtx_man

```
fourier matrix in matlab style with a limited number of rows,
columns of higher frequencies are omitted

input :
n  : number of samples
nr : number of columns

output :
F : fourier matrix
```

5.3 example_fourier_window

5.4 fft_derivative

```
derivative by fourier transform
exponential convergence for periodic functions
results in spurious oscillations for aperiodic functions

input:
x : data, sampled in equal intervals
k : order of the derivative

dx : kth-derivative of x
```

5.5 fft_man

```
fast fourier transform for complex input data

input:
F : data in real space

output :

F : fourier transformation of F
```

5.6 fftsmooth

```
smooth the fourier transform and determine upper and lower bound
    confidence intervals

input :
f :
sfunc : a smoothing function (for example fir convolution with
    rectangular window)
         returns filtered (mean) value and normalized fir window
nf     : window length
nsigma : number of standard deviations for confidnce intervals


output :
ff : filtered fourier transform
l : lower bound
u : upper bound
```

5.7 fix_fourier

```
fill gaps (missing data) by means of fourier extrapolation

fix periodic data series with fourier interpolation
longest gap should not exceed 1/2 of the shortest time span of
    interest (1/cutoff frequency)
 note: this limit equals the position of first side lobe of the ft
     of a rectangular window with gap length
```

5.8 fourier_axis

```
return axis of frequencies and periods for the discrete fourier
    transform
as computed by fft (matlab-style)
```

```
input:
X : sample locations (equal interval)
L : length of samples
n : number of samples

output :
f    : frequencies
T    : periods
mask : mask for 1/2 of the fourier transform
       (as both halves are complex conjugates)
N    : frequency id
```

5.9 fourier_coefficient_piecewise_linear

```
fourier series coefficients of a piecewise linear function
(not coefficient of discrete fourier transform)
function can be discontinuous between intervals
scales domain length to 2pi

input :
l,r : end points of piecewise linear function
lval, rval : values at end points
L : length of domain
n : number of samples/highest frequency

output :
a, b : coefficients for frequency components
```

5.10 fourier_coefficient_piecewise_linear_1

```
fourier series coefficients of a piecewise linear function
(not coefficient of discrete fourier transform)
function can be discontinuous between intervals
scales domain length to 2pi

input :
X : end points of piecewise linear function
Y : values at end points

output :
ab : coefficients for frequency components
```

5.11 fourier_coefficient_ramp3

```
fourier series coefficient of a ramp
```

5.12 fourier_coefficient_ramp_pulse

```
fourier series coefficient of a ramp pules
```

5.13 fourier_coefficient_ramp_step

```
fourier coefficient of a ramp-step
```

5.14 fourier_coefficient_square_pulse

```
fourier series coefficients of a square pulse
```

5.15 fourier_derivative

```
coefficients of the derivative of a fourier series
not of discrete fourier transform (fft)
```

5.16 fourier_expand

```
expand values of fourier series
```

5.17 fourier_fit

```
fit a fourier series to a set of sample points that are not spaced
    in
equal intervals
```

5.18 fourier_interpolate

```
interpolate samples y sampled at moments (location) t to locations
    ti
```

5.19 fourier_matrix

```
transformation matrix for a continuous fourier series
(not for the discrete dft/fft)
```

5.20 fourier_matrix2

```
transformation matrix for a continuous fourier series
(not for the discrete dft/fft)
```

5.21 fourier_matrix3

```
transformation matrix for the continous fourier transform
this is a matrix with (2*n+1) real columns
```

5.22 fourier_power

```
powers of a continuous fourier series in sin/cos form

powers of a^p = (ur + u1 sin(ot) + u2 sin(ot+dp))^p
phase of first component assumed 0

frequencies higher than 2-omega ignored in input
frequencies higher than 3-omega not computed
```

5.23 fourier_power_exp

```
powers of the continuous fourier series
      a^p = (ur + u1 sin(ot) + u2 sin(ot+dp))^p
phase of first component assumed 0

higher orders than 2 ignored input
higher order than 3 not computed in output
```

```
y = a_0 + sum (a_j sin(jot) + b_j cos(jot))
  = Real(sum_{i=0}^inf c_i exp(1i*omega), c_i = a_i + b_i

NOT the alternative sum_{i=-inf}^inf \tilde c_i, tile c_j = 1/2 a_j
     + 1/2i b_j
```

5.24 fourier_predict

```
expand a continous fourier series at times t
```

5.25 fourier_range

```
approximate range of a continous Fourier series with 2 components
range(y) = max(y) - min(y)
```

5.26 fourier_regress

```
fit a continous fourier series to a set of sample points not
    sampled
at equal intervals
```

5.27 fourier_resampled_fit

```
fits coefficients of a continuous fourier transform,
but stores them as resampled values
```

5.28 fourier_resampled_predict

```
interpolates a continuous fourier series that has been stored as
    values
at their support points
```

5.29 fourier_signed_square

```
coefficients of the fourier series of | cos a + cos t | (cos a +
    cos t)
in general
          cos a is midrange
          cos t is tidal variation
c.f Dronkers
```

5.30 fourier_transform

```
continuous fourier transformation of y
(not discrete fourier transformation dft/fft)

input:
      b : data sampled at equal intervals
      T : length of data in time or space, i.e. position of last
          sample if
          position of first sample is 0
     T_max : maximum period to include

output :
      A  : fourier matrix
     p  : fourier transformation of b
      tt : TODO
```

5.31 hyperbolic_fourier_box

5.32 idftmtx_man

```
inverse matrix for the discrete fourier transform in matlab style
with a limited number of columns, thus ignoring higher frequencies
keep 2nc+1 columns (mean and conj-complex pairs of nc frequencies)
```

5.33 laplace_2d_pwlinear

```
solution to the Laplacian in two dimensions for a finite
    rectangular domain
with piecewise constant boundary conditions
```

```
linear system with 4 unknowns per freqency component
these are coefficients of s,c,sh,ch
      (pu*(s + c) + qu*(s' + c'))*(shu + chu) = ru      % upper bc
      (pd*(s + c) + qd*(s' + c'))*(shd + chd) = rd      % lower bc
      ( (sl + cl)*( pl*(shl + chl) + ql*(shl' + chl')) = rl % left
          bc
      ( (sr + cr)*( pr*(shr + chr) + qr*(shr' + chr')) = rr % right
           bc

 least squares with piecewise integration
[x0,p,q,r] piecewise linear polynomials at the boundaries
```

5.34 nanfft

```
discrete fourier transform of a data series with gaps
```

5.35 peaks

```
peaks of the power spectrum of a disctrete fourier transform

rule for peaks: there is no higher value left or right of the "peak
    "
              until the signal drops to p*y_peak, p = 0.5

works best, when spectrum has been smoothened

input :
f : frequency
y : absolute value of fourier transform (power spectrum)
L : length in space or time of series

output :

a0 : amplitude
s0 : standard deviation (error?) of amplitude
w0 : width of peak
lambda = wave length (period?)
pdx : index of peak
f : frequency (if not given as input)
```

5.36 roots_fourier

```
zeros of continuous fourier series series

      f  = a_0 + sum_j=^n a_i cos(j x) + b_i sin(j x)
```

5.37 spectral_density

```
spectral density
```

5.38 test_complex_exp_product

5.39 test_idftmtx

6 geometry/@Geometry

6.1 Geometry

6.2 arclength

```
arc length of a two dimensional curve

8th order accurate
does not require the segments length to vary smoothly

note: the curve can be considered parametric, e.g. x = x(t), y=y(t)
     and
      and t = t(s), but the error term contains derivatives of t,
     thus a non smooth t (strongly varying distance between points)
     requires the scaling as done below
```

6.3 arclength_old

```
arc length of a two dimensional function
```

6.4 arclength_old2

```
arc length of a two dimensional function
```

6.5 base_point

```
base point (fusspunkt), i.e. point on a line with shortest distance
to another point
```

6.6 base_point_limited

```
base point (Fusspunkt) of a point on a line
```

6.7 centroid

```
centroid pf a polygone
```

6.8 cosa_min_max

6.9 cross2

```
cross product in two dimensions
```

6.10 curvature

```
curvature of a function in two dimensions
```

6.11 ddot

```
sum of squares of cos of inner angles of triangle
```

6.12 distance

```
equclidan distance between two points
```

6.13 distance2

```
euclidean distance between two points
this function requires a and be of equal dimensions, or the least
    the first pair or second pair to be a scalar
```

6.14 dot

```
dot product
```

6.15 edge_length

```
edge length
```

6.16 enclosed_angle

```
angle enclosed between two lines
```

6.17 enclosing_triangle

```
smallest enclosing equilateral triangle with bottom site paralle to
     X axis
```

6.18 hexagon

```
coordinates of a hexagon, scaled and rotated
```

6.19 inPolygon

```
flag points contained in a polygon
much faster than matlab internal function
```

6.20 inTetra

```
flag points contained in tetrahedron
```

6.21 inTetra2

```
flag points contained in tetrahedron
```

6.22 inTriangle

```
flag points contained in triangle
```

6.23 intersect

```
intersect between two lines
```

6.24 lineintersect

```
intersect of two lines
```

6.25 lineintersect1

```
intersect of two lines
```

6.26 minimum_distance_lines

```
minimum distance of two lines in three dimensions
```

6.27 mittenpunkt

```
mittenpunkt of a triangle
```

6.28 nagelpoint

```
nagelpoint of a triangle
```

6.29 onLine

6.30 orthocentre

```
orthocentre of triangle
```

6.31 plumb_line

6.32 poly_area

```
area of a polygon
```

6.33 poly_edges

```
edges of a polygon
```

6.34 poly_set

```
associate point at arbitary location with a polygon it is contained
     in
and assign the value of the polygon to it
```

6.35 poly_width

```
width of polygon width holes by surface normals
holes / islands separated with NaN
order of points of outer boundary must be cw
order of points of holes must be ccw
note that this function does not give the true width for expanding
    sections
use voronoi polygons for this
```

6.36 polyxpoly

```
intersections of two polygons
```

6.37 project_to_curve

```
closest point on a curve with respect to a point at distance to the
     curve
```

6.38 random_disk

```
draw random points on the unit disk
```

6.39 random_simplex

```
random point inside of a triangle
```

6.40 sphere_volume

```
volume of a sphere
```

6.41 tetra_volume

```
volume of a tetrahedron
```

6.42 tobarycentric

```
cartesian to barycentric coordinates
```

6.43 tobarycentric1

```
cartesian to barycentric coordinates
```

6.44 tobarycentric2

```
cartesian to barycentric coordinates
```

6.45 tobarycentric3

```
cartesian to barycentric coordinates
```

6.46 tri_angle

```
cos of angles of a triangle
```

6.47 tri_area

```
angle of a triangle
```

6.48 tri_centroid

```
centroid of a triangle
```

6.49 tri_distance_opposit_midpoint

```
distance between corner of a triangle and its opposing mid-point
```

6.50 tri_edge_length

```
edge length of a triangle
```

6.51 tri_edge_midpoint

```
mid point of a triangle
```

6.52 tri_excircle

```
excircle of a triangle
```

6.53 tri_height

```
height of a triangle
```

6.54 tri_incircle

```
incircle of a triangle
```

6.55 tri_isacute

```
flag acute triangles
```

6.56 tri_isobtuse

```
flag obntuse triangles
```

6.57 tri_semiperimeter

```
semiperimeter of a triangle
```

6.58 tri_side_length

```
edge lenght of triangle
```

7 geometry

7.1 Polygon

```
Simple 2D polygon class

  Polygon properties:
      x - x coordinates of polygon
      y - y coordinates of polygon
      nnodes - number of nodes in the polygon

  Polygon methods:
      in - checks whether given points lie inside, on the edge, or
          outside of the polygon
      area - returns the area of the polygon
      centerline - computes the centerline of the river
      iscw - check whether polygon is clockwise
      reverse - reverse the order of the polygon
```

7.2 bounding_box

```
bounding box of X
```

7.3 curvature_1d

```
curvature of a sampled parametric curve in two dimensions
```

7.4 cvt

```
centroidal voronoi tesselation
```

7.5 deg_to_frac

```
degree, minutes and seconds to fractions
```

7.6 ellipse

```
n-points on an ellipse
```

7.7 ellipseX

```
x-coordinates of y-coordinates of an ellipse
```

7.8 ellipseY

7.9 first_intersect

```
get first intersection between lines in A and B
```

7.10 golden_ratio

```
golden ratio
```

7.11 hypot3

```
hypothenuse in 3D
```

7.12 meanangle

```
weighted mean of angles
```

7.13 meanangle2

```
mean angle
```

7.14 meanangle3

```
mean angle
```

7.15 meanangle4

```
mean angle
```

7.16 medianangle

```
median angle
angle, that has the smallest squared distance to all others
```

7.17 medianangle2

```
median angle

input
alpha : x*m, [rad] angle

ouput
ma    : 1*m, [rad] median angle
sa    : 1*m, [rad] standard error of median angle for uncorrelated
    error
```

7.18 pilim

```
limit to +- pi
```

7.19 streamline_radius_of_curvature

```
streamline radius of curvature
simplifies when rotatate to streamwise coordinates to R = 1/dv/ds *
     u
```

