Manual for Package: root
Revision 6:11M

Karl Kästner

March 28, 2020

Contents

1 root

```
Root folder of the source code belonging to the doctoral thesis:

"Multi-Scale Monitoring and Modelling of the Kapuas River Delta",
    Karl K\"aster, 2019,

and master thesis:

"Computing the Spectrum of the Confined Hydrogen Atom", Karl K\"
    astner, 2012.

Copyright (C) 2010-2019 Karl K\"astner

Installation instructions:
```

```
1) Install Matlab
2) Install subversion (svn) and add subversion to the search path,
    so that
   it can be called from Matlab
3) Checkout this umbrella-project:
   svn checkout https://github.com/karlkastner/root/trunk root/
4) Start Matlab
5) Change into this directory ('root/')
6) Run the Matlab script "startup" located in this directory

The script then fetches the sub-repositories and adds them to the
Matlab search path

Note:

The code upload is work in progress, more parts will be subsequently
documented, added and tested.

This is experimental code. Use it wisely and at your own risk.

Licence:

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses
    />.
```

1.1 ROOTFOLDER

1.2 addpath_recursive

```
 recursively add a directory and sub-directories to the Matlab
     search path
 call restoredefaultpath to undo this
```

2 lib/auxiliar

2.1 Expanding_Double

3 lib/auxiliar/adaptor

`adators for backward compatibility for renamed files`

3.1 Keller

3.2 MMesh

3.3 SMesh

3.4 Slg

4 lib/auxiliar

4.1 arabic2roman

4.2 autocat

4.3 bplus

4.4 btimes

4.5 centre_axis

4.6 circshift_fractional

4.7 cmap_rolling

4.8 colormap3

4.9 colormap_byr

4.10 copy_fields

4.11 copyfields_deep

4.12 count_occurence

4.13 cummax

4.14 cummean

4.15 cumstd

4.16 cumvar

4.17 cvec

4.18 diag3

4.19 down

4.20 dspace

4.21 field_range

4.22 finite

4.23 flat

4.24 frac

4.25 getfield_deep

4.26 getout

4.27 hashcode

4.28 imagesc_

4.29 innerspace

5 lib/auxiliar/io/@IniFile

5.1 IniFile

6 lib/auxiliar/io

6.1 Stream

6.2 catXML

6.3 csv2cell

6.4 filewrite

7 lib/auxiliar/io/netcat

7.1 nc

7.2 nc_read_row

7.3 nc_read_sequential

7.4 nc_read_sequential_column

7.5 nc_readall

7.6 nc_writeall

8 lib/auxiliar/io

8.1 parseXML

8.2 printdef

8.3 printf

8.4 save_

8.5 xml2struct

9 lib/auxiliar

9.1 isfield_deep

9.2 isprop_deep

9.3 issym

9.4 iterate_cell

9.5 jmemory

9.6 leftdiff

9.7 leftmean

9.8 limits

9.9 linspace_man

9.10 linspace_man2

9.11 logspace_trimmed

9.12 matlab_messages

9.13 maxid

```
index of maximum
if value is not required (e.g. use in other functions such as
    accummarray)
```

9.14 memsize

9.15 mlint_all

9.16 myhot

9.17 none

9.18 objcopy

10 lib/auxiliar/plot

10.1 addx

10.2 addy

10.3 adjust_quiver_arrowhead_size

10.4 area_man

10.5 arrow

10.6 axis_equal_man

10.7 candlestick_man

10.8 circle

10.9 cmap

10.10 colormap_man

10.11 colormap_man2

10.12 colormap_man_old

10.13 columnlegend

10.14 copyaxes

10.15 datetick_man

10.16 daytick

10.17 dcolormap

10.18 dots

10.19 errorarea

10.20 errorarea2

10.21 errorbar_man

10.22 errorlines

10.23 fetchsubplot

10.24 fillmarker

10.25 freezeColors

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

 getCDataHandles -- get handles of all descendents with indexed
     CData
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 getParentAxes -- return handle of axes object to which a given
     object belongs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 checkArgs -- Validate input arguments
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

10.26 get_coordinates

10.27 hatch

10.28 hline

10.29 hold_color

10.30 hourspace

10.31 hourtick

10.32 interpplot

10.33 legendtitle

10.34 line_fewer_markers

```
 find marker spec in varargin and remove it; extract special params:
      LockOnMax,Spacing
 input size check
 a) once only the line with all points with the style
 b) last time the markers, using fewer points with style
 c) once with a visible handle, only the first point, using the
     complete style you specified
'x' -> marker delta-x constant; 'curve' : spacing constant along the
     curve length
```

10.35 monthspace

10.36 monthtick

10.37 mycolourmap

10.38 namedfigure

10.39 nansurf

10.40 nmcolormap

10.41 patch_man

10.42 pdfprint

10.43 percenttick

10.44 plot2svg

```
" height="100%%" viewBox="0 0 %0.3f %0.3f" ',paperpos(3),paperpos(4)
    );
          fprintf(fid,' <filter x="%0.3f%%" y="%0.3f%%" width="%0.3
              f%%" height="%0.3f%%" id="%s">\n', 0, 0, 100, 100,
              filterId);
          %    fprintf(fid,' <filter x="%0.3f%%" y="%0.3f%%" width
              ="%0.3f%%" height="%0.3f%%" id="%s">\n', -(offset *
              100), -(offset * 100), 100 + (offset * 200), 100 + (
              offset * 200), filterId);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SUBFUNCTIONS %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 Octave keeps s, d, p and h in the HandleGraphics object, for the
     square, diamond, pentagram, and hexagram markers, respectively
     -- Jakob Malm
 Octave keeps s, d, p and h in the HandleGraphics object, for the
     square, diamond, pentagram, and hexagram markers, respectively
     -- Jakob Malm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

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

10.45 plot_ellipse

10.46 plot_style

10.47 plotshaded

10.48 ploty4

10.49 plotyyy

10.50 quadsurf

10.51 quadsurf2

10.52 quadsurf3

10.53 quiver3_man

10.54 quiver_man

10.55 quiver_man2

10.56 quiver_man3

10.57 rectangles

10.58 scaleplot

10.59 setfontsize

10.60 shade_night

10.61 splitfigure

10.62 turtle

10.63 velplot

10.64 vline

10.65 vline_man

10.66 weekspace

10.67 weektick

10.68 xtick

10.69 xticklabel

10.70 ytick

10.71 yticklabel

11 lib/auxiliar

11.1 relpos

11.2 rightdiff

11.3 rmfield_optional

11.4 rvec

11.5 select

11.6 setfield_deep

11.7 setfields

11.8 sign2str

11.9 signs

11.10 simplifyignore

11.11 str_cell_reverse_index

12 lib/auxiliar/strings

12.1 chomp

12.2 chomp1

12.3 num2str_log10

12.4 num2str_power_10

12.5 strjoin

12.6 strsplit_man

12.7 suffix

13 lib/auxiliar

13.1 struct2obj

13.2 struct_avg

13.3 struct_flat

13.4 structcopy_deep

13.5 structfun_deep

13.6 sub2ind_man

13.7 subsall

13.8 swap

14 lib/auxiliar/system

`emulate POSIX and BASH functions`

14.1 alloc

14.2 basename

14.3 cbrt

14.4 dirname

14.5 head

14.6 head_str

14.7 tail

14.8 tail_str

15 lib/auxiliar

15.1 toInt32

15.2 unique_columnwise

15.3 unpack_struct

15.4 unwrap_periodic

15.5 up

15.6 zoomaxis

16 lib/gis

16.1 GPX

16.2 batavia_zero

17 lib/gis/centreline/@Centreline

17.1 Centreline

17.2 channel_planimetry

17.3 clip

17.4 connect_graph

17.5 curvature

17.6 cut

17.7 determine_width

17.8 distance

17.9 export_cross_section

17.10 export_node

17.11 export_shp

17.12 find_nearest_segment

17.13 from_polygon

17.14 from_shp

17.15 get

17.16 init