8 linear-algebra

8.1 averaging_matrix_2

8.2 colnorm

```
norms of columns
```

8.3 condest_

```
estimation of the condition number
```

9 linear-algebra/coordinate-transformation

9.1 barycentric2cartesian

```
barycentric to cartesian coordinates
```

9.2 barycentric2cartesian3

```
convert barycentric to cartesian coordinates
```

9.3 cartesian2barycentric

```
cartesian to barycentric coordinates
```

9.4 cartesian_to_unit_triangle_basis

```
transform coodinates into unit triangle
```

9.5 example_approximate_utm_conversion

9.6 latlon2utm

transform latitude and longitude to WGS84 UTM

9.7 latlon2utm_simple

9.8 lowrance_mercator_to_wgs84

convert lowrance coordinates to wgs84

based on spreadsheet by D Whitney King and Patty B at Lowrance

9.9 nmea2utm

convert nmea messages to utm coordinates

9.10 sn2xy

convert sn to xy coordinates

9.11 unit_triangle_to_cartesian

transform coordinates in unit triangle to cartesian coordinates

9.12 utm2latlon

convert wgs84 utm to latitute and longitude

9.13 xy2nt

```
project all points onto the cross section and assign them nz-
    coordinates

transform coordinate into N-T reference
rotate coordinate, so that cross section goes along x-axis
then x and y are n and t respectively scaled by width
N and T coordinates
```

9.14 xy2sn

```
convert cartesian to streamwise coordiantes
```

9.15 xy2sn_java

```
use java port for speed up
```

9.16 xy2sn_old

```
transform points from cartesian into streamwise coordinates

NOTE : prefer the java version, this has some problems with round
    off
```

10 linear-algebra

10.1 det2x2

```
2x2 matrix inverse of 2x2 matrices stacked along dim 3
```

10.2 det3x3

```
determinant of stacked 3x3 matrices
```

10.3 det4x4

```
determinant of stacked 4x4 matrices
```

10.4 diag2x2

```
diagonal of stacked 2x2 matrices
```

10.5 eig2x2

```
eigenvalues of stacked 2x2 matrices
```

10.6 first

10.7 gershgorin_circle

```
range of eigenvalues determined by the gershgorin circle theorem
```

10.8 haussdorff

```
haussdorf dimension
box counting: count cectangles passed through by line (covered by
    polygon)

Koch snow flake 3:4 -> 1.2619
Kantor set     2:3, (4:9) -> 0.6309
quadrat        4:2, 9:3, 16:4 -> 2
```

10.9 ieig2x2

```
reconstruct matrix from eigenvalue decomposition
```

10.10 inv2x2

```
2x2 inverse of stacked matrices
```

10.11 inv3x3

10.12 inv4x4

```
inverse of stacked 4x4 matrices
```

10.13 lpmean

```
mean of pth-power of a
```

10.14 lpnorm

```
norm of lth-power of a
```

10.15 matvec3

```
matrix-vector product of stacked matrices and vectors
```

10.16 max2d

```
maximum value and i-j index for matrix
```

10.17 mpoweri

```
approximation of A^p, where p is not integer by quadtratic
    interpolation
```

10.18 mtimes2x2

10.19 mtimes3x3

```
product of stacked 3x3 matrices
```

10.20 nannorm

```
norm of a vector, skips nan-values
```

10.21 nanshift

```
shift vector, but set out of range values to NaN
```

10.22 nl

```
number rows (lines) of a matrix

analogue to unix nl command
```

10.23 normalise

```
normalise a vector or the columns of a matrix
note that the columns are independently normalised, and hence not
    necessarily
orthogonal to each other use the gram schmidt algorithm for this (
    qr or orth)
```

10.24 normalize1

```
normalize columns in x to [-1,1]
```

10.25 normrows

10.26 orth2

```
make matrix A orhogonal to B
```

10.27 orth_man

```
orthogonalize the columns of A
```

10.28 orthogonalise

```
make x orthogonal to Y
```

10.29 paddext

```
padd values to vactor
not suitable for noisy data
order = 0 : constant extrapolation (hold)
order = 1 : linear extrapolation
```

10.30 paddval1

```
padd values at end of x
```

10.31 paddval2

```
padd values to x
```

11 linear-algebra/polynomial

11.1 chebychev

```
chebycheff polynomials
```

11.2 piecewise_polynomial

```
evaluate piecewise polynomial
```

11.3 roots1

```
roots of linear functions
```

11.4 roots2

```
roots of quadratic function
c1 x^2 + c2 x + c3 = 0
```

11.5 vanderi_1d

```
vandermonde matrix of an integral
```

11.6 vandermonde

```
van der monde matrix
```

12 linear-algebra

12.1 randrot

```
random rotation matrix
```

12.2 right

```
get right column by shifting columns to left
extrapolate rightmost column
```

12.3 rot2

```
rotation matrix from angle
```

12.4 rot2dir

```
rotation matrix from direction vector
```

12.5 rot3

12.6 rownorm

12.7 simmilarity_matrix

12.8 spnorm

```
frobenius norm
```

12.9 spzeros

```
allocate a sparze matrix of zeros
```

12.10 transpose3

```
transpose stacked 3x3 matrices
```

12.11 transposeall

13 logic

```
bitwise operations on integers
```

13.1 bitor_man

```
bitwise OR of the numbers of the columns of A

input:
      A (positive integer)
```

14 master/derive

14.1 derive_bc_one_sided

14.2 derive_convergence

14.3 derive_error_fdm

14.4 derive_fdm_vargrid

14.5 derive_fem_2d_mass

14.6 derive_fem_error_2d

14.7 derive_fem_error_3d

14.8 derive_grid_constants

14.9 derive_interpolation

14.10 derive_laplacian

14.11 derive_limit

14.12 derive_nc_1d

14.13 derive_nc_1d_

14.14 derive_nc_2d

14.15 derive_nonuniform_symmetric

%

14.16 derive_poly

14.17 derive_power

14.18 derive_richardson

14.19 derive_sum

14.20 derive_sym_fem_2d

14.21 derive_taylor

14.22 nn

14.23 t

14.24 test_derive

14.25 test_filter

14.26 test_vargrid

15 master/eigenvalue

15.1 eig_bisection

15.2 eig_inverse_iteration

15.3 eig_power_iteration

16 master/eigenvalue/jacobi-davidson/JDQR

16.1 Example1

16.2 Example2

```
% dimension of the matrix operation
```

16.3 ILU

16.4 jdqr

```
% Read/set parameters
% Initiate global variables
% Return if eigenvalueproblem is trivial
% Initialize V, W:
%  V,W orthonormal, A*V=W*R+Qschur*E, R upper triangular
% The JD loop (Standard)
%   V orthogonal, V orthogonal to Qschur
%   V*V=eye(j), Qschur'*V=0,
%   W=A*V, M=V'*W
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%   Both V and W orthonormal and orthogonal w.r.t. Qschur
%   V*V=eye(j), Qschur'*V=0, W'*W=eye(j), Qschur'*W=0
%   (A*V-tau*V)=W*R+Qschur*E, E=Qschur'*(A*V-tau*V), M=W'*V
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    V W AV.
%    Both V and W orthonormal and orthogonal w.r.t. Qschur, AV=A*V-
    tau*V
%    V*V=eye(j), W'*W=eye(j), Qschur'*V=0, Qschur'*W=0,
%    (I-Qschur*Qschur')*AV=W*R, M=W'*V; R=W'*AV;
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    W orthonormal, V and W orthogonal to Qschur,
%    W'*W=eye(j), Qschur'*V=0, Qschur'*W=0
%    W=(A*V-tau*V)-Qschur*E, E=Qschur'*(A*V-tau*V),
%    M=W'*V
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
 W=V*Q; V=V(:,1:j)/R; E=E/R; R=eye(j); M=Q(1:j,:)'/R;
 W=V*H; V(:,j+1)=[];R=R'*R; M=H(1:j,:)';
%======== ARNOLDI (for initializing spaces)
    ===============================
%=========== END ARNOLDI
    ============================================
% not accurate enough M=Rw'\(M/Rv);
%=========== COMPUTE SORTED JORDAN FORM
    ==================================
% accepted separation between eigenvalues:
% no preconditioning
% solve left preconditioned system
% compute vectors and matrices for skew projection
% precondion and project r
% solve preconditioned system
% no preconditioning
% solve two-sided expl. precond. system
% compute vectors and matrices for skew projection
% precondion and project r
% solve preconditioned system
% "unprecondition" solution
%%%% u(:,j+1)=Atilde*u(:,j)
%%%% r(:,j+1)=Atilde*r(:,j)
%------ compute schur form -------------
 A*Q=Q*S, Q'*Q=eye(size(A));
% transform real schur form to complex schur form
%------ find order eigenvalues ---------------
%------ reorder schur form ----------------
%------ compute qz form ----------------
%------ sort eigenvalues ---------------
%------ sort qz form ------------------
% i>j, move ith eigenvalue to position j
% determine dimension
% defaults
```

16.5 testA

16.6 testB

```
1
```

17 master/eigenvalue/jacobi-davidson

17.1 afun_jdm

17.2 davidson

17.3 jacobi_davidson

17.4 jacobi_davidson_qr

17.5 jacobi_davidson_qz

17.6 jacobi_davidson_simple

17.7 jdqr

```
% Read/set parameters
% Initiate global variables
% Return if eigenvalueproblem is trivial
% Initialize V, W:
%  V,W orthonormal, A*V=W*R+Qschur*E, R upper triangular
% The JD loop (Standard)
%   V orthogonal, V orthogonal to Qschur
%   V*V=eye(j), Qschur'*V=0,
%   W=A*V, M=V'*W
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%   Both V and W orthonormal and orthogonal w.r.t. Qschur
%   V*V=eye(j), Qschur'*V=0, W'*W=eye(j), Qschur'*W=0
%   (A*V-tau*V)=W*R+Qschur*E, E=Qschur'*(A*V-tau*V), M=W'*V
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    V W AV.
%    Both V and W orthonormal and orthogonal w.r.t. Qschur, AV=A*V-
    tau*V
%    V*V=eye(j), W'*W=eye(j), Qschur'*V=0, Qschur'*W=0,
%    (I-Qschur*Qschur')*AV=W*R, M=W'*V; R=W'*AV;
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    W orthonormal, V and W orthogonal to Qschur,
%    W'*W=eye(j), Qschur'*V=0, Qschur'*W=0
%    W=(A*V-tau*V)-Qschur*E, E=Qschur'*(A*V-tau*V),
%    M=W'*V
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
 W=V*Q; V=V(:,1:j)/R; E=E/R; R=eye(j); M=Q(1:j,:)'/R;
 W=V*H; V(:,j+1)=[];R=R'*R; M=H(1:j,:)';
%======== ARNOLDI (for initializing spaces)
    ===============================
%=========== END ARNOLDI
    ============================================
% not accurate enough M=Rw'\(M/Rv);
%=========== COMPUTE SORTED JORDAN FORM
    ==================================
% compute vectors and matrices for skew projection
% solve preconditioned system
% 0 step of bicgstab eq. 1 step of bicgstab
% Then x is a multiple of b
% HIST=[0,1];
 explicit preconditioning
% compute norm in l-space
%  HIST=[HIST;[nmv,rnrm/snrm]];
% sufficient accuracy. No need to update r,u
 implicit preconditioning
% collect the updates for x in l-space
% but, do the orth to Q implicitly
% compute norm in l-space
% HIST=[HIST;[nmv,rnrm/snrm]];
% sufficient accuracy. No need to update r,u
% Do the orth to Q explicitly
% In exact arithmetic not needed, but
% appears to be more stable.
%  plot(HIST(:,1),log10(HIST(:,2)+eps),'*'), drawnow, pause
% 0 step of gmres eq. 1 step of gmres
% Then x is a multiple of b
%=======================================================================

% 0 step of gmres eq. 1 step of gmres
% Then x is a multiple of b
 HIST=1;
% Lucky break-down
 HIST=[HIST;(gamma~=0)/sqrt(rho)];
% Lucky break-down
% solve in least square sense
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
 plot(J,HIST(:,1),'*'); drawnow,% pause
 r=r/rho; rho=1;
%  HIST=rho;
%  HIST=[HIST;rho];
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
```

```
 plot(J,HIST(:,1),'*'); drawnow,% pause
% HIST = rho;
%  HIST=[HIST;rho];
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
 plot(J,HIST(:,1),'*'); drawnow, pause
% HIST = rho;
%  HIST=[HIST;rho];
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
 plot(J,HIST(:,1),'*'); drawnow, pause
%------ compute schur form -------------
 A*Q=Q*S, Q'*Q=eye(size(A));
% transform real schur form to complex schur form
%------ find order eigenvalues ---------------
%------ reorder schur form ----------------
%------ compute qz form ----------------
%------ sort eigenvalues ---------------
%------ sort qz form -------------------
% i>j, move ith eigenvalue to position j
% determine dimension
% defaults
%% 'v'
```

17.8 jdqr_sleijpen

```
% Read/set parameters
% Initiate global variables
% Return if eigenvalueproblem is trivial
% Initialize V, W:
%   V,W orthonormal, A*V=W*R+Qschur*E, R upper triangular
% The JD loop (Standard)
%    V orthogonal, V orthogonal to Qschur
%    V*V=eye(j), Qschur'*V=0,
%    W=A*V, M=V'*W
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    Both V and W orthonormal and orthogonal w.r.t. Qschur
%    V*V=eye(j), Qschur'*V=0, W'*W=eye(j), Qschur'*W=0
%    (A*V-tau*V)=W*R+Qschur*E, E=Qschur'*(A*V-tau*V), M=W'*V
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    V W AV.
%    Both V and W orthonormal and orthogonal w.r.t. Qschur, AV=A*V-
    tau*V
%    V*V=eye(j), W'*W=eye(j), Qschur'*V=0, Qschur'*W=0,
%    (I-Qschur*Qschur')*AV=W*R, M=W'*V; R=W'*AV;
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    W orthonormal, V and W orthogonal to Qschur,
%    W'*W=eye(j), Qschur'*V=0, Qschur'*W=0
%    W=(A*V-tau*V)-Qschur*E, E=Qschur'*(A*V-tau*V),
%    M=W'*V
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
 W=V*Q; V=V(:,1:j)/R; E=E/R; R=eye(j); M=Q(1:j,:)'/R;
 W=V*H; V(:,j+1)=[];R=R'*R; M=H(1:j,:)';
%======== ARNOLDI (for initializing spaces)
    ===============================
%=========== END ARNOLDI
    ============================================
% not accurate enough M=Rw'\(M/Rv);
%=========== COMPUTE SORTED JORDAN FORM
    ==================================
% compute vectors and matrices for skew projection
% solve preconditioned system
% 0 step of bicgstab eq. 1 step of bicgstab
% Then x is a multiple of b
% HIST=[0,1];
 explicit preconditioning
% compute norm in l-space
%  HIST=[HIST;[nmv,rnrm/snrm]];
% sufficient accuracy. No need to update r,u
 implicit preconditioning
% collect the updates for x in l-space
% but, do the orth to Q implicitly
% compute norm in l-space
```

```
% HIST=[HIST;[nmv,rnrm/snrm]];
% sufficient accuracy. No need to update r,u
% Do the orth to Q explicitly
% In exact arithmetic not needed, but
% appears to be more stable.
%  plot(HIST(:,1),log10(HIST(:,2)+eps),'*'), drawnow, pause
% 0 step of gmres eq. 1 step of gmres
% Then x is a multiple of b
%=====================================================================

% 0 step of gmres eq. 1 step of gmres
% Then x is a multiple of b
 HIST=1;
% Lucky break-down
 HIST=[HIST;(gamma~=0)/sqrt(rho)];
% Lucky break-down
% solve in least square sense
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
 plot(J,HIST(:,1),'*'); drawnow,% pause
 r=r/rho; rho=1;
%  HIST=rho;
%  HIST=[HIST;rho];
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
 plot(J,HIST(:,1),'*'); drawnow,% pause
% HIST = rho;
%  HIST=[HIST;rho];
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
 plot(J,HIST(:,1),'*'); drawnow, pause
% HIST = rho;
%  HIST=[HIST;rho];
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
 plot(J,HIST(:,1),'*'); drawnow, pause
%------ compute schur form -------------
 A*Q=Q*S, Q'*Q=eye(size(A));
% transform real schur form to complex schur form
%------ find order eigenvalues ---------------
%------ reorder schur form ----------------
%------ compute qz form ----------------
%------ sort eigenvalues ---------------
%------ sort qz form -------------------
% i>j, move ith eigenvalue to position j
% determine dimension
% defaults
%% 'v'
```

17.9 jdqr_vorst

```
% Read/set parameters
% Initiate global variables
% Return if eigenvalueproblem is trivial
% Initialize V, W:
%   V,W orthonormal, A*V=W*R+Qschur*E, R upper triangular
% The JD loop (Standard)
%    V orthogonal, V orthogonal to Qschur
%    V*V=eye(j), Qschur'*V=0,
%    W=A*V, M=V'*W
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    Both V and W orthonormal and orthogonal w.r.t. Qschur
%    V*V=eye(j), Qschur'*V=0, W'*W=eye(j), Qschur'*W=0
%    (A*V-tau*V)=W*R+Qschur*E, E=Qschur'*(A*V-tau*V), M=W'*V
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
```

```
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    V W AV.
%    Both V and W orthonormal and orthogonal w.r.t. Qschur, AV=A*V-
    tau*V
%    V*V=eye(j), W'*W=eye(j), Qschur'*V=0, Qschur'*W=0,
%    (I-Qschur*Qschur')*AV=W*R, M=W'*V; R=W'*AV;
%
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
% Expand the subspaces of the interaction matrix
% The JD loop (Harmonic Ritz values)
%    W orthonormal, V and W orthogonal to Qschur,
%    W'*W=eye(j), Qschur'*V=0, Qschur'*W=0
%    W=(A*V-tau*V)-Qschur*E, E=Qschur'*(A*V-tau*V),
%    M=W'*V
% Compute approximate eigenpair and residual
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for convergence
% Expand the partial Schur form
 Rschur=[[Rschur;zeros(1,k)],Qschur'*MV(u)]; k=k+1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Expand preconditioned Schur matrix PinvQ
% Check for shrinking the search subspace
% Solve correction equation
```

```
% Expand the subspaces of the interaction matrix
 W=V*Q; V=V(:,1:j)/R; E=E/R; R=eye(j); M=Q(1:j,:)'/R;
 W=V*H; V(:,j+1)=[];R=R'*R; M=H(1:j,:)';
%======== ARNOLDI (for initializing spaces)
    ===============================
%=========== END ARNOLDI
    ============================================
% not accurate enough M=Rw'\(M/Rv);
%=========== COMPUTE SORTED JORDAN FORM
    ==================================
% accepted separation between eigenvalues:
% no preconditioning
% solve left preconditioned system
% compute vectors and matrices for skew projection
% precondion and project r
% solve preconditioned system
% no preconditioning
% solve two-sided expl. precond. system
% compute vectors and matrices for skew projection
% precondion and project r
% solve preconditioned system
% "unprecondition" solution
%%%% u(:,j+1)=Atilde*u(:,j)
%%%% r(:,j+1)=Atilde*r(:,j)
%------ compute schur form -------------
 A*Q=Q*S, Q'*Q=eye(size(A));
% transform real schur form to complex schur form
%------ find order eigenvalues ---------------
%------ reorder schur form ----------------
%------ compute qz form ----------------
%------ sort eigenvalues ---------------
%------ sort qz form -------------------
% i>j, move ith eigenvalue to position j
% determine dimension
% defaults
```

17.10 jdqz

```
% Read/set parameters
% Return if eigenvalueproblem is trivial
% Initialize target, test space and interaction matrices
% V=RepGS(Qschur,V); [AV,BV]=MV(V); %%% more stability??
% W=RepGS(Zschur,eval(testspace)); %%% dangerous if sigma~lambda
% Solve the preconditioned correction equation
% Expand the subspaces and the interaction matrices
% Check for stagnation
% Solve projected eigenproblem
```

```
% Compute approximate eigenpair and residual
%=== an alternative, but less stable way of computing z =====
% display history
% save history
% check convergence
% EXPAND Schur form
% Expand preconditioned Schur matrix MinvZ=M\Zschur
% check for conjugate pair
% To detect whether another eigenpair is accurate enough
% restart if dim(V)> jmax
% Initialize target, test space and interaction matrices
% additional stabilisation. May not be needed
% V=RepGS(Zschur,V); [AV,BV]=MV(V);
% end add. stab.
% Solve the preconditioned correction equation
% expand the subspaces and the interaction matrices
% Check for stagnation
% compute approximate eigenpair
% Compute approximate eigenpair and residual
% display history
% save history
% check convergence
% expand Schur form
% ZastQ=Z'*Q0
% the final Qschur
% check for conjugate pair
% t perp Zschur, t in span(Q0,imag(q))
% To detect whether another eigenpair is accurate enough
% restart if dim(V)> jmax
%======== END JDQZ
    ====================================================
%======================================================================

%======== PREPROCESSING
    ===============================================
%======================================================================

%======== ARNOLDI (for initial spaces)
    ================================
%% then precond=I and target = 0: apply Arnoldi with A
%======== END ARNOLDI
    =================================================
%======================================================================

%======== POSTPROCESSING
    ==============================================
%======================================================================
```

```
%======== SORT QZ DECOMPOSITION INTERACTION MATRICES
    ==================
%======== COMPUTE SORTED JORDAN FORM
    ===================================
%======== END JORDAN FORM
    ==============================================
%======== OUTPUT
    =======================================================
%======================================================================

%======== UPDATE PRECONDITIONED SCHUR VECTORS
    =========================
%======================================================================

%======================================================================

%======== SOLVE CORRECTION EQUATION
    ===================================
%======================================================================

% solve preconditioned system
%======================================================================

%======== LINEAR SOLVERS
    ===============================================
%======================================================================

% [At,Bt]=MV(x); At=theta(2)*At-theta(1)*Bt;
% xtol=norm(r-At+Z*(Z'*At))/norm(r);
%===== Iterative methods
    ===============================================
% 0 step of bicgstab eq. 1 step of bicgstab
% Then x is a multiple of b
% HIST=[0,1];
 explicit preconditioning
% compute norm in l-space
% HIST=[HIST;[nmv,rnrm/snrm]];
% sufficient accuracy. No need to update r,u
 implicit preconditioning
% collect the updates for x in l-space
% but, do the orth to Z implicitly
% compute norm in l-space
% HIST=[HIST;[nmv,rnrm/snrm]];
% sufficient accuracy. No need to update r,u
% Do the orth to Z explicitly
% In exact arithmetic not needed, but
% appears to be more stable.
% plot(HIST(:,1),log10(HIST(:,2)+eps),'*'), drawnow
% 0 step of gmres eq. 1 step of gmres
```

```
% Then x is a multiple of b
%============================================================================

% 0 step of gmres eq. 1 step of gmres
% Then x is a multiple of b
 HIST=1;
% Lucky break-down
 HIST=[HIST;(gamma~=0)/sqrt(rho)];
% Lucky break-down
% solve in least square sense
 HIST=log10(HIST+eps); J=[0:size(HIST,1)-1]';
 plot(J,HIST(:,1),'*'); drawnow
%======== END SOLVE CORRECTION EQUATION
    ===============================
%============================================================================

%======== BASIC OPERATIONS
    ============================================
%============================================================================

 y(1:5,1), pause
%======== COMPUTE r AND z
    =============================================
% E*u=Q*sigma, sigma(1,1)>sigma(2,2)
%======== END computation r and z
    =====================================
%============================================================================

%======== Orthogonalisation
    ===========================================
%============================================================================

%======== END Orthogonalisation
    ======================================
%============================================================================

%======== Sorts Schur form
    ============================================
%============================================================================

   kappa=max(norm(A,inf)/max(norm(B,inf),1.e-12),1);
   kappa=2^(round(log2(kappa)));
%------ compute the qz factorization -------
%------ scale the eigenvalues --------------
%------ sort the eigenvalues ---------------
%------ swap the qz form -------------------
% repeat SwapQZ if angle is too small
%============================================================================
```

```
%=========================================================================

% i>j, move ith eigenvalue to position j
% compute q s.t. C*q=(t(i,1)*S-s(i,1)*T)*q=0
 C*P=Q*R
 check whether last but one diag. elt r nonzero
 C*q
% end computation q
%======== END sort QZ decomposition interaction matrices
    ==============
%=========================================================================

%======== INITIALIZATION
    ==============================================
%=========================================================================

%=========================================================================

% defaults            %%%% search for 'xx' in fieldnames
%% 'ma'
%% 'sch'
%% 'to'
%% 'di'
% jmin=nselect+p0 %%%% 'jmi'
% jmax=jmin+p1  %%%% 'jma'
%% 'te'
%% 'pai'
%% 'av'
%% 'tr'
%% 'fix'
%% 'ns'
%% 'ch'
%% 'lso'
%% 'ls_m'
%% 'ls_t'
%% 'ls_e'
%% 'ty'
%% 'l_'
%% 'u_'
%% 'p_'
%% 'sca'
%% 'v0'
 initiation
 'standard'
 'harmonic'
 'searchspace'
%=========================================================================

% or Operator_Form=3 or Operator_Form=5???
```

```
%==========================================================================

%========= DISPLAY FUNCTIONS
    ============================================
%==========================================================================

%==========================================================================

%==========================================================================

%==========================================================================
```