```
obj.seg_S(id(end)) = NaN;
```

17.17 init_connect

17.18 init_node_D

17.19 link_centreline

17.20 plot

17.21 plot_connection

17.22 prune

17.23 prune_leaves

17.24 prune_manually

17.25 reachable

17.26 remove_duplicate_points

17.27 resample

17.28 routing

17.29 routing2

17.30 shp_resample_simple

17.31 snmesh

17.32 squeeze

17.33 trim_ends

17.34 weighed_connection_matrix

17.35 xy2sn

18 lib/gis/centreline/@Segment

18.1 Segment

18.2 build_inverse_index

18.3 connectivity_matrix

18.4 init_seg_id

19 lib/gis/centreline

19.1 sn2xy_quadratic

19.2 thalweg

19.3 xy2sn_quadratic

20 lib/gis

20.1 gpx_export_csv

20.2 hgt_plot

20.3 hgt_read

```
% [ floor(mednan(z(kk))) meannan(z(kk)) min(z(kk)) max(z(kk)) ]
```

20.4 hgt_read_all

20.5 hgt_resample

20.6 nmeatime

21 lib/gis/shapefile/@Shp

21.1 Shp

21.2 area

21.3 buffer

21.4 cat

21.5 clip

21.6 clip_rect

21.7 close_polygon

21.8 concat

21.9 connect_network

```
 TODO make unique
 attach segments to
XY = [cvec(shp.X),shp.;
 knnsearch for nearest n neighbours
 for each segment
```

21.10 contour

21.11 cp

21.12 create

21.13 curvature

21.14 cut

21.15 diameter

21.16 edges

21.17 export_geo

21.18 export_gpx

21.19 export_gpx_track

21.20 export_ldb

21.21 export_poly

21.22 export_sdf

21.23 export_spline

21.24 extract_coastline

21.25 first_point

21.26 flat

21.27 generate_four_colour_index

21.28 import_geo

21.29 import_poly

21.30 join_lines

21.31 last_point

21.32 length

21.33 length2

21.34 line2point

21.35 link_lines

21.36 make_clockwise

21.37 merge

21.38 merge2

21.39 padd_nan

21.40 plot

21.41 points

21.42 polygon_boundary

21.43 read

21.44 readZ

21.45 remove_duplicate_points

21.46 remove_leaves

21.47 remove_nan

21.48 remove_polygon_closure

21.49 remove_short_elements

21.50 renumber

21.51 resample

21.52 resample_2

21.53 resample_min

21.54 resample_quick

21.55 scale

21.56 segment

21.57 select_for_refinement

21.58 set_geometry

21.59 set_resolution

21.60 skip

21.61 smooth

21.62 split_jump

21.63 split_line

21.64 split_nan

21.65 swap_hemisphere

21.66 translate

21.67 write

22 lib/gis/shapefile

22.1 astar_multi

22.2 astar_recursive

```
astar path finding algorithm
```

22.3 edge_chain

22.4 edge_from_bnd

22.5 preload_shp

22.6 read_gpx

22.7 shapewrite__

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses
    />.
-*- texinfo -*-
@deftypefn {Function File} {@var{status} =} shapewrite (@var{shpstr
    }, @var{fname})
Write contents of map- or geostruct to a GIS shape file.

@var{shpstr} must be a valid mapstruct or geostruct, a struct array
     with an
entry for each shape feature, with fields Geometry, BoundingBox,
    and X and Y
(mapstruct) or Lat and Lon (geostruct). For geostructs, Lat and Lon
     field
data will be written as X and Y data. Field Geometry can have data
    values
of 'Point', 'MultiPoint', 'Line', or 'Polygon', all case-
    insensitive. For
each shape feature, field BoundingBox should contain the minimum
    and maximum
(X,Y) coordinates in a 2x2 array [minX, minY; maxX, maxY]. The X
    and Y
fields should contain X (or Latitude) and Y (or Longitude)
    coordinates for
each point or vertex as row vectors; for polylines and polygons
    vertices of
each subfeature (if present) should be separated by NaN entries.

@var{fname} should be a valid shape file name, optionally with a '.
    shp'
suffix.

shapewrite produces 2 or 3 files, i.e. a .shp file (the actual
    shape file),
a .shx file (index file), and if @var{shpstr} contained additional
    fields,
a .dbf file (dBase type 3) with the contents of those additional
    fields.

@var{status} is 1 if the shape file set was written successfully, 0
otherwise.

@seealso{shaperead, shapeinfo}
```

```
@end deftypefn
Author: Philip Nienhuis <prnienhuis@users.sf.net>
Created: 2014-12-30
Input validation
Assess shape variable type (oct or ml/geo ml/map)
Yep. Find out what type
Assume it is an Octave-style struct read by shaperead
Assume it is a Matlab-style mapstruct
Assume it is a Matlab-style geostruct
Not a supported struct type
Check file name
Later on bname.shx and bname.dbf will be read
Prepare a few things
Change Lat/Lon fields into X/Y
Only now (after input checks) open .shp and .shx files & rewind
    just to be sure
Write headers in .shp & .shx (identical). First magic number 9994 +
     5 zeros
In between here = filelength in 16-bit words (single). For .shx it’
    s known
Next, shp file version
Shape feature type
Bounding box. Can be run later for ML type shape structs. Fill with
     zeros
Skip to start of first record position
Write shape features one by one
Write record start pos to .shx file
Write record contents
Point
Record index number
Record length (fixed)
Shape type
Simply write XY cordinates
MultiPoint
Record index number
Record length
Shape type
Bounding box
Nr of points
Polyline/-gon
Record index number
Prepare multipart polygons
Augment idx for later on, & this trick eliminates trailing NaN rows
Record length
Shape type
Bounding box
Number of parts, number of points, part pointers
Write file length into .shp header
Close files
```

```
Check for dbfwrite function
Write rest of attributes
Attributes + shp data in mapstruct
Attributes + shp data in geostruct
```