17.11 mfunc_jdm

17.12 mgs

17.13 minres_

17.14 mv_jacobi_davidson

18 master/fdm

18.1 fdm_adaptive_grid

18.2 fdm_adaptive_refinement_old

18.3 fdm_assemble_d1_2d

18.4 fdm_assemble_d2_2d

18.5 fdm_confinement

18.6 fdm_d_vargrid

18.7 fdm_h_unstructured

18.8 fdm_hydrogen_vargrid

18.9 fdm_mark_unstructured_2d

18.10 fdm_plot

18.11 fdm_plot_series

18.12 fdm_refine_2d

18.13 fdm_refine_3d

18.14 fdm_refine_unstructured_2d

18.15 fdm_schroedinger_2d

18.16 fdm_schroedinger_3d

18.17 relocate

19 master/fem

19.1 assemble_1d_dphi_dphi

19.2 assemble_1d_phi_phi

19.3 assemble_2d_dphi_dphi_java

19.4 assemble_2d_phi_phi_java

19.5 assemble_3d_dphi_dphi_java

19.6 assemble_3d_phi_phi_java

19.7 boundary_1d

19.8 boundary_2d

19.9 boundary_3d

19.10 check_area_2d

19.11 cropradius

19.12 derivative_1d

19.13 display_2d

19.14 display_3d

19.15 distort

19.16 err_2d

19.17 estimate_err_2d_3

19.18 example_1d

19.19 example_2d

19.20 explode

19.21 fem_2d

19.22 fem_2d_heuristic_mesh

19.23 fem_plot_1d

19.24 fem_plot_1d_series

19.25 fem_plot_2d

19.26 fem_plot_2d_series

19.27 fem_plot_3d

19.28 fem_plot_3d_series

19.29 fem_plot_confine_series

19.30 fem_radial

```
adaptive grid
constant grid
```

19.31 flip_2d

19.32 get_mesh_arrays

19.33 hashkey

20 master/fem/int

20.1 int_1d_gauss_1

20.2 int_1d_gauss_2

20.3 int_1d_gauss_3

20.4 int_1d_gauss_4

20.5 int_1d_gauss_5

20.6 int_1d_gauss_6

20.7 int_1d_nc_2

20.8 int_1d_nc_3

20.9 int_1d_nc_4

20.10 int_1d_nc_5

20.11 int_1d_nc_6

20.12 int_2d_gauss_1

20.13 int_2d_gauss_12

20.14 int_2d_gauss_13

20.15 int_2d_gauss_16

20.16 int_2d_gauss_25

20.17 int_2d_gauss_3

20.18 int_2d_gauss_33

20.19 int_2d_gauss_6

20.20 int_2d_gauss_7

20.21 int_2d_gauss_9

20.22 int_2d_nc_10

20.23 int_2d_nc_15

20.24 int_2d_nc_21

20.25 int_2d_nc_3

20.26 int_2d_nc_6

20.27 int_3d_gauss_1

20.28 int_3d_gauss_11

20.29 int_3d_gauss_14

20.30 int_3d_gauss_15

20.31 int_3d_gauss_24

20.32 int_3d_gauss_4

20.33 int_3d_gauss_45

20.34 int_3d_gauss_5

20.35 int_3d_nc_11

20.36 int_3d_nc_4

20.37 int_3d_nc_6

20.38 int_3d_nc_8

21 master/fem

21.1 mark

21.2 mark_1d

21.3 mesh_1d_uniform

21.4 mesh_2d_uniform

21.5 mesh_3d_uniform

21.6 neighbour_1d

21.7 pdeeig_1d

21.8 pdeeig_2d

21.9 pdeeig_3d

21.10 potential_const

21.11 potential_coulomb

21.12 potential_harmonic_oscillator

21.13 project_circle

21.14 project_rectangle

21.15 promote_1d_2_3

21.16 promote_1d_2_4

21.17 promote_1d_2_5

21.18 promote_1d_2_6

21.19 recalculate_regularity_2d

21.20 refine_1d

21.21 refine_2d_21

21.22 refine_2d_structural

21.23 regularity_1d

21.24 regularity_2d

21.25 regularity_3d

```
{       T = [1 2 3 4];
}
```

21.26 relocate_2d

21.27 vander_1d

22 master/hydrogen-spectrum

22.1 hydrogen_spectrum_1d

22.2 hydrogen_spectrum_2d

22.3 hydrogen_spectrum_3d

23 master/lanczos

23.1 arnoldi

23.2 arnoldi_new

23.3 eigs_lanczos_man

23.4 lanczos

23.5 lanczos_

23.6 lanczos_biorthogonal

23.7 lanczos_biorthogonal_improved

23.8 lanczos_ghep

23.9 mv_lanczos

23.10 reorthogonalise

23.11 test_lanczos

24 master/linear-systems

24.1 gmres_man

```
break on convergence
```

24.2 minres_recycle

25 master/plot

25.1 attach_boundary_value

25.2 cartesian_polar

25.3 img_vargrid

25.4 plot_basis_functions

25.5 plot_convergence

25.6 plot_dof

25.7 plot_eigenbar

25.8 plot_error_estimation

25.9 plot_error_estimation_2

25.10 plot_error_fem

25.11 plot_fdm_kernel

25.12 plot_fdm_vs_fem

25.13 plot_fem_accuracy

25.14 plot_function_and_grid

25.15 plot_hat

25.16 plot_hydrogen_wf

25.17 plot_mesh

25.18 plot_mesh_2

25.19 plot_refine

25.20 plot_refine_3d

25.21 plot_runtime

25.22 plot_spectrum

25.23 plot_wavefunction

26 master/ported

26.1 assemble_2d_dphi_dphi

26.2 assemble_2d_phi_phi

26.3 assemble_3d_dphi_dphi

26.4 assemble_3d_phi_phi

26.5 dV_2d_

26.6 derivative_2d

26.7 derivative_3d

26.8 element_neighbour_2d

26.9 prefetch_2d_

26.10 promote_2d_3_10

26.11 promote_2d_3_15

26.12 promote_2d_3_21

26.13 promote_2d_3_6

26.14 promote_3d_4_10

26.15 promote_3d_4_20

26.16 promote_3d_4_35

26.17 vander_2d

26.18 vander_3d

27 master/sandbox

27.1 adapt

27.2 assoc_laguerre

27.3 assoc_legendre

27.4 c23

27.5 confinement_dat

27.6 convergence_2d_3d

27.7 convergence_matrix_powers

27.8 cut_out

27.9 derivative_2d

27.10 derivative_3d

27.11 dummy

27.12 eig_error

27.13 eigs_fix

27.14 energy_level

27.15 equalise

27.16 example_int64

```
Basic operations
```

```
Matrix multiplication
Timing
```

28 master/sandbox/fem-matlab

28.1 boundary_circle

28.2 boundary_rectangle

28.3 geometry_circle_with_hole

28.4 geometry_rectangle

29 master/sandbox

29.1 fem_2d_estimate_error

29.2 fem_assemble_scratch

29.3 fem_s

29.4 fourier_h

29.5 grad_2d

29.6 grad_3d

29.7 gradient

29.8 harmonic_oscillator

29.9 hydrogen_2d_analytic

29.10 hydrogen_boxed

29.11 hydrogen_boxed_old

29.12 hydrogen_wave

```
% Hydrogen atom
```

29.13 hydrogen_wf

29.14 ichol_man

29.15 known_eigenvalue

29.16 kron_man

29.17 laguerre

29.18 laplacian_arbitrary_order_old

29.19 laplacian_convergence

29.20 laplacian_cut_out

29.21 laplacian_cylindrical

29.22 laplacian_non_uniform_old

29.23 laplacian_polar

29.24 laplacian_simple

29.25 lderivative_3d

29.26 list_dat

29.27 matlab-horner

29.28 mesh_to_grid_2d_3

29.29 mg_mat

29.30 mv

29.31 orth2

29.32 partial_derivative_2d

29.33 partition_function

29.34 partition_function_old

29.35 poisson

29.36 poisson_fem

29.37 potential

29.38 quick_newihbour

29.39 radial

29.40 radial_convergence

29.41 radial_wafefunction

29.42 refine_2d

29.43 refine_3d

29.44 relerr

29.45 restore_cw

29.46 runtime_bm

29.47 rydberg

29.48 s_old

29.49 snorm

29.50 spherical_harmonic

29.51 split_eig

29.52 sum1

29.53 sum3

30 master/sandbox/summation

30.1 acc

30.2 add

30.3 ape

30.4 mmul_accurately

30.5 sum_kahan

30.6 sum_pairwise

30.7 test_sum

31 master/sandbox

31.1 test_fem_1d

31.2 test_fem_2d

31.3 test_fem_3d

31.4 test_increase

31.5 test_ldl

31.6 test_power

31.7 trefethen_p8_fdm

31.8 vander_3d

31.9 wavefunc

31.10 xgrid

32 master/test

32.1 dat_test_lanczos_3d_k_20_n_40

32.2 poisson2d_blk

32.3 qr_implicit_givens_2

32.4 spectral_derivative_2d

32.5 test_2d_eigensolver_hydrogen

32.6 test_2d_refine

32.7 test_3d_eigensolver_hydrogen

32.8 test_FEM

32.9 test_Mesh_3d

32.10 test_arnoldi

32.11 test_arpackc

32.12 test_assemble

32.13 test_assembly_performance

32.14 test_bc_one_sided

32.15 test_compare_solvers

32.16 test_complete

32.17 test_convergence

32.18 test_convergence_b

32.19 test_df_2d

32.20 test_eig_algs

32.21 test_eigs_lanczos

32.22 test_eigs_lanczos_1

32.23 test_eigs_lanczos_2

32.24 test_eigs_lanczos_performance

32.25 test_fdm

32.26 test_fdm_d_vargrid

32.27 test_fdm_spectral

32.28 test_fem

32.29 test_fem_1d

32.30 test_fem_1d_higher_order

32.31 test_fem_2d_adaptive

32.32 test_fem_2d_higher_order

32.33 test_fem_3d_higher_order

32.34 test_fem_3d_refine

32.35 test_fem_b

32.36 test_fem_derivative

32.37 test_fem_quadrature

32.38 test_final

32.39 test_fix_substitution

32.40 test_forward

32.41 test_get_sparse_arrays

32.42 test_harmonic_oscillator

32.43 test_high_order_fdm_periodic_bc

32.44 test_hydrogen_wf

32.45 test_ichol

32.46 test_interpolation

32.47 test_it_vs_exact

32.48 test_jama

32.49 test_jd

32.50 test_jdqz

32.51 test_lanczos_2

32.52 test_lanczos_biorthogonal

32.53 test_laplacian

32.54 test_laplacian_non_uniform

32.55 test_laplacian_simple

32.56 test_mesh_2d_uniform

32.57 test_mesh_2d_uniform_2

32.58 test_mesh_circle

32.59 test_mesh_generation

32.60 test_mg

32.61 test_minres_recycle

32.62 test_multigrid

32.63 test_nc

32.64 test_nonuniform_symmetric

32.65 test_pde

32.66 test_permutation

32.67 test_poison_fem

32.68 test_polar

32.69 test_potential

32.70 test_powers

32.71 test_precondition

32.72 test_project_rectangle

32.73 test_qr

32.74 test_quantum_well

32.75 test_radial_adaptive

32.76 test_radial_confinement

32.77 test_radial_fixes

32.78 test_refine_2d

32.79 test_refine_2d_b

32.80 test_refine_3d

32.81 test_refine_structural

32.82 test_regularisation

32.83 test_round_off

32.84 test_schrödinger_potentials

32.85 test_uniform_mesh

32.86 test_vargrid

33 number-theory

33.1 ceiln

```
floor to leading n-digits
```

33.2 digitsb

```
number of digits with respect to specified base
```

33.3 floorn

```
floor to n-digits
```

33.4 iseven

```
true for even numbers in X
```

33.5 multichoosek

```
all combinations of lenght k from set values with repetitions
c.f. nchoosek, combinations without repetition

input :
      x : scalar integer or vector of arbitrary numbers
      k : length of subsets
output :
      if x scalar : number of combinations
      if x vector : the exact combinations
```

33.6 nchoosek_man

```
vecotrised binomial coefficient
b = N!/K!(N-K)!
```

33.7 pythagorean_triple

```
pythagorean triple
```

33.8 roundn

```
round to n digits
```

34 numerical-methods/differentiation

34.1 derivative1

```
first derivative on variable mesh
second order accurate
```

34.2 derivative2

```
second derivative on a variable mesh
```

35 numerical-methods/finite-difference

35.1 cdiff

```
differences of columns of X
degree = 1 : central first order differences
degreee = 2 : central second order differences
```

35.2 cdiffb

```
differences of columns of X
degree = 1 : central first order differences
degreee = 2 : central second order differences
TODO use difference matrix function for simplicity
```

35.3 cmean

```
single gaussian smoothing step with kernel 1/4*[1,2,1]
```

35.4 derivative_matrix_1_1d

```
finite difference matrix of first derivative in one dimensions
```

35.5 derivative_matrix_2_1d

```
finite derivative matrix of second derivative in one dimension
```

35.6 derivative_matrix_2d

```
finite difference derivative matrix in two dimensions
```

35.7 derivative_matrix_curvilinear

```
derivative matrix on a curvilinear grid
```

35.8 derivative_matrix_curvilinear_2

```
derivative matrix on a two dimensional curvilinear grid
the grid has not necessarily to be orthogonal
```

35.9 difference_kernel

```
difference kernels for equispaced grids
c.f. Computing the Spectrum of the Confined Hydrogen Atom, Kastner,
     2012
```

35.10 distmat

```
distance matrix for a 2 dimensional rectangular matrix
```

35.11 gradpde2d

```
objective function gradiend on two dimensional regular grid
numeric gradient for non-linear least squares optimisation
of a PDE on a rectangular grid
x_* = min(f(x))
f = (v(x) - v(x_*))^2 = f(x) + A dx + O(dx^2)
a_ij = df_i/dx_j
```

35.12 laplacian

35.13 laplacian_fdm

```
finite difference matrix of the laplacian
BC
```

35.14 left

```
left element of vector, leftmost column is extrapolated
```

35.15 lrmean

```
mean of the left and right element
```

35.16 mid

```
mid point between neighbouring vector elements
```

35.17 pwmid

```
segment end point to segment mid point transformation for regular 1
    d grids
```

35.18 ratio

```
ratio of two subsequent values
```

35.19 steplength

```
step length of a vector if it were equispaced
```

35.20 swapoddeven

```
swap odd and even elements in a vector
```

35.21 test_derivative_matrix_2d

35.22 test_derivative_matrix_curvilinear

35.23 test_difference_kernel

36 numerical-methods/finite-volume/@Advection

36.1 Advection

```
FVM treatment of the Advection equation
```

36.2 dot_advection

```
advection equation
```

37 numerical-methods/finite-volume/@Burgers

37.1 burgers_split

```
viscous Burgers' equation,
mixed analytic and numerical derivative in frequency space
by splitting sheme
u_t = -(0.5*u^2)_x + c*u_xx
```