22.8 shapewrite_man

22.9 shp2geo

22.10 shp2kml

22.11 shp_plot_attribute

22.12 split_section

22.13 write_polygon

23 lib/instrumentation/adcp/@ADCP

23.1 ADCP

```
ADCP superclass
converts ADCP fixed integer raw data to floats with SI units
provides functions for ADCP data manipulation
calculated from the water temperature and sound frequency
```

23.2 Ds

```
depth of bin, distance between water surface (z_s) and (z_i)

Ds = z_s - z_bin

does not correct for tilts
```

23.3 Dt

```
projected distance from transducer to cell centres
if the instrument is not tilted, this is the vertical distance (
    depth)
between the transducer and cell centres
does not account for transducer depth
```

23.4 R

```
unprojected (slanted) distance between the transducer and cell
    centres
```

23.5 adc_current_slope

```
instrument type specific slope for converting raw current to Ampere
c.f WorkHorse Commands and Output Data Format, March 2014
c.f. XMT Voltage and Current Channels
originally undoccumented by RDI, and taken from Shields 2010
```

23.6 adc_voltage_slope

```
instrument series specific conversion factors for voltage
c.f. WorkHorse Commands and Output Data Format, March 2014
c.f. XMT Voltage and Current Channels
originally undoccumented by RDI, and taken from Shields 2010
```

23.7 assign_file

```
ensemble indices of each file
```

23.8 assign_water_level

```
assign water level to adcp ensembles (combine gauge with boat data)
```

23.9 average_profile

```
average backscatter for each sample within an specific interval
```

23.10 backscatter2ssc

```
wrapper for backscatter conversion
```

23.11 binsize

```
bin size (vertical distance between two bins)
```

23.12 blnk

```
blanking range, range from transduce to centre of first bin
```

23.13 btrange

```
convert raw btrange to vertical distance (projected distance) of
    the bed
level below the transducer, when the transducer is looking
    vertically down
this is the depth less the transducer depth
```

23.14 calc_backscatter

```
backscatter from echo intensity
```

23.15 clock_offset_STATIC

```
dt : median difference between adcp clock and UTC
sd_dt : standard error of dt
```

23.16 convert_raw_binprops_STATIC

```
convert the raw bin properties to si-units
```

23.17 convert_raw_serial_STATIC

```
convert bytes of serial number into single number
big endian system
```

23.18 convert_raw_time_STATIC

```
convert measurement time stamps into matlab internal format
```

23.19 convert_raw_velocity

```
convert scaled integer raw velocity to float SI (m/s)
```

23.20 convert_raw_velocity_STATIC

```
convert raw velocity to SI units (m/s)
```

23.21 copy

```
copy constructor
```

23.22 distmidbin1

```
convert raw distance to first bin centre to SI
```

23.23 file_ensemble_index

```
ensemble index eid_f with respect to file for ensemble eid
```

23.24 file_index

```
first and last ensemble index of of a file
```

23.25 filetime_min

```
start time of each file
```

23.26 fill_coordinate_gaps

```
fill gaps in ensemble coordinates
```

23.27 filter_range

```
filter HADCP velocity by detecting the last valid bin
if the bacscatter does not decreas over 10 bins, than obtstacle or
    intersection
```

23.28 heading_rad

```
convert raw instrument heading angle to [rad]
```

23.29 instrument_depth_m

```
depth of instrument (for submerged deployments)
```

23.30 instrument_to_ship_STATIC

```
transform velocities from instrument reference to ship reference
by correcting for pitch_rad and roll_rad

input
vel   : float [arbitrary unit] instrument reference
btvel : float [arbitrary unit] instrument reference
pitch_rad : float [radians] true pitch_rad, not measured pitch_rad
roll_rad : float [radians]

output
vel and btvel [input unit] ship reference
```

23.31 lngthtranspulse

```
convert raw transmit pulse length to SI units (m)
```

23.32 load_RSSI_values_STATIC

```
load instrument specific backscatter conversion parameters
```

23.33 nbins

```
number of bins for each file
```

23.34 near_field_correction

```
new fiel correction of the acoustic backscatter
c.f. wall 2006
Psi : (nr,1) near field correction factor
```

23.35 nens

```
number of ensembles
```

23.36 pitch_rad

```
convert raw pitch to radians
```

23.37 pressure_bar

```
convert raw pressure to bar
```

23.38 range2binid

```
convert distance to transducer to bin index
```

23.39 roll_rad

```
convert raw instrument roll angle to [rad]
```

23.40 rotate_velocity

```
rotate the velocity in the horizontal plane with respect to the
    directional
vector dir
dir : direction of the transect
```

23.41 rotate_velocity_sw

```
rotate velocity to local streamwise reference
input velocity can have arbitrary reference
```

23.42 ship_to_earth_STATIC

```
converts velocity from ship to earth coordinate reference
expects input arguments informat:
vel   : float arbitrary unit
btvel : float same unit as vel
heading_rad: float [radians]
```

23.43 sort_STATIC

```
sort files by start time
```

23.44 squeeze_STATIC

```
cut ensembles, skip ensembles or average ensembles in time

adcp : adcp structure
dt   : time between output ensembles in seconds
mode : {'average', 'skip'}
mask : selection of ensembles to keep (computed from dt if not
    provided)
                         fprintf(1,'Progress: %g\n%% %gs\n',idx/
                             nt,tlast);
```

23.45 temperature_offset_C

```
instrument specific temperature offset
```

23.46 to_abs

```
velocity magnitude
```

23.47 transducer_temperature_C

```
convert raw transducer temperature to SI units [Celsius]
T   : (1,nt) water temperature
```

23.48 verify_pc_time

```
verify the time stored in the data file
```

24 lib/instrumentation/adcp/@Ensemble

24.1 Ensemble

```
container for ADCP ensemble data and properties
```

24.2 calc_beamcoords

```
claculate positions in world coordinates where the individual beams
     hit the bottom
```

25 lib/instrumentation/adcp/@HADCP

25.1 HADCP

```
coverts raw data of horizontal ADCPs into physical quantities
and provides functions for data processing
```

25.2 beam_to_instrument_STATIC

```
transform the 3 beam velocities into a set of 2 orthogonal
    velocities
and 1 error velocity
This uses always three beams (no two beam solutions)

input
vel      : float [arbitrary unit] beam reference system
btvel    : float [arbitrary unit] beam reference system
beamangle : float [radians]

output
vel and btvel [input unit] instrument reference system
```

25.3 bootstrap_backscatter

```
bootstrap uncertainty of the backscatter parameters
```

25.4 calc_beam_spreading_cone

```
beam spreading
Note: beams spread in the form of bessel functions
      this is the engineering approach as cones, which is however
         not
      a good approximation, it is better to approximate it as a
         gaussian
```

25.5 calc_bin_coordinates

```
get the cartesian (world) coordinates of the HADCP central beam
    bins
```

25.6 calibrate_backscatter

```
calibrate backscatter to sediment concentration by the method of
    Sassi
```

25.7 filter_velocity

```
filter outliers in velocity data
```

25.8 firmware_fix_STATIC

```
correct RDI HADCP firmware bug (2014)
this bug successively invalids data every 4th-bin, which led to 3-
    beam solutions
and consequentially jumps of the transformed velocities
                     vel_s(:,fdx,:) = vel(:,fdx,:);
                     vel_b(:,fdx,:) = vel(:,fdx,:);
```

25.9 fixnan

```
interpolate invalid bin-samples from last and next ensemble
```

25.10 instrument_to_beam_STATIC

```
transform the 3 beam velocities into a set of 2 orthogonal
    velocities
and 1 error velocity
This uses always three beams (no two beam solutions)

input
vel      : float [arbitrary unit] beam reference system
btvel    : float [arbitrary unit] beam reference system
beamangle : float [radians]

output
vel and btvel [input unit] instrument reference system

mode : beams used for all transformations
      123, 12, 23, 13
```

25.11 reorder_velocity_STATIC

```
reorder the HADCP velocity data into the first three slots, the
    HADCP
has just three beams, but the software stores data for
four beams, similar to the four beam VADCPs
```

25.12 to_beam_STATIC

```
wrapper for conversion to beam velocity
Note that back-conversion to beam velocity is not unique in case of
     3 beam
solutions (as RDI instruments doe not store which beams were used)
    and
if instrument internal bin-mapping is used (whichs precise
    algorithm remains
an RDI secret)
```

25.13 to_earth_STATIC

```
wrappter to transform velocities to world coordinate reference
```

25.14 to_instrument_STATIC

```
wrapper to convert velocity to instrument coordinate reference
```

25.15 to_ship_STATIC

```
wrapper for conversion to ship velocity
```

26 lib/instrumentation/adcp/@RDI_mmt

26.1 RDI_mmt

26.2 read

26.3 write

27 lib/instrumentation/adcp/@VADCP

27.1 VADCP

```
coverts raw data of vertical ADCPs into physical quantities
```

27.2 assign_transect

```
assign transect index to ensembles
```

27.3 backscatter_report

```
human readable output of calibration properties
%       fprintf(['Parameters and uncertainty with respect to 95%%
   confidence\n']);
```

27.4 beam_to_instrument_STATIC

```
transform the 4 beam velocities into a set of 3 orthogonal
    velocities
and 1 error velocity

input
vel       : float [arbitrary unit] beam reference system
btvel     : float [arbitrary unit] beam reference system
beamangle : float [radians]

output
vel and btvel [input unit] instrument reference system

TODO account for NaNs either by three beam solution or
    interpolation
```

27.5 bottom_track_STATIC

```
compute bottom track coordinates
```

27.6 bscalibrate

```
backscatter to sediment calibration

calibtation subroutine
M_ref : sediment concentration calibration values
d_k   : depth of virtual reference value K
       (choose close to receiver, but out of near field, e.g.
           within 2m .. 4m)
TODO : better documentation of input values
TODO : rename nk into ik, bacause it is an index and not a length
TODO rename r_ref and d_k into r_1 and r_2
```

27.7 bsgrid

```
evaluate the objective function at the selected points
```

27.8 bsinvert

```
backscatter inversion
```

27.9 bsjackknife

```
compute the jackknife estimates of the parameters and their
    covariances
```

27.10 bsjointcalibration

```
calibrate backscatter
```

27.11 btvel_from_position

```
determine boat velocity from bottom track, inverse of bottom track
```

27.12 calc_ssc

```
calculate the backscatter
```

27.13 cdf

```
compute and plot cumulative distribution (cdf) of the velocity
    components
```

27.14 convert_nFiles

```
convert coordinates of NMEA-nFiles
```

27.15 correct_coordinates

```
correct the bottom coordinates for pitch and roll
```

27.16 correct_for_platform_velocity_STATIC

```
correct for platform (boat) velocity, this is the negative bed
    velocity
```

27.17 depth_average_velocity

```
average the velocity over depth
```

27.18 depth_integrate

```
depth integrate the velocity to obtain specific discharge
```

27.19 depth_integrate_sediment_discharge

```
depth integrated sediment discharge
```

27.20 filter_velocity

```
filter the velocity data
```

27.21 fit_sediment_concentration_profile

```
fit_suspended_sediment_concentration_profile(obj,profile_cls,
    ensmask,nwin)
```

27.22 fit_velocity_profile

```
fit velocity profile to the streamwise velocity
```

27.23 map_z

```
z-mapping, i.e. correct for roll and pitch of instrument
```

28 lib/instrumentation/adcp/@VADCP/old

28.1 assign_crossing

29 lib/instrumentation/adcp/@VADCP

29.1 optstr

```
string of arguments, for file name generation
```

29.2 plot_track

```
plot the boat track
```

29.3 plot_velocity_components

```
plot the velocity components
```

29.4 process

```
process VADCP data
```

29.5 range2depth

```
depth below transducer for individual bins of the beams
```

29.6 rangemask

```
mask all bins in range
```

29.7 to

```
transform velocity to given reference
```

29.8 to_beam_STATIC

```
convert velocity data to beam reference
```

29.9 to_cs

```
transform velocity to cross section references
cs-velocity is here defined as the velocity orthogonal to the cs
      % [ 0 1][c -s]=[-s c]
      % [ 1 0][s  c] [ c s]
```

29.10 to_earth_STATIC

```
transform coordinates to cartesian world reference system (earth)
```

29.11 to_sw

```
transform velocity with respect to depth averaged streamwise
    velocity
```

29.12 velocity_near_bed

```
velocity near the bed
```

29.13 xy2nts

```
project coordinates onto a single cross section and assign them nz-
    coordinates at a single cross section
TODO this should be part of transect
```

30 lib/instrumentation/adcp

```
adcp : processing of Acoustic Doppler Current Profiler (ADCP) data

Processing in 3 Levels:

Level 0 : Read in of raw-data (externally provided by ADCPtools,
    Vermeulen et al.)

Level 1 : VADCP, HADCP, SPADCP
       - convert raw data to CI units (m,s,kg)
       - transform velocities to arbitrary coordinate references
```

```
       - depth averaging and integration
       - fit velocity profiles
       - convert backscatter to suspended sediment concentration

Level 2 : CrossSection
       - interpolate and integrate for cross sections

Instruction:
       see and run saggau/sanggau_process_discharge for a working
           example
       to process VADCP discharge at a non-tidal station
```