37.2 dot_burgers_fdm

```
viscous burgers' equation
u_t = -d/dx (1/2*u^2) + c d^2/dx^2 u_xx
```

37.3 dot_burgers_fft

```
viscous Burgers' equation in frequency space
u_t + (0.5*u^2)_x = c*u_xx
```

38 numerical-methods/finite-volume/@Finite_Volume

38.1 Finite_Volume

```
finite volume method for partial differential equations 1+1
    dimensions
(time and space)
```

38.2 apply_bc

```
apply boundary conditions
```

38.3 solve

```
solve the the PDE by successively stepping in time
this is a trivial implmentation with constant step length
severity of diffusive error depends on dt/dx-ratio
stability depends on wave height
                      printf('Progress %2.1f%% %2.1fs\n',100*(t-Ti
                          (1))/(Ti(2)-Ti(1)),t_real);
```

38.4 step_split_strang

```
step in time, treat inhomogeneous part by Strang splitting
this scheme is not suitable for stationary solutions, for example
steady shallow water flow
```

38.5 step_unsplit

```
step in time, without splitting the inhomogeneous term
```

39 numerical-methods/finite-volume/@Flux_Limiter

39.1 Flux_Limiter

```
class of flux limiters
```

39.2 beam_warming

```
beam warming sheme
low resolution
note: works only if sign of eigenvalues point into the same
    direction according to RL
```

39.3 fromm

```
fromme limiter
low res
```

39.4 lax_wendroff

```
lax wendroff scheme
second order accurate, but no tvd
this is effectively not a limiter
eq. 6.39 in randall, leveque
```

39.5 minmod

```
min-mod schock limiter
```

39.6 monotized_central

```
monotonized central flux limiter
```

39.7 muscl

```
muscl flux limiter
```

39.8 superbee

```
superbee limiter
```

39.9 upwind

```
godunov scheme
godunov, first order accurate
```

39.10 vanLeer

```
van Leer limiter
```

40 numerical-methods/finite-volume/@KDV

40.1 dot_kdv_fdm

```
korteweg de vries equation
u_t + (0.5*u^2)_x = c*u_xxx
```

40.2 dot_kdv_fft

```
korteweg de vries equation
compute derivatives in frequency space
u_t + (0.5*u^2)_x = c*u_xxx
```

40.3 kdv_split

```
korteweg de vries equation in frequency space,
derivative treated by splitting scheme
```

41 numerical-methods/finite-volume/@Reconstruct_Average_Evolve

41.1 Reconstruct_Average_Evolve

```
Reconstruct Average Evolve Finite Volume Method for treatment of
    1+1D pdes

McCronack Scheme
err = O(dt^2) + O(dx^2), except as discontinuities
error:
      h_xxx(3:end-2) = 1/dx^3*( -0.5*h(1:end-4) + h(2:end-3) - h(4:
          end-1) + 0.5*h(5:end) );
      th = -1/6*dx^2*qh_.*(1 - (qh_*dt/dx).^2).*h_xxx;
```

41.2 advect_highres

```
single time step for the reconstruct evolve algorithm
```

41.3 advect_lowress

```
single time step
low resolution
```

42 numerical-methods/finite-volume

42.1 Godunov

```
Godunov, upwind method for systems of pdes
```

42.2 Lax_Friedrich

```
Lax-Friedrich-Method
for hyperbolic conservation laws
err = O(dt) + O(dx)
|a dt/dx| < 1
```

42.3 Measure

42.4 Roe

```
non linear roe solver for the SWE (randall, leveque 15.3.1)

The roe solver guarantess:
- A is diagonalisable with real eigenvalues (15.12)
- can be determined by a closed formula
- is an efficient replacement for true Rieman solver
```

42.5 fv_swe

```
wrapper for solving SWE
```

42.6 staggered_euler

```
forward euler method with staggered grid
```

42.7 staggered_grid

```
staggered grid approximation to the SWE
```

43 numerical-methods

43.1 grid2quad

```
extract rectangular elements of a structured grid
in form of an unstructured quad-mesh format
```

44 numerical-methods/integration

44.1 cumintL

```
cumulative integral from left to right
```

44.2 cumintR

```
cumulative integral from right to left
```

44.3 int_trapezoidal

```
integrate y along x with the trapezoidal rule
```

45 numerical-methods/interpolation/@Kriging

45.1 Kriging

```
class for Kriging interpolation
```

45.2 estimate_semivariance

```
estimate the parameter of the semivariance model for Kriging
    interpolation
            % set up the regression matrix and solve for
                parameters
```

45.3 interpolate_

```
interpolate with Krieging method

this function may interpolate several quantities per coordinate,
using the same variogram, if the semivariance of the quantities
    differs,
the user may prefer to estimate the semivariance and interpolate
    each quantity
individually
```

```
Xs  : source point coordinates
Vs  : value at source points
Xt  : targe point coordinates
Vt  : value at target points
E2t : squared interpolation error at target points
```

46 numerical-methods/interpolation/@RegularizedInterpolator

46.1 RegularizedInterpolator1

```
class for regularized interpolation (Thikonov) on a 1D mesh
```

46.2 init

```
initialize the interpolator with a set of sampling points
```

47 numerical-methods/interpolation/@RegularizedInterpolator

47.1 RegularizedInterpolator2

```
class for regularized interpolation on an unstructures mesh (
    interpolation)
```

47.2 init

```
initialize the interpolator with a set of point samples
```

48 numerical-methods/interpolation/@RegularizedInterpolator

48.1 RegularizedInterpolator3

```
class for regularized interpolation (Tikhonov) on a triangulation
(unstructured mesh)
```

48.2 init

```
initialize the interpolator with a set of sampling points
```

49 numerical-methods/interpolation

49.1 IDW

```
spatial averaging by inverse distance weighting
```

49.2 IPoly

```
polynomial interpolation class
```

49.3 IRBM

```
interpolate by the radial basis function method
                    fprintf(1,'Progress IRBM: %d%%\n',round(100*
                        idx/size(Xi,1)));
```

49.4 ISparse

```
sparse interpolation class
```

49.5 Inn

```
nearest neighbour interpolation
```

49.6 Interpolator

```
interpolator super-class
                           fprintf(1,'Progress: %f%% %fs\n',100*
                               idx/size(Xt,1),t);
```

49.7 fixnan

```
fill nan-values in vector with gaps
```

49.8 idw1

```
spatial average ny inverse distance weighting
```

49.9 idw2

```
spatial average by inverse distance weighting
```

49.10 inner2outer

```
linear interpolation of segment mit point to grid points at segment
     ends
assumes equal grid spacing
```

49.11 inner2outer2

```
interpolate from element (segment) centres to edge points
```

49.12 interp1_limited

```
interpolate values, but not beyond a certain distance
this function is idempotent, i.e. it will not extrapolate over into
     gaps
exceedint the limit and thus not spuriously extend the series when
    called a second time on the same data
```

49.13 interp1_man

```
interpolate
```

49.14 interp1_save

```
make interpolation save to round off errors
the matlab internal interpolation suffers from rounding errors,
    which
are unacceptable when values of X and Y are large (for example UTm
    coordinates)
this normalization prevents this
```

49.15 interp1_slope

```
quadratic interpolation returning value and derivative(s)
```

49.16 interp1_smooth

49.17 interp1_unique

```
matlab fails to interpolate, when x values are not unique
this function makes the values unique before use
```

49.18 interp2_man

```
nearest neighbour interpolation in two dimensions
```

49.19 interp_angle

```
interpolate an angle
```

49.20 interp_fourier

```
interpolation by the fourier method
```

49.21 interp_fourier_batch

```
batch interpolation by the fourier interpolation
```

49.22 interp_sn

```
interpolate along streamwise coordinates
This gives similar result to setting aspect ratio for sN to
    infinity,
but not quite,as the input point set is not dense (scale for sN to
    infinity does not work)
              sdx  = sdx(sdx_);
```

49.23 interp_sn2

```
interpolation in streamwise coordinates
```

49.24 interp_sn3

49.25 interp_sn_

49.26 limit_by_distance_1d

```
smooth subsequent values along a curve such that
      v(x0+dx) < v(x0) + (ratio-1)*dx
if v is the edge length in a resampled polygon, then v_i/v_(i+1) <
    ratio
      ratio^1 = exp(a*1)
```

49.27 resample1

```
interpolation along a parametric curve with variable step width
```

49.28 resample_d_min

```
resample a function
```

49.29 resample_vector

```
resample a track so that velocity vectors do not run into each
    other
```

49.30 test_interp1_limited

50 numerical-methods

50.1 inverse_complex

51 numerical-methods/ode

51.1 bvp2_check_arguments

51.2 bvp2c

```
solve system of non-linear second order odes (in more than one
    variable)
as boundary value problems

odefun provides ode coefficients c:
c(x,1) y''(x) + c(x,2) y'(x) + c(x,3) y = c(x,4)
   c_1 y"    + c_2 y'     + c_3 y + c_4 = c_4

subject to the boundary conditions
bcfun provides v and p and optionally q, so that:

b_1 y + b_2 y' = f
   q(x,1)*( p(x,1) y_l(x) + p(x,2) y_l'(x)
```

```
 + q(x,2)*( p(x,1) y_r(x) + p(x,2) y_r'(x) = v(x)
where q weighs the waves travelling from left to right and right to
     left (default [1 1])
```

51.3 bvp2c2

```
solve second order boundary value problem via roots of the
    characteristic
polynomial

input:

x   : [nx1] discretized domain
      n : number of vertices
     nxc = n-1 : number of segments

bc  : struct : boundary condition
      bc.p(1)*y(0) + bc.pd(2)*y'(0) = bc.val(1)
      bc.p(2)*y(L) + bc.pd(2)*y'(L) = bc.val(2)

output:

A   : [2*nxc x 2*ns] disrcretisation matrix
rhs : [2*nxc x 1] right hand size

y = A^-1 rhs
```

51.4 bvp2fdm

```
solve system of non-linear second order odes (in more than one
    variable)
as boundary value problems by the finite difference method

odefun provides ode coefficients c:
c(x,1) y''(x) + c(x,2) y'(x) + c(x,3) y = c(x,4)
 c_1 y" + c_2 y' + c_3 y + c_4 = 0

subject to the boundary conditions
bcfun provides v and p and optionally q, so that:

b_1 y + b_2 y' = f
   q(x,1)*( p(x,1) y_l(x) + p(x,2) y_l'(x)
 + q(x,2)*( p(x,1) y_r(x) + p(x,2) y_r'(x) = v(x)
where q weighs the waves travelling from left to right and right to
     left (default [1 1])
```

51.5 bvp2wavetrain

```
solve second order boundary value problem by repeated integration
```

51.6 bvp2wavetwopass

```
two pass solution for the linearised wave equation
solve first for the wave number k, and then for y
```

51.7 ivp_euler_forward

```
solve intial value problem by the euler forward method
```

51.8 ivprk2

```
solve initial value problem by the two step runge kutta method
```

51.9 ode2_matrix

```
transformation matrix of second order ode
to left and right going wave

c = odefun(x)
c1 y'' + c2' y + c3 y == 0
y = y_p + y_m, left and right going wave
d/dx [y_p, y_m] = A*[y_m, y_p]
```

51.10 ode2characteristic

```
second order odes
transmittded and reflected wave
```

51.11 step_trapezoidal

```
single trapezoidal step
```

51.12 test_bvp2

52 numerical-methods/optimisation

52.1 armijo_stopping_criterion

```
armijo stopping criterion for optimizations
```

52.2 astar

```
astar path finding alforithm
```

52.3 binsearch

```
binary search on a line
```

52.4 bisection

```
bisection
```

52.5 box1

```
test objective function for optimisation routines
```

52.6 box2

52.7 cauchy

52.8 cauchy2

```
solve non-linear system by cuachy's method
slower than quadratic optimisation, but does not require a hessian
fun : objective function, returns
      f  : scalar, objective function value
      g  : nx1, gradient
x   : nx1, initial position
opt : options
```

52.9 directional_derivative

```
directional (projected) derivative
d : derivative, highest first
p : series expansion around x0
```

52.10 dud

```
optimization by the dud algorithm
```

52.11 extreme3

```
extract maxima by quadratic approximation from sampled function val
    (t)
intended to be called after [mval, mid] = max(val) for refinement
    of
locatian and maximum

input
t    : sampling time (uniformly spaced)
v    : values at sampling times
ouput:
tdx  : index where extremum should be computed
t0   : location of the extremum
val0 : value of extremum

v'(dt0) = 0 and v''(dt0) determines type of extremum
```

52.12 extreme_quadratic

52.13 ftest

52.14 grad

```
numerical gradient
```

52.15 hessian

```
numerical hessian
```

52.16 hessian_from_gradient

```
numerical hessian from gradient
```

52.17 hessian_projected

```
numerical hessian projected to one dimenstion
```

52.18 line_search

```
bisection routine
```

52.19 line_search2

```
bisection method

fun : objective funct
x0  : start value
f0  : objective function value at x0
g   : gradient at x0
p   : search direction from x0 (p = g for steepest descend)
h   : initial step length (default 1)
lb  : lower bound for x
up  : upper bound for x
```