30.1 ADCP_Bin

```
ADCP bin (single velocity values)
```

30.2 SPADCP

```
stream pro acoutic current doppler profiler
```

31 lib/instrumentation/adcp/backscatter/@Backscatter

31.1 Backscatter

```
acoustic backscatter processing
```

31.2 backscatter2ssc

```
convert backscatter to suspended sediment concentration
c.f lee hanes / sassi, with linear relation for reference
    concentration
```

31.3 backscatter2ssc_implicit

```
convert backscatter to suspended sediment concentration

this is the methog called "implicit" by hanes, though it is here
    still
implemented in an explicit way, as "explicit/imlicit" in hanes only
mean euler forward or trapezoidal integration
```

31.4 backscatter2ssc_implicit_sample

```
convert backscatter to suspended sediment concentration, implicit
    method
```

31.5 backscatter2ssc_sample

```
convert backscatter 2 suspended sediment concentration
```

31.6 backscatter2ssc_sassi

```
convert backscatter to suspended sediment concentration
c.f. sassi
```

31.7 backscatter2ssc_sassi_sample

```
convert backscatter to suspended sediment concentration
c.f. sassi
```

31.8 fit

```
fit backscatter coefficients

function [res, leverage, w, obj] = fit(obj,ssc0,R0,R,bs,last,param0
    )

ssc0          - ns x 1, reference concentration
R0           - ns x 1, distance to sample along beam
bs           - ns x nbin, backscatter profile per sample
R            - ns x nbin, distance to bin from transducer along beam
last         - last : index last valid bin
param0       - initial value for parameters
```

31.9 regmat

```
regression matrix
```

32 lib/instrumentation/adcp/backscatter

32.1 attenuation_coefficient

```
accoustic attenuation coefficient of suspended particles

hanes 2012

[d_mm] = mm
[f]    = Hz = 1/s
[as]   = 1/m (neper)
for db : chi_db = 8.7 chi_neper
[M]    = kg/m^3 = mg/l

for normalization : chis = as(M=2650)

function [as,asnu,ass,X,chi] = attenuation_coefficient(d_mm,f,M,
    mode)
```

32.2 backscatter_coefficient

```
analytic determination of the backscatter coefficient
```

32.3 backscatter_coefficient_2

```
analytic basckatter coefficient
thorne 2002
thorne 2012
```

32.4 backscatter_form_function

```
acoustic backscatter form function
```

32.5 backscatter_to_concentration

```
convert acoustic backscatter to suspended sediment mass
    concentration
backscatter S has to be corrected for attenuation
```

32.6 backscatter_to_concentration2

```
convert acoustic backscatter to sediment concentration
```

32.7 derive_attenuation_coefficient

32.8 normalized_particle_radius

```
normalized particle radius
```

32.9 scattering_cross_section_general

```
acoustic cross sectin ? of sediment particles
Medwin, ch. 7.5.3
Axially Symmetric Spherical Mode Solutions
```

32.10 sigma_geometric

```
differential cross section
geometrical backscattering for spherical bodies
ka >> 1, large particles or high frequencies
k : wave number
a : radius of the particle
```

32.11 sigma_rayleigh

```
Rayleigh scattering for a sphere (ka << 1)
small particles or low frequencies
Medwin 7.5.2 Rayleigh Scatter From a Sphere (ka << 1)
```

32.12 ssc2backscatter

```
convert suspended sediment concentration to backscatter
function bs = ssc2backscatter(ssc,d_mm,f,varargin)

ssc : mass concentration of sediment [ssc] = g/l = kg/m^3
d_mm : grain size diameter [d_mm] = mm
f : frequency [f] = Hz = 1/2
```

33 lib/instrumentation/adcp/cross-section/@ADCP_Transect

33.1 ADCP_Transect

```
zero dimensional processing of ADCP data
no resampling, meshing or gridding
```

33.2 assign_to_transect

```
assign ensemble to respective transects
this has a side-effect (writes to) the adcp object,
but values of induvidial cross sections remain unaffected by each
    other
```

33.3 compare

```
discharge summary
```

33.4 detect_crossings

```
detect consecutive navigation of transects (channel crossings)
```

33.5 detect_crossings_circling

```
separatate individual navigation of transects,
for cases when the boat goes in circles and crosses the branches
    one after
the other before returning to the original cross section,
```

thus the boat does not turn at the other bank to return across the same section
and always navigates the cross section in the same direction

33.6 detect_crossings_returning

groups the ensembles into transects,
one transect is defined as all ensembles recorded during the time the boat
moved from one bank to the other (return is defined as separate transect)

33.7 detect_rounds

detect rounds, i.e. when boat returns to initial position

33.8 export_mmt

export RDI mmt

33.9 extrapolate_to_bank

extrapolate values to bank

33.10 fit

33.11 integrate_discharge

integrate discharge

```
Q  = sum q
q  = A_n*u_s = h dn us
   = h * [dx, dy]*[-v; u]
   = h * dt * [-ub, -vb] * [-v; u]
```

```
note that uvb * dt is usually more accurate than dx of GPS position
    ,
if uvb determined by doppler shift of ADCP bottom echo,
except when the GPS position (or velocity) is determined from the
    carrier frequency

note that projection can be left out, if cs is defined with
    transect individual end points,
but not recommended, if there are strong secondary currents as
    encountered at
bends or bifurcations
```

33.12 plot

```
plot the transect as a line in cartesian coordinates
```

33.13 plot2d

```
plot transects
```

33.14 plot_rounds

```
plot rounds (consecutiver transects) navigated with the boat
```

34 lib/instrumentation/adcp/cross-section/@CrossSection

34.1 CrossSection

```
Level-3 ADCP data processing, projection to cross section and
    integration/averaging
```

34.2 calc_auxiliary_quant

```
compute auxiliary quantities
```

34.3 compare

```
interpolate for all cross-sections the values to the same time-slot
for comparison
```

34.4 determine_time_slots

```
split data set into specific time slots
```

34.5 discharge

```
integrate the discharge over all finite elements of the cross
    section
```

34.6 extrapolate_S

```
extrapolate missing values along the vertical
```

34.7 extrapolate_backscatter

```
extrapolate the backscatter
```

34.8 extrapolate_backscatter_2d_STATIC

```
extrapolate backscatter to bed, surface and banks
```

34.9 extrapolate_bed_profile

```
extrapolate bed profile to channel banks
```

34.10 extrapolate_n

```
extrapolate value beyond end of cross section
```

34.11 extrapolate_velocity

```
extrapolate the velocity to the bank, bed, and surface
```

34.12 extrapolate_velocity_1d_STATIC

```
extrapolate depth averaged velocity
```

34.13 extrapolate_velocity_2d_STATIC

```
extrapolate velocity to banks, surface and bottom
TODO, this is only applicable for Grid2
```

34.14 fit_bathymetry_2d

34.15 fit_bed_profile

```
fit the bed profile, has to precede n-z meshing of the cross-
    section
```

34.16 fit_cross_section

```
fit the optimal cross section as the main axis of the transect
by regressing a line through the measurement points in the x-y
    plane

y = c0 + c1 x
```

34.17 fit_vertical_profile_of_velocity

```
fit vertical profile of the streamwise velocity

this function will work with both ensenble data, eg. U_bin taken
    from ensembles,
```

```
as well as gridded data, (U_bin taken from the velocity grid)

input
cs    : struct       : cross section averaged data
U_bin : [nrow x ncol] : vertical profiles of stream wise velocity
Z_bin : [nrow x ncol] : positions of bin above bottom for each
    element in U_bin
ens.N : [ncolx1]     : position of each column of U in along the
    cross section
ens.H : [ncolx1]     : depth of each column of U
ens.sH : [ncolx1]     : std of depth at each colum of U
ens.ldx : [ncolx1]   : last valid sample in column of U
dw_z0  : scalar   : grid cell size for grid_n
obj.roughnessmethod        : method to use for the computation
output:
grid_n : struct     : function of u_s and z_0 along cross section
us_ens, ln_z0_ens, U_ens : local estimates for input ensembles/grid
     columns
          not returned by every obj.roughnessmethod
```

34.18 fit_water_level

```
fit water level from depth measurement
this works only if the ADCP is stationary
```

34.19 generate_mesh_tn

```
generate 1+1D mesh over time and across section
```

34.20 generate_mesh_tnz

```
generate a t-n-z mesh
```

34.21 optstr

```
string of options, for file name generation
```

34.22 plot_n_quiver

```
plot quiver across section
```

34.23 plot_nz

```
plot along n and z
```

34.24 plot_nz_quiver

```
quiver plot of velocity across section
```

34.25 plot_tn

```
plot over time and across channel
```

34.26 plot_xyz

```
plot values in "val" in the 2D cross section, where the cartesian
rather than the local coordinates of the cross-section are used
```

34.27 process_backscatter

```
process backscatter, i.e. fit to cross-section grid from bin-values
```

34.28 process_backscatter_tn

```
      process depth integrated backscatter over time t and acrross
          section N
note: backscatter is processed as flux
       due to high concentration and backscatter near the bottom,
      the inner rpoduct of the discharge and concentration
      \bar u \bar c_s is not a good estimate of the
     depth averaged sediment flux \bar{u c_s}
```

34.29 process_backscatter_tnz

```
process the backscatter in 2+1D (time, across channel and along
    vertical)
```

34.30 process_discharge

```
process the discharge
```

34.31 process_velocity_tn

```
process the velocity data
```

34.32 process_velocity_tnz

```
process velocity data in 2+1D (time, across-section and along
    vertical)
```

34.33 summarise

```
summarize discharge of cross section
```

34.34 var_n

```
return value stored in field "fieldname" at position "N" in the
    cross section
```

34.35 var_t

```
return value stored in filed "fieldname" at time t
cross sectionally integrated or averaged value
```

34.36 var_tn

```
return values of field "fieldname" at time t and position N along
    cross section
typically depth integrated or averaged values
```

34.37 var_tnz

```
generically return value stored in field "fieldname" at time t and
    position N
```

35 lib/instrumentation/adcp/cross-section

35.1 complete_profiles

```
fill gaps in profiles
assumes profile to be constant in time, this is not true
for tidal flow in compound cross sections and near banks
```

35.2 define_transect

```
gui user selection of cross-section end points
```

35.3 discharge_division

```
discharge division ratio
```

35.4 discharge_summary

```
compute and store discharge summary
             q_tn = cs.q_tn(ti);
             ndx      = abs(N)<=Nlim;
             Qi       = cs.dw*sum(q_tn)';
             Qi_centre = cs.dw*sum(q_tn(ndx,:))';
             Q = [Q; Qi];
             Q_centre = [Q_centre; Qi_centre];
```

35.5 load_vadcp_discharge

```
load previously computed vadcp discharge (auxiliary function for
    plotting)
this function stacks data from several vadcp reference measurements
     into one structure
```

```
This assumes that all data sets where processed with the same
    settings
```

35.6 split_transect2

36 lib/instrumentation/adcp/hadcp/@HADCP_Discharge

36.1 HADCP_Discharge

```
superclass for HADCP discharge estimation methods
```

36.2 fit

```
fit the model parameter for HADCP discharge prediction,
estimate errors with the Jacknife method
```

37 lib/instrumentation/adcp/hadcp/@HDischarge

37.1 Hbin

37.2 calc_specific_discharge_weights

```
calculate unite discharge weights
```

37.3 estimate_discharge

```
integrate and scale specifc discharge to total discharge for each
    ensemble
```

38 lib/instrumentation/adcp/hadcp/@HIVM

38.1 HIVM

```
Index velocity method of Horizontal ADCP data
```

39 lib/instrumentation/adcp/hadcp/@IVM

39.1 IVM

```
index velocity method
```

40 lib/instrumentation/adcp/hadcp

40.1 ESM

40.2 ESM_individual

40.3 SDM

```
Specific Discharge Method
upscale specific discharge to cross sectionally integrate discharge
    ,
 than average
this method is provenly less accurate than averaging before
    upscaling
```

40.4 VPM

```
velocity profile method
correct individual bin velocities for vertical velocity profile
    variation,
then averagem, then upscale to cross sectionally integrated
    discharge
```

40.5 hadcp_homogenize_profile

```
homogenize the hadcp profile
```

40.6 hadcp_homogenize_profile2

```
homogenise the horizontal velocity profile
```

40.7 wavg

```
weighted average ?
```

40.8 wavg_mean

```
weighted average
```

40.9 wopt

```
optimal weights for averaging (lumped) velocities that are each
    associated
with error variance s2
```

41 lib/instrumentation/adcp

```
adcp : processing of Acoustic Doppler Current Profiler (ADCP) data

Processing in 3 Levels:

Level 0 : Read in of raw-data (externally provided by ADCPtools,
    Vermeulen et al.)

Level 1 : VADCP, HADCP, SPADCP
        - convert raw data to CI units (m,s,kg)
        - transform velocities to arbitrary coordinate references
        - depth averaging and integration
        - fit velocity profiles
        - convert backscatter to suspended sediment concentration
```

```
Level 2 : CrossSection
       - interpolate and integrate for cross sections

Instruction:
       see and run saggau/sanggau_process_discharge for a working
           example
       to process VADCP discharge at a non-tidal station
```