52.20 line_search_polynomial

```
polynomial line search
fun : objective funct
x0  : start value
f0  : objective function value at x0
g   : gradient at x0
dir : search direction from x0 (p = g for steepest descend)
h   : initial step length (default 1)
lb  : lower bound for x
up  : upper bound for x
```

52.21 line_search_polynomial2

```
cubic line search
fun : objective funct
x0  : start value
f0  : objective function value at x0
g   : gradient at x0
dir : search direction from x0 (p = g for steepest descend)
h   : initial step length (default 1)
lb  : lower bound for x
up  : upper bound for x
```

52.22 line_search_quadratic

```
quadratic line search
fun : objective funct
x0  : start value
f0  : objective function value at x0
g   : gradient at x0
dir : search direction from x0 (p = g for steepest descend)
h   : initial step length (default 1)
lb  : lower bound for x
up  : upper bound for x
```

52.23 line_search_quadratic2

```
quadratic line search
```

52.24 line_search_wolfe

```
line search by wolfe method
c.f.: OPTIMIZATION THEORY AND METHODS - Nonlinear Programming, Sun,
     Yuan
```

52.25 ls_bgfs

```
least squares by the bgfs method
```

52.26 ls_broyden

```
least squares by the broyden method
for rectangular / non symmetric systems
      Numerical Optimization nocedal
      Practical Methods of  Optimization fletcher
c.f. gerber 1981
c.f. fletcher 1978 (more advanced, not used here)
c.f. Kelley 1999 ch. 4

BGFS:
Broyden 1965
Fletcher 1970
Goldfarb 1970
Shanno 1970
```

52.27 ls_generalized_secant

```
least squares by the secant method
Barnes, 1965
Wolfe, 1959
Fletcher 1980, 6.3
seber 2003
gerber
```

52.28 nlcg

```
non-linear conjugate gradient
input:
x   : nx1 start vectort
```

```
opt : struct options
fdx : gradient constraint
```

52.29 nlls

```
non-linear least squares
```

52.30 picard

```
picard iteration
```

52.31 poly_extrema

```
extrema of a polynomial
```

52.32 quadratic_function

```
evaluate quadratic function in higher dimensions
```

52.33 quadratic_programming

```
optimize by quadratic programming
```

52.34 quadratic_step

```
single step of the quadratic programming
```

52.35 rosenbrock

```
rosenbrock test function
```

52.36 sqrt_heron

```
Heron's method for the square root
```

52.37 test_directional_derivative

52.38 test_dud

52.39 test_line_search_quadratic2

52.40 test_ls_generalized_secant

52.41 test_nlcg_6_order

52.42 test_nlls

```
f = w'*(p*abs(x-1).^4) + w'*(1-p)*abs(x-1).^2;
```

53 numerical-methods/piecewise-polynomials

53.1 Hermite1

```
hermite polynomial interpolation in 1d
```

53.2 hp2_fit

```
fit a hermite polynomial
coefficients are derivative free
x0  : left point of first segment
x1  : right point of last segment
n   : number of segments
x   : sample x-value
val : sample y-value
c   : coefficients (values at points, no derivatives)
```

53.3 hp2_predict

```
prediction with pw hermite polynomial
c are values at support points
```

53.4 hp_predict

```
predict with piecewise hermite polynomial
```

53.5 hp_regress

```
fit piecewise hermite polynomial
coefficients are values and derivatives
```

53.6 lp_count

```
lagrangian basis for interpolation
count number of valid samples
```

53.7 lp_predict

```
lagrangian basis piecwie interpolation, predicor
```

53.8 lp_regress

53.9 lp_regress_

54 regression/@PolyOLS

54.1 PolyOLS

```
class for polynomial least squares
```

54.2 coefftest

54.3 detrend

```
detrending by polynomial regression
```

54.4 fit

```
fit a polynomial function
like polyfit, but returns parameter error estimates
```

54.5 fit_

```
fit a polynomial function
```

54.6 predict

```
predict polynomial function values
```

54.7 predict_

54.8 slope

```
slope by linear regression
```

55 regression/@PowerLS

55.1 PowerLS

```
class for power law regression
```

55.2 fit

```
fit a power law
like polyfit, but returns parameter error estimates
```

55.3 predict

```
predict with power law
      S2 = diag((A*obj.C)*A');
      L  = Y - S;
      U  = Y + S;
```

55.4 predict_

56 regression/@Theil

56.1 Theil

```
Kendal-Theil-Sen robust regression
```

56.2 detrend

```
linear detrending of a set of samples by the Theil-Senn Slope
```

56.3 fit

```
fit slope and intercept to a set of sample with the Theil-Sen
    method

c     : confidence interval c = 2*ns*normcdf(1) for ns-sigma
    intervals
param : itercept and slope
P : confidence interval
```

56.4 predict

```
predict values and confidence intervals with the Theil-Sen method
```

56.5 slope

```
fit the slope with the Theil-Sen method
```

57 regression

linear and non-linear regression

57.1 Theil_Multivariate

```
extension of the Theil-Senn regression to higher dimensions by
means of the Gauss-Seidel iteration
```

57.2 areg

```
regression using the pth-fraction of samples with smallest residual
```

57.3 ginireg

```
gini regression
```

57.4 hesssimplereg

```
hessian, gradient and objective function value of the simple
    regression
rhs = p(1) + p(2) x + eps
```

57.5 l1lin

```
solve ||Ax - b||_L1 by means of linear programming
```

57.6 lsq_sparam

```
parameter covariance of the least squares regression

fun : model function for predtiction
b   : sample values
f(p) = b
p   : parameter at point of evaluation (preferably optimum)
```

57.7 polyfitd

```
fit a polynomial of order n to a set of sampled values and sampled
    values
of the derivative

x0 must contain at least for conditioning as otherwise the
    intercept
cannot be determined
```

57.8 regression_method_of_moments

```
fit linear function ||a b x = y||_L2 by the method of moments
y+eps = alpha + beta*x
```

57.9 robustlinreg

```
fit a linear function by splitting the x-values at their median
      (med(y_left) - med(y_right))/(med(x_left)-med(x_right)
this approach performs poorly compared to the theil-senn operator
```

57.10 theil2

```
Theil senn-estimator for two dimensions (glm)
```

57.11 theil_generalised

```
generalization of the Theil-Senn operator to higher dimensions,
for arbitrary functions such as polynomials and multivariate
    regression
either higher order polynomials or glm
c.f. "On theil's fitting method", Pegoraro, 1991
```

57.12 total_least_squares

```
total least squares
```

57.13 weighted_median_regression

```
weighted median regression
c.f. Scholz, 1978
```

58 set-theory

58.1 issubset

```
test if set B is subset of A in O(n)-runtime

A : first set
B : second set
P : set of primes (auxiliary)
```

59 signal-processing

59.1 acf_effective_sample_size

```
effective sample size from acf
```

59.2 acf_genton

```
autocorrelation function
```

59.3 acfar1

```
Autocorrelation function of the finite AR1 process

a_k = 1/(n-k)sum x_ix_i+1 + (xi + xi+k)mu + mu^2
    = r^k + 1/n sum_ij + 1/n
       pause
```

59.4 acfar1_2

```
autocorrelation of the ar1 process
```

59.5 acfar2

```
impulse response of the ar2 process
```

59.6 acfar2_2

```
autocorrelation of the ar2 process
X_i + a1 X_i-1 + a2 X_i-2 = 0
```

59.7 ar1_cutoff_frequency

59.8 ar1_effective_sample_size

```
effective sample size correction for autocorrelated series
```

59.9 ar1_mse_mu_single_sample

```
standard error of a single sample of an ar1 correlated process
```

59.10 ar1_mse_pop

```
variance of the population mean of a single realisation around zero

E[(mu_N-0)^2] = E[mu_N^]
```

59.11 ar1_mse_range

```
mean standard error of the mean of a range of values taken from an
    ar1 process
```

59.12 ar1_spectrum

```
spectrum of the ar1 process
```

59.13 ar1_to_tikhonov

```
convert ar1 correlation to tikhonovs lambda
```

59.14 ar1_var_factor

```
variance correction factor for an autocorrelated finite process
n   : [1 .. inf] population size
m   : [1 .. n]  samples size
rho : [ -1 < rho < 1 (for convergence) ] correlation of samples
```

59.15 ar1_var_factor_

```
variance of an autocorrelated finite process
```

59.16 ar1_var_range2

```
variance of sub sample starting at the end of the series

from the finite length first order autocorrelated process

s2 = 1/m^2 sum_i^m sum_j^m rho^-|i-j|
```

59.17 ar1delay

```
approximate acf by the ar1 process
acf: autocovariance or autocorrerlation function
nf : skip first samples (for mixed geometric-arithmetic series (
    ARMA)
```

59.18 ar1delay_old

```
autocorrelation of the residual
```

59.19 ar2conv

```
coefficients of the ar2 process determined from the two leading
    correlations
of the acf [1,r1,r2,...]
```

59.20 ar2dof

```
effective samples size for the ar2 process
```

59.21 ar2param

```
ar2 parameter estimation from first two terms of acf

acf = [1 a1 a2 ...]
```

59.22 asymwin

```
creates asymmetrical filter windows
filter will always have negative weights
```

59.23 autocorr_fft

```
autocorrelation function
```

59.24 bandpass

```
bandpass filter
```

59.25 bandpass2

```
bandpass filter
```

59.26 bartlett

```
Effective sample size factor for bartlett window
c.f. thiebaux
c.f spectral analysis-jenkins, eq. (6.3.27)
c = acf
note: results seams always to be 1 tac too low
T : reduction factor for dof
for ar1 with a = rho^k = exp(-k/L), T = 2L
```

59.27 bartlett_spectrogram

```
bartlet spectrogramm
TODO sliding window
```

59.28 bin1d

```
bin values of v sampled at x into bins bounded by "edges"
apply function v to it
```

59.29 bin2d

```
bin values of V sampled at X and Y into the grid structured grid ex
    ,ey
apply function func to all walues in the bin
func = mean : default
func = sum : non-normalized frequency histogram in 2D
```

59.30 binormrnd

```
generate two correlated normally distributed vectors
```

59.31 conv1_man

```
convolutions with padding
```

59.32 conv2_man

```
convolution in 2d
```

59.33 conv2z

59.34 conv30

```
convolve with rectangular window of lenght n
circular boundaries
```

59.35 conv_

```
convolution of a with b
```

59.36 conv_centered

```
convolve x with filter window f
when length of f is even, this guarantees a symmetric result (no
    off by on
displacement) by making the lenght of f odd at first
```

59.37 convz

59.38 cosexpdelay

59.39 csmooth

```
smooth recursively with [1,2,1]/4 kernel
```

59.40 daniell_window

```
Daniell window for smoothing the power spectrum
c.f. Daniell 1946
Bloomfield 2000
meko 2015
```

59.41 danielle_window

```
danielle fourier window
```

59.42 db2neper

```
convert decibel to neper
```

59.43 db2power

```
power ratio from db
```

59.44 derive_danielle_weight

59.45 derive_limit_0_acfar

59.46 detect_peak

```
detect peaks in a vector
requires function value to fall to p*max before new value is
    allowed
```

59.47 digital_low_pass_filter

```
design coefficients of a low pass filter with specified cut of
    frequency
and sampling period
alalogue low pass with pole at s=-omega_c=1/tau=1/RC
Ha = tau/(tau + s) = 1/(1 + omega_c*s)
```

59.48 doublesum_ij

```
double sum of r^i
```

59.49 effective_sample_size_to_ar1

```
convert effective sample size to ar1 correlation
```

59.50 filt_hodges_lehman

59.51 filter1

```
filter along one dimension
```

59.52 filter2

```
filter columns of x (matlab does only support vector input)
```

59.53 filter_

```
invalidate values that exceed n-times the robust standard deviation
```

59.54 filteriir

```
filter adcp t-n data over time

v : nz,nt  : values to be filtered
H : nt,1   : depth of ensemble
last : nt,1 : last bin above bottom that can be sampled without
    side lobe interference
nf : scalar : number of reweighted iterations

when samples
- distance to bed is reference (advantageous for near-bed suspended
     transport)
TODO for wash load: distance to surface is more relevant
interpolate depending on z

when depth changes, neighbouring indices do not correspond to same
    relative position in the water column
relative poisition in the colum (s-coordinate) smoothes values
```

```
near the bed: absolute distance to bed is chosen
near surface: absolute distance to surface is chosen
-> cubic transformation of index

faster and avoid alising (smoothing along z)
      resample ensemble to same number of bins in S -> filter ->
          resample back
      use nonlinear transform z-s coordinates
-> resampling has to be local (Hi -> H-filtered)

filtered profile coordinates to sample coordinates
      zf -> zi (special transform)
corresponding indices and fractions
filtration step (update of hf and vf)
sample coordinates to updated profile coordinates
(the inverse step is actually not necessary)
write filtered value
```

59.55 filterp

59.56 filterp1

```
fir filter with some fancy extras
```

59.57 filterstd

59.58 firls_man

```
design finite impulse response filter by the least squares method
```