41.1 smooth_track

```
smooth a repeatedly navigated (circular) track to produce and
    idealized
average track
```

41.2 streawise_velocity

```
rotate ensembles in stream direction (transverse velocity
    integrates to zero)
```

42 lib/instrumentation/adcp/test

42.1 example_backscatter_coefficient_2

42.2 test_backscatter_coefficient

42.3 test_bedslope

42.4 test_delta_z_correction

42.5 test_depth_range

42.6 test_linearisation

42.7 test_procTrans_vele

42.8 test_rotvel

42.9 test_sanggau_load_bed_level_2016

42.10 test_sanggau_rc

43 lib/instrumentation/adcp

```
adcp : processing of Acoustic Doppler Current Profiler (ADCP) data

Processing in 3 Levels:

Level 0 : Read in of raw-data (externally provided by ADCPtools,
    Vermeulen et al.)

Level 1 : VADCP, HADCP, SPADCP
       - convert raw data to CI units (m,s,kg)
       - transform velocities to arbitrary coordinate references
       - depth averaging and integration
       - fit velocity profiles
       - convert backscatter to suspended sediment concentration

Level 2 : CrossSection
       - interpolate and integrate for cross sections

Instruction:
       see and run saggau/sanggau_process_discharge for a working
           example
       to process VADCP discharge at a non-tidal station
```

43.1 zztransform

```
non-linear mapping for bin coordinates when depth averages between
    ensembles
for avaraging several ensembles

      preserve discharge w int u_avg dz = int int u dz dn = Q
      perserve shear stress is the same (u_avg)^2_s = mean((u_s)^2)
      preserve sediment transport w int u_avg c_avg dz = int int u
          c dz dn
     preserve rouse number

      alternative : correct parameters for effects of averaging

several approaches :
s-transform : z_1' = H0/H1 z_1, perserves u_bar
                             does not preserve u_* (du/dz|_0)
clipping   : z_1' = z_1, z_1 < H0, does not preserve u_bar
                                  unclear if H0>H1
                                perserves (du/dz)_0 (u_*)
zz-transform : perserve both u_bar and u_
TODO this is non-monotoneous when difference in H0 and H1 is large
```

44 lib/instrumentation/pressure-gauge/@PressureGauge

44.1 PressureGauge

44.2 apply_corrections

```
obj.time(fdx) = NaN;
```

44.3 assign_upstream_km

44.4 check_filetime

44.5 estimate_altitude_transducer

44.6 export_csv

44.7 filter

44.8 merge

44.9 readIDC

44.10 readTxt

44.11 resample

44.12 stft

44.13 wavelet_transform

45 lib/instrumentation/sonar/@Sonar

45.1 Sonar

45.2 cat

45.3 compare

45.4 complete

45.5 equalise_echo

45.6 export_shp

45.7 export_table

45.8 export_xyz

45.9 from_data

45.10 from_dep

45.11 from_sl2

45.12 from_slg

45.13 from_slg2txt

45.14 remove

45.15 select

45.16 to_metric

46 lib/instrumentation/sonar

46.1 DEP

46.2 DGPS

46.3 sortfiles

46.4 test_loadSLG

47 lib/mathematics/calendar

47.1 days_per_month

47.2 isnight

48 lib/mathematics

```
mathematical functions of various kind
```

48.1 cast_byte_to_integer

```
 cast byte to integer
```

49 lib/mathematics/complex-analysis

```
operations on complex numbers
```

49.1 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
```

49.2 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
```

49.3 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
```

49.4 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
```

49.5 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
```

49.6 root_complex

```
root of a complex number
```

49.7 test_imroots

50 lib/mathematics/derivation

```
derivation of several functions by means of symbolic computation
```

50.1 derive_acfar1

50.2 derive_ar2param

50.3 derive_arc_length

50.4 derive_fourier_power

50.5 derive_fourier_power_exp

50.6 derive_laplacian_curvilinear

50.7 derive_laplacian_fourier_piecewise_linear

50.8 derive_logtripdf

50.9 derive_smooth1d_parametric

51 lib/mathematics/derivation/master

51.1 derive_bc_one_sided

51.2 derive_convergence

51.3 derive_error_fdm

51.4 derive_fdm_poly

51.5 derive_fdm_power

51.6 derive_fdm_taylor

51.7 derive_fdm_vargrid

51.8 derive_fem_2d_mass

51.9 derive_fem_error_2d

51.10 derive_fem_error_3d

51.11 derive_fem_sym_2d

51.12 derive_grid_constants

51.13 derive_interpolation

51.14 derive_laplacian

51.15 derive_limit

51.16 derive_nc_1d

51.17 derive_nc_1d_

51.18 derive_nc_2d

51.19 derive_nonuniform_symmetric

%

51.20 derive_richardson

51.21 derive_sum

51.22 nn

51.23 test_derive

51.24 test_derive_fdm_poly

51.25 test_filter

51.26 test_vargrid

52 lib/mathematics/derivation

```
derivation of several functions by means of symbolic computation
```

52.1 simplify_atan

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

53 lib/mathematics

```
mathematical functions of various kind
```

53.1 exp10

54 lib/mathematics/finance

54.1 derive_skewrnd_walsh_paramter

54.2 gbm_cdf

54.3 gbm_fit

54.4 gbm_fit_old

54.5 gbm_inv

54.6 gbm_mean

54.7 gbm_median

54.8 gbm_pdf

54.9 gbm_simulate

54.10 gbm_skewness

54.11 gbm_std

54.12 gbm_transform_time_step

54.13 put_price_black_scholes

54.14 skewgbm_simulate

54.15 skewrnd_walsh

55 lib/mathematics/finance/test

55.1 test_gbm

55.2 test_gbm_pdf

55.3 test_skewrnd_walsh

56 lib/mathematics/fourier/@STFT

56.1 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
```

56.2 itransform

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

56.3 stft_

```
static wrapper for STFT
```

56.4 stftmat

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

56.5 transform

```
short time fourier transform
```

57 lib/mathematics/fourier

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

57.1 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 (!)
```

57.2 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
```

57.3 example_fourier_window

57.4 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
```

57.5 fft_man

```
fast fourier transform for complex input data

input:
F : data in real space

output :

F : fourier transformation of F
```

57.6 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
```

57.7 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
```

57.8 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
```

57.9 fourier_cesaro_correction

57.10 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
```

57.11 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
```

57.12 fourier_coefficient_ramp3

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

57.13 fourier_coefficient_ramp_pulse

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

57.14 fourier_coefficient_ramp_step

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

57.15 fourier_coefficient_square_pulse

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

57.16 fourier_cubic_interaction_coefficients

57.17 fourier_derivative

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

57.18 fourier_expand

```
expand values of fourier series
```

57.19 fourier_fit

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

57.20 fourier_interpolate

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

57.21 fourier_matrix

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

57.22 fourier_matrix2

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

57.23 fourier_matrix3

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

57.24 fourier_matrix_exp

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

57.25 fourier_multiplicative_interaction_coefficients

57.26 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
```

57.27 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
```