59.59 flattopwin

```
the flat top window
```

59.60 frequency_response_boxcar

```
frquency response of a boxcar filter
```

59.61 freqz_boxcar

```
frequncy response of a boxcar filter
```

59.62 gaussfilt1

```
filter data series with a gaussian window
```

59.63 hanchangewin

```
hanning window for change point detection
```

59.64 hanchangewin2

```
nanning window for chage point detection
```

59.65 hanwin

```
hanning filter window
```

59.66 hanwin_

```
hanning filter window
```

59.67 highpass

```
high pass filter
```

59.68 kaiserwin

```
kaiser filter window
```

59.69 kalman

```
Kalman filter
```

59.70 lanczoswin

```
Lanczos window
```

59.71 last

```
lake tail, but for matrices
```

59.72 lowpass

```
low pass filter
```

59.73 lowpass2

```
design low pass filter with cutoff-frequency f1
```

59.74 lowpass_iir

```
iir-low pass
```

59.75 lowpass_iir_symmetric

```
two-sided iir low pass filter (for symmetry)
```

59.76 lowpassfilter2

```
low-pass filter of data
```

59.77 maxfilt1

59.78 meanfilt1

```
moving average filter with special treatment of the boundaries
```

59.79 medfilt1_man

```
moving median filter, supports columnwise operation
```

59.80 medfilt1_man2

```
moving median filter with special treatment of boundaries
```

59.81 medfilt1_padded

```
median filter with padding
```

59.82 medfilt1_reduced

```
median filter with padding
```

59.83 mid_term_single_sample

```
variance of single sample, mid term
```

59.84 minfilt1

59.85 mu2ar1

```
error variance of the mean of the finite length ar1 process

(mu)^2 = (sum epsi)^2 = sum_i sum_j eps_i eps_j = sum_ii(rho,n)/n^2
this has the limit s^2 for rho->1
```

59.86 nanautocorr

```
autocorrelation with nan-values
```

59.87 nanmedfilt1

```
medfilt1, skipping nans
```

59.88 neper2db

```
convert neper to db
```

59.89 peaks_man

```
peaks of a periodogram
```

59.90 polyfilt1

```
polynomial filter,
can be achieved by iteratively processing the data with
a mean (zero-order) filter
```

59.91 qmedfilt1

```
medfilt1, after fitting a quadratic polynomial
```

59.92 randar1

```
generate random ar1 process
e1 = randar1(sigma,p,n,m)
```

59.93 randar1_dual

```
draw random variables of two corrlated ar1 processes
```

59.94 randar2

```
generate ar2 process
```

59.95 randarp

```
randomly generate the instance of an ar-p process
```

59.96 range_window

```
range of values within a certain range of indices (window)
```

59.97 rectwin

```
rectangular window
```

59.98 recursive_sum

59.99 select_range

59.100 smooth1d_parametric

```
smooth position of p0=x0,y0 between p1=x1,y1 and p2=x2,y2,
so that distance to p1 and p2 becomes equal
and the chord length remains the same
```

59.101 smooth2

```
smooth vectos of X
```

59.102 smooth_man

59.103 smooth_parametric

```
smooth a parametric function given in x-y coordinates
      matvec2x2(R,[dxc;dyc])
```

59.104 smooth_parametric2

```
parametrically smooth the curve
```

59.105 smoothfft

```
filter with fast fourier transform
```

59.106 spectrogram

```
spectrogram
```

59.107 std_window

```
moving block standard deviation
```

59.108 sum_i_lag

```
sum of ar1 matrix with lag
sum_i=1^n rho^|i-k|
```

59.109 sum_ii

```
sum of ar1 matrix
sum_i=1^n sum_j=1^n rho^|i-j|
this is for the variance, take square root for the standard
    deviation factor
```

59.110 sum_ii_

59.111 sum_ij

```
sum of ar1 matrix
sum_{i=1}^n sum_{j=1}^m r^|i-j|
```

59.112 sum_ij_

59.113 sum_ij_partial_

59.114 sum_multivar

```
sum of matrix entries of bivariate ar1 process
```

59.115 test_acfar1

59.116 test_acfar1_2

59.117 test_acfar1_3

59.118 test_acfar1_4

59.119 test_acfar2

59.120 test_ar1_var_factor

59.121 test_ar1_var_factor_2

59.122 test_ar1_var_mu_single_sample

59.123 test_ar1_var_pop

59.124 test_ar1_var_pop_1

59.125 test_ar1delay

59.126 test_bivariate_covariance_term

59.127 test_convexity

59.128 test_lanczoswin

59.129 test_madcorr

59.130 test_randar1

59.131 test_randar1_multivariate

59.132 test_randar2

59.133 test_sum_ij

59.134 test_sum_multivar

59.135 test_trifilt1

59.136 test_wautocorr

59.137 test_wavelet_transform

59.138 test_wordfilt

59.139 test_xar1_mid_term

59.140 tikhonov_to_ar1

```
convert coefficient of the tikhonov regularization to correlatioon
    of the ar1 process
```

59.141 trapwin

```
trapezoidal filter window
```

59.142 trifilt1

```
filter with triangular window
```

59.143 triwin

```
triangular filter window
```

59.144 triwin2

```
triangular filter window
```

59.145 varar1

```
error variance of a single sample of a finite length ar1 process
with respect to the mean, averaged over the population
```

59.146 welch_spectrogram

```
welch spectrogram
```

59.147 wfilt

```
filter with window
```

59.148 winbandpass

```
filter with bandpass
```

59.149 window_make_odd

59.150 winfilt0

```
filter with window
```

59.151 winlength

```
window length for desired cutoff frequency
power at fc is halved
H(wf) = 1/sqrt(2) H(f)
if the filter window were used as a low pass filter
note: the user should prefer a windowed ideal low pass filter
TODO, relate this to DOF
```

59.152 wmeanfilt

```
mean filter with window
```

59.153 wmedfilt

```
median filter with window
```

59.154 wordfilt

```
weighted order filter
```

59.155 wordfilt_edgeworth

```
weighed order filter
```

59.156 xar1

59.157 xcorr_man

```
cross correlation of two sampled ar1 processes
```

60 sorting

60.1 sort2

```
sort two numbers
```

60.2 sort2d

```
sort elements of matrix in X
returns row and column index of sorted values
```

61 special-functions

61.1 bessel_sphere

```
spherical Bessel function of the first kind
```

61.2 hankel_sphere

```
spherical Hankel function for the far field (incident plane wave)
first kind
```

61.3 hermite

```
probabilistic's hermite polynomial by recurrence relation

input :
n : order
x : value

output:
f : H_n(x)
df : d/dx H_n(x)
```

61.4 legendre_man

```
legendre polynomials
```

61.5 neumann_sphere

```
spherical Neumann function
Bessel function of the second kind
```

62 statistics

62.1 atan_s2

```
stadard deviation of the arcus tangens by means of taylor expansion
```

62.2 beta_mode_to_parameter

```
transform modes (mean and sd) to paramets of the beta function
```

62.3 correlation_confidence_pearson

```
confience intervals of the correlation coefficient
c.f. Fischer 1921
```

63 statistics/distributions

63.1 PDF

```
class for quasi-distributions from a set of sampling points
```

63.2 binorm_separation_coefficient

```
separation coefficient of a bimodal normal distribution
```

63.3 binormcdf

```
bio-modal gaussian distribution
```

63.4 binormfit

```
fit sum of to normal distribution to a histogram
```

63.5 binormpdf

63.6 edgeworth_cdf

```
edgeworth expansion of an unknown cumulative distribution
with mean mu, standard deviation sigma, and third and fourth
    cumulants
c.f. Rao 2010
```

63.7 edgeworth_pdf

```
probability density of and unknown distribution
with mean mu, standard deviation sigma, and third and fourth
    cumulants
c.f. Rao 2010
```

63.8 logn_mode2param

```
transform modes (mu,sd) to parameters of the log normal
    distribution
```

63.9 logn_param2mode

```
transform parameters to mode (mu, sd) for the log normal
    distribution
```

63.10 lognpdf_

```
log normal distribution called by modes rather than parameters
```

63.11 pdfsample

```
pdf from sample distribution
Note: better use kernal density estimates
```

63.12 t2cdf

```
Hotelling's T-squared cumulative distribution
```

63.13 t2inv

```
inverse of Hotelling's T-squared cumulative distribution
```

64 statistics

64.1 example_standard_error_of_sample_quantiles

64.2 f_var_finite

```
reduction of variance when sampling from a finite population
without replacement
```

64.3 gamma_mode_to_parameter

```
transform modes (mu,sd) to parameters of the gamma distribution
```

64.4 hodges_lehmann_correlation

```
hodges_lehmann correlatoon coefficient
c.f. Shamos 1976
c.f. Bickel and Lehmann 1976
c.f. rousseeuw 1993
c.f. Shevlyakov 2011
```

64.5 hodges_lehmann_dispersion

```
dispersion determined by the hodges lehman method
asymptotic efficiency of dispersion estimates:
standard deviation:   E(s - hat s)/s = 2/sqrt(2 n) ~ 0.707/sqrt(n)
              (100%)
hodges lehmann dispersion E(s-\hat s)/s = (pi/3)^2 /(sqrt(2 n)) ~
    0.775/sqrt(n) (91%)
mad                   E(s-\hat s)/s ~ 1.17 s/sqrt(n)
                  (60%)
c.f. Shamos 1976
c.f. Bickel and Lehmann 1976
c.f. rousseeuw 1993
nb: rousseeuw uses the 25th percentile, which is more efficient for
     small sample sizes
```

65 statistics/information-theory

65.1 akaike_information_criterion

```
akaike information criterion

serr : rmse of model prediction
n : effective sample size
k : number of parameters

c.f. akaike (1974)
c.f. sugiura 1978
```

65.2 bayesian_information_criterion

```
bayesian information criterion
```

66 statistics

66.1 kurtncdf

66.2 kurtnpdf

66.3 kurtosis_bias_corrected

```
bias corrected kurtosis
```

66.4 limit

```
limit a by lower and upper bound
```

66.5 logfactorial

```
approximate log of the factorial
```

66.6 loglogpdf

66.7 logskewcdf

66.8 logskewpdf

67 statistics/logu

67.1 lambertw_numeric

```
lambert-w function
```

67.2 logtrialtcdf

```
pdf of a logarithmic triangular distribution
```

67.3 logtrialtinv

```
 inverse of the logarithmic triangular distribution
 = (d F log(a) log(b) + a log(b) - b log(a) - d F log(a) log(c) - a
     log(c) + d F log(b) log(c) + b log(c) - d F log^2(b))/((log(a)
     - log(b)) W((a^(-1/(log(a) - log(b))) (b^(-log(c)/log(a) - 1/
     log(a)) c)^(-log(a)/(log(a) - log(b))) (-d F log^2(b) + a log(b
     ) + d F log(a) log(b) + d F log(c) log(b) - b log(a) - a log(c)
      + b log(c) - d F log(a) log(c)))/(log(a) - log(b))))
x = (d F log(a) log(b) + a log(b) - b log(a) - d F log(a) log(c) - a
     log(c) + d F log(b) log(c) + b log(c) - d F log^2(b))/((log(a)
    - log(b)) W((a^(-1/(log(a) - log(b))) (b^(-log(c)/log(a) - 1/log
    (a)) c)^(-log(a)/(log(a) - log(b))) (-d F log^2(b) + a log(b) +
    d F log(a) log(b) + d F log(c) log(b) - b log(a) - a log(c) + b
    log(c) - d F log(a) log(c)))/(log(a) - log(b))))
```

67.4 logtrialtmean

```
 mean of the logarithmic triangular distribution
```

67.5 logtrialtpdf

```
 density of the logarithmic triangular distribution
```

67.6 logtrialtrnd

67.7 logtricdf

```
 cumulative distribution of the logarithmic triangular distribution
```

67.8 logtriinv

```
 invere of the logarithmic triangular distribution
```

67.9 logtrimean

```
mean of the logarithmic triangular distribution
```

67.10 logtripdf

```
probability density of the logarithmic triangular distribution
```

67.11 logtrirnd

67.12 logucdf

```
probability density of the logarithmic uniform distribution
```

67.13 logucm

```
central moments of the log-uniform distribution
```

67.14 loguinv

```
inverse of the log-uniform distribution
```

67.15 logumean

```
mean of the log-uniform distribution
```

67.16 logupdf

```
pdf of the log uniform distribution
```

67.17 logurnd

```
random numbers following a log-uniform distribution
```

67.18 loguvar

```
variance of the log-uniform distribution
```

67.19 medlogu

```
median of the log-uniform distribution
```

67.20 test_logurnd

67.21 tricdf

```
cumulative distribution of the log-triangular distribution
```

67.22 triinv

```
inverse of the triangular distribution
```

67.23 trimedian

```
median of the triangular distribution
```

67.24 tripdf

```
probability density of the triangular distribution
```

67.25 trirnd

```
random numbers of the triangular distribution
```

68 statistics

68.1 maxnnormals

```
expected maximum of n normal variables
c.f. Wolperts
this is the median, not the mean of the maximum!
see median of gumbel
```

68.2 midrange

```
mid range of columns of X
```

68.3 minavg

```
solution of the minimum variance problem
minimise the variance of the weighted sum of n-independent
random variables with equal mean and individual variance
```

68.4 mode_man

69 statistics/moment-statistics

69.1 autocorr_man3

```
autoccorrelation of the columns of X
```

69.2 autocorr_man4

```
autocorrelation for x if x is a vector, or indivvidually for the
columns of x if x is a matrix

c.f. box jenkins 2008 eq. 2.1.12

Note that it is faster to compute the acf in frequency space
as done in the matlab internal function
```

69.3 autocorr_man5

```
autocorrellation of the columns of X
```

69.4 blockserr

```
estimate the standard error of potetially sequentilly correlated
    data
by blocking
block length should be sufficiently larger than correlation length
and sufficiently smaller than data length
this uses a sliding block approach, which reduces the variation of
    the error estimate
```

69.5 comoment

```
non-central higher order moments of the multivariate normal
    distribution

c.f. Moments and cumulants of the multivariate real and complex
    Gaussian distributions

note : there seem to be some typos in the original paper,
      for x^4 cii^2, the square seems to be missing
mu : nx1 mean vector
C  : nxn covariance matrix
k  : nx1 powers of variables in moments
```

69.6 corr_man

```
correlation of two vectors
```

69.7 cov_man

```
covariance matrix of two vectors
```

69.8 dof

```
mininum number of support points
for a polynomial of degree order in dim dimensions
```

69.9 edgeworth_quantile

```
inverse edgeworth expansion
c.f. cornis fisher 1937
c.f. Rao 2010
c.f. 2.50 in hall
CHERNOZHUKOV 3.3
```

69.10 effective_sample_size

```
effective sample size of the weighted mean of uncorrelated data
c.f. Kish
```

69.11 f_correlation

```
correction factor for standard error of the mean of n ar1-
    correlated iid samples
```

69.12 f_finite

```
reduction factor of standard error for sampling from a finite
    distribution
without replacement
```

69.13 lmean

```
mean of x.^l, not of abs
```

69.14 lmoment

```
l-moment of vector x
```

69.15 maskmean

```
mean of the masked values of X
```

69.16 masknanmean

69.17 mean1

```
mean of x
```

69.18 mean_man

```
mean and standard error of X
```

69.19 mse

```
mean squared error of residual vector res
this is de-facto the std for an unbiased residual
```

69.20 nanautocorr_man1

```
autocorrelation of a vector with nan-values
```

69.21 nanautocorr_man2

```
autocorrelation of a vector with nan-values
```

69.22 nanautocorr_man4

```
compute autocorrelation for x if x is a vector, or indivvidually
    for the
columns of x if x is a matrix
box jenkins 2008 eq. 2.1.12
TODO nan is problematic!
Note that it is faster to compute the acf in frequency space
as done in the matlab internal function
```

69.23 nancorr

```
(co)-correlation matrix when samples a NaN
```

69.24 nancumsum

```
cumulative sum, setting nan values to zero
```

69.25 nanlmean

```
mean of the l-th power of the absolute value of x
```

69.26 nanr2

```
coefficient of determination when samples are invalid
```

69.27 nanrms

```
root mean square value when sample contains nan-values
```

69.28 nanrmse

```
root mean square error from vector of residuals
this is de-facto the std for an unbiased residual
```

69.29 nanserr

```
standard error of x with respect to mean when x contains nan values
```

69.30 nanwmean

```
weighted mean
min_x sum w (x-mu)^2 => mu = sum(wx)/sum(w)
varargin can be dim
function [mu serr] = nanwmean(w,x)
```

69.31 nanwstd

```
weighed standard deviation
```

69.32 nanwvar

```
weighted variance of columns, corrected for degrees of freedom (
    bessel)

s^2 = sum(w*(x-sum(wx)/sum(w))^2)/sum(w)
```

69.33 nanxcorr

69.34 pearson

```
pearson correlation coefficient
```

69.35 pearson_to_kendall

```
conversion of pearson to kendall correlation coefficient
c.f. Kruskal 1958
```

69.36 pool_samples

```
pooled mean and standard deviation of several groups of different
    size, mean and standard deviation
```

69.37 qmean

```
trimmed mean
```

69.38 range_mean

69.39 rmse

```
root mean square error computed from a residual vector
this is de-facto the std for an unbiased residual
```

69.40 serr

```
standard error of the mean of a set of uncorrelated samples
```

69.41 serr1

69.42 test_qskew

69.43 test_qstd_qskew_optimal_p

69.44 wautocorr

```
autocorrelation for x if x is a vector, or indivvidually for the
columns of x if x is a matrix
samples can be weighted

c.f. box jenkins 2008 eq. 2.1.12

c.f. autocorr_man4

Note that it is faster to compute the acf in frequency space
as done in the matlab internal function
```

69.45 wcorr

```
correlation of two vectors when samples are weighted
```

69.46 wcov

```
covariance of two vectors when samples are weighted
```

69.47 wdof

```
effective degrees of freedom for weighted samples
```

69.48 wkurt

```
kurtosis with weighted samples
```

69.49 wmean

```
weighted mean

min_x sum w (x-mu)^2 => mu = sum(wx)/sum(w)

varargin can be dim
function [mu serr] = wmean(w,x)
```

69.50 wrms

```
weighted root mean square error
```

69.51 wserr

```
weighted root mean square error
```

69.52 wskew

```
skewness of a weighted set of samples
```

69.53 wstd

```
weighed standard deviation
```

69.54 wvar

```
weighted variance of columns, corrected for degrees of freedom (
    bessel)
variance of the weighted sample mean of samples with same mean (but
     not necessarily same variance)
s^2 = sum (w^2(x-sum(wx)^2))

s2_mu : error of mean, s2_mu : sd of prediction
```

70 statistics

70.1 nangeomean

70.2 nangeostd

```
geometric standard deviation ignoring nan-values
```

71 statistics/nonparametric-statistics

71.1 kernel1d

```
X   : ouput x axis bins
xi  : samples along x
m   : number of bins in X
fun : kernel function
pdf : propability density of xi
```

71.2 kernel2d

```
kernel density estimate in two dimensions
```

72 statistics

72.1 normmoment

```
expected norm of x.^n, when values x in x are iid normal with mu
    and sigma
```

72.2 normpdf2

```
pdf of the bivariate normal distribution
```

73 statistics/order-statistics

73.1 hodges_lehmann_location

```
hodges lehman location estimator

Asymptotic rms efficency of location estimte:
     mean:          1 s/sqrt(n)
     hodges lehman: sqrt(pi/3)*s ~ 1.0233 s/sqrt(n)
     median:        pi/2 s/sqrt(n) ~ 1.25 s / sqrt(n)
```

73.2 kendall

```
kendall correlation coefficient
```

73.3 kendall_to_pearson

```
convert kendall rank correlation coefficient to the person product
    moment
correlation coefficient

c.f. Kruska, 1985
```

73.4 mad2sd

```
transform median absolute deviation to standard deviation
for normal distributed values
```

73.5 madcorr

```
proxy correlation by median absolute deviation
```

73.6 median2_holder

73.7 median_ci

```
median and its confidence intervals under assumption of normality
se_me = sqrt(1/2 pi) 1.25331 * sd/sqrt(n)
```

73.8 median_man

```
median and confidence intervals
c is a P value for the confidence interval,
default is 0.95 (2-sigma)
median of the colums of X
```

73.9 mediani

```
index of median, if median is not unique, any of the values is
    chosen
```

73.10 nanmadcorr

```
proxy correlation by median absolute deviation
```

73.11 nanwmedian

```
weighted median, skips nan-values
```

73.12 nanwquantile

```
weighted quantile, skips nan values
```

73.13 oja_median

```
two dimensional oja median
note: the multivariate median is not unique

oja 1983, for extension to multivariate function, see chaudhri
```

73.14 qkurtosis

```
kurosis computed for quantiles

Note : this is a measurement of shape-tailedness and yields the
    same value for the
      normal distribution as "kurtosis"
      However, this is a separate statistic and hence requires
          different
      methods for calculating P-values and hypothesis testing
```

73.15 qmoments

```
moments estimated from quantiles
```

73.16 qskew

```
skewness estimated from quantiles

Note : this is a measurement of shape-symmetry and yields the same
    value for the
       skew-normal distribution as "skewness"
       However, this is an own statistic and hence requires
           different
       methods for calculating P-values and hypothesis testing
```

73.17 qskewq

```
skewness estimated by quantiles
```

73.18 qstdq

```
proxy standard deviation determined by quantiles
```

73.19 quantile1_optimisation

73.20 quantile2_breckling

```
qunatile regression
```

73.21 quantile2_chaudhuri

```
quantile regression
```

73.22 quantile2_projected

```
quantile in two dimensions
```

73.23 quantile2_projected2

```
spatial qunatile for chosen direction
```

73.24 quantile_envelope

73.25 quantile_regression_simple

```
simple quantile regression
```

73.26 ranking

```
ranking for spearman statistics
```

73.27 spatial_median

```
c.f. Oja 2008
is this the same as the oja simplex median (c.f. small 1990)?
```

73.28 spatial_quantile

```
spatial quantile
```

73.29 spatial_quantile2

```
spatial quantile
```

73.30 spatial_quantile3

```
spatial quantile
```

73.31 spatial_rank

```
unsigned rank
```

73.32 spatial_sign

```
spatial sign
```

73.33 spatial_signed_rank

```
signed rank
Note: this is only a true rank if X is normal with zero mean,
    abitrary variance
```

73.34 spearman

```
spearman's product moment coefficient
```

73.35 spearman_rank

73.36 spearman_to_pearson

```
conversion of spearman rank to person product moment correlation
    coefficient
```

73.37 wmedian

```
weighted median
```

73.38 wquantile

```
weighted quantile
```

74 statistics

74.1 qstd

74.2 quantile_extrap

75 statistics/random-number-generation

75.1 laplacernd

```
random number of laplace distribution
```

75.2 randc

```
correlate to correlated standard normally distributed vectors
```

75.3 skewrnd

```
random numbers of the skew normal distribution
```

75.4 skewrnd2

```
random numbers of the skew normal distribution
```

76 statistics

76.1 range

```
mid range
```

76.2 resample_with_replacement

77 statistics/resampling-statistics/@Jackknife

77.1 Jackknife

```
class for leave out 1 (delete 1) Jackknife estimates

note 1 : the 1-delete jackknife does not yield consistend estimates
     for all functions,
       in particular it will perform poorly on robust estimation
           functions
       this is overcome by the d-delete jacknife, where d has to
           exceed the breakdown point
       of the estimating function, for example sqrt(n) for the
           median
       as this leads to unreasonably large number of repetitions,
           bootstrap
       is recommended for large sample cases (or blocking for
           sequential data)
note 2 : as a linearisation, jackknife underestimates the error
    variance in case of
         dependence in the data
note 3 : studentisation and the leave out 1 jackknife are related
note 4 : the double 1 sample jacknife performs iferior to the d1
    jacknife
```

77.2 estimated_STATIC

```
jacknife estimate of mean, bias and standard error
theta0 : estimate from all samples
thetad : set of estimates obtained by leaving out one data point
    each
         last dimension of theta is assumed to be the jackknife
             dimension
```

77.3 matrix1_STATIC

```
matrix of estimation for leaving out two samples at a time
```

77.4 matrix2

```
matrix of estimations for jacknive with two samples left out
```

78 statistics/resampling-statistics

78.1 block_jacknife

78.2 jackknife_moments

```
moments determined by the jacknife

func : function of interest on the samples (e.g. mean)
A    : parameter matrix
       columns : parameters
       rows    : samples of the parameter sets
d   : number of samples left out
```

78.3 moving_block_jacknife

```
blocked Jacknfife for autocorrelated data
sliding block, statistically more efficient but computationally
    expensive
note, number of blocks must be sufficiently large h ~ sqrt(n)? << n
```

78.4 randblockserr

```
standard error of sequentilly correlated data by blocking
block length should be sufficiently larger than correlation length
and sufficiently smaller than data length
```

```
this uses a sliding block approach, which reduces the variation of
    the error estimate
TODO this does not work, randomly picking samples does not reveal
    the correlation
```

78.5 resample

```
resample a vector and apply function to it

TODO, should be with replacement

n  : number of samples
m  : number of subsamples
cx : maximum number of combinations
```

79 statistics

79.1 scale_quantile_sd

```
scale factor for the standard deviation
of the asymtpotic distibution of sample quantiles
(for normal distribution)
see cadwell, 1952
```

79.2 skewpdf

```
skew-normal distribution
c.f. Azzalini 1985
```

79.3 trimmed_mean

```
trimmed mean
```

79.4 ttest2_man

```
two-sample t-test
here posix return value standard: h = 0 accepted, h = 1 failed
note: the matlab logic is inverse : h = 1 accepted, h = 0 failed
two sided univariate t-test
```

79.5 ttest_man

```
two-sample t-test
unequal sample size
equal variance
```

79.6 ttest_paired

```
paired t-test
unequal sample size
equal variance
more powerfull than unpaired test, as long as correlation between
    x1 and x2 > 0
```

79.7 wharmean

```
weighted harmonic mean
```

80 wavelet

80.1 contiuous_wavelet_transform

```
continuous wavelet transform
follows "The Illustrated Wavelet Transform Handbook: Introductory
    Theory and ..."
```

80.2 cwt_man

```
continuous fourier transform
as of time of implmentation, the matlab interal cwt is affected by
serious round-off errors and has issues with the scaling,
which is not the case here
```

80.3 example_wavelets

80.4 phasewrap

```
wrap the phase to +/- pi
```

80.5 test_cwt_man

80.6 test_phasewrap

80.7 test_wavelet

80.8 test_wavelet2

80.9 test_wavelet_analysis

80.10 test_wavelet_reconstruct

80.11 test_wtc

80.12 wavelet

```
wavelet windows
```

80.13 wavelet_reconstruct

```
iverses wavelet transform for single frequency
(reconstruction of time series)
n : window lengths in multiples of filter period 1/f0
```

80.14 wavelet_transform

```
wavelet transform for single frequency
n : window lengths in multiples of filter period 1/f0
```

81 mathematics

```
mathematical functions of various kind
```

81.1 wrapphase