57.28 fourier_predict

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

57.29 fourier_quadratic_interaction_coefficients

57.30 fourier_range

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

57.31 fourier_regress

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

57.32 fourier_resampled_fit

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

57.33 fourier_resampled_predict

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

57.34 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
```

57.35 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
```

57.36 hyperbolic_fourier_box

57.37 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)
```

57.38 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
```

57.39 nanfft

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

57.40 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)
```

57.41 roots_fourier

```
zeros of continuous fourier series series

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

57.42 spectral_density

```
spectral density
```

57.43 test_complex_exp_product

57.44 test_fourier_filter

57.45 test_idftmtx

58 lib/mathematics/geometry/@Geometry

58.1 Geometry

58.2 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
```

58.3 arclength_old

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

58.4 arclength_old2

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

58.5 base_point

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

58.6 base_point_limited

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

58.7 centroid

```
centroid pf a polygone
```

58.8 cosa_min_max

58.9 cross2

```
cross product in two dimensions
```

58.10 curvature

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

58.11 ddot

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

58.12 distance

```
equclidan distance between two points
```

58.13 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
```

58.14 dot

```
dot product
```

58.15 edge_length

```
edge length
```

58.16 enclosed_angle

```
angle enclosed between two lines
```

58.17 enclosing_triangle

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

58.18 hexagon

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

58.19 inPolygon

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

58.20 inTetra

```
flag points contained in tetrahedron
```

58.21 inTetra2

```
flag points contained in tetrahedron
```

58.22 inTriangle

```
flag points contained in triangle
function [flag, c] = inTriangle(P1,P2,P3,P0)
```

58.23 intersect

```
intersect between two lines
```

58.24 lineintersect

```
intersect of two lines
```

58.25 lineintersect1

```
intersect of two lines
```

58.26 minimum_distance_lines

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

58.27 mittenpunkt

```
mittenpunkt of a triangle
```

58.28 nagelpoint

```
nagelpoint of a triangle
```

58.29 onLine

58.30 orthocentre

```
orthocentre of triangle
```

58.31 plumb_line

58.32 poly_area

```
area of a polygon
function A = poly_area(x,y)
```

58.33 poly_edges

```
edges of a polygon
```

58.34 poly_set

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

58.35 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
```

58.36 polyxpoly

```
intersections of two polygons
```

58.37 project_to_curve

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

58.38 quad_isconvex

58.39 random_disk

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

58.40 random_simplex

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

58.41 sphere_volume

```
volume of a sphere
```

58.42 tetra_volume

```
volume of a tetrahedron
```

58.43 tobarycentric

```
cartesian to barycentric coordinates
```

58.44 tobarycentric1

```
cartesian to barycentric coordinates
```

58.45 tobarycentric2

```
cartesian to barycentric coordinates
```

58.46 tobarycentric3

```
cartesian to barycentric coordinates
```

58.47 tri_angle

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

58.48 tri_area

```
angle of a triangle
```

58.49 tri_centroid

```
centroid of a triangle
```

58.50 tri_distance_opposit_midpoint

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

58.51 tri_edge_length

```
edge length of a triangle
```

58.52 tri_edge_midpoint

```
mid point of a triangle
```

58.53 tri_excircle

```
excircle of a triangle
```

58.54 tri_height

```
height of a triangle
```

58.55 tri_incircle

```
incircle of a triangle
```

58.56 tri_isacute

```
flag acute triangles
```

58.57 tri_isobtuse

```
flag obntuse triangles
```

58.58 tri_semiperimeter

```
semiperimeter of a triangle
```

58.59 tri_side_length

```
edge lenght of triangle
```

59 lib/mathematics/geometry

59.1 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
```

59.2 bounding_box

```
bounding box of X
```

59.3 curvature_1d

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

59.4 cvt

```
centroidal voronoi tesselation
```

59.5 deg_to_frac

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

59.6 ellipse

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

59.7 ellipseX

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

59.8 ellipseY

59.9 first_intersect

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

59.10 golden_ratio

```
golden ratio
```

59.11 hypot3

```
hypothenuse in 3D
```

59.12 meanangle

```
weighted mean of angles
```

59.13 meanangle2

```
mean angle
```

59.14 meanangle3

```
mean angle
```

59.15 meanangle4

```
mean angle
```

59.16 medianangle

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

59.17 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
```

59.18 pilim

```
limit to +- pi
```

59.19 streamline_radius_of_curvature

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

60 lib/mathematics/histogram/@Histogram

60.1 2x

60.2 Histogram

60.3 bimodes

60.4 cdf

60.5 cdfS

60.6 chi2test

60.7 cmoment

60.8 cmomentS

60.9 entropy

60.10 entropyS

60.11 iquantile

60.12 kstest

60.13 kurtosis

60.14 kurtosisS

60.15 mean

60.16 meanS

60.17 median

60.18 medianS

60.19 mode

60.20 modeS

60.21 moment

60.22 momentS

60.23 pdf

60.24 quantile

60.25 quantileS

60.26 setup

60.27 skewness

60.28 skewnessS

60.29 stairs

60.30 stairsS

60.31 std

60.32 stdS

60.33 var

60.34 varS

61 lib/mathematics/histogram

61.1 hist_man

61.2 histadapt

61.3 histconst

61.4 pdf_poly

61.5 plotcdf

61.6 test_histogram

62 lib/mathematics/linear-algebra

62.1 averaging_matrix_2

62.2 colnorm

```
norms of columns
```

62.3 condest_

```
estimation of the condition number
```

63 lib/mathematics/linear-algebra/coordinate-transformation

63.1 barycentric2cartesian

```
barycentric to cartesian coordinates
```

63.2 barycentric2cartesian3

```
convert barycentric to cartesian coordinates
```

63.3 cartesian2barycentric

```
cartesian to barycentric coordinates
```

63.4 cartesian_to_unit_triangle_basis

```
transform coodinates into unit triangle
```

63.5 ellipsoid2geoid

63.6 example_approximate_utm_conversion

63.7 latlon2utm

```
transform latitude and longitude to WGS84 UTM
```

63.8 latlon2utm_simple

63.9 lowrance_mercator_to_wgs84

```
convert lowrance coordinates to wgs84

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

63.10 nmea2utm

```
convert nmea messages to utm coordinates
```

63.11 sn2xy

```
convert sn to xy coordinates
```

63.12 unit_triangle_to_cartesian

```
transform coordinates in unit triangle to cartesian coordinates
```

63.13 utm2latlon

```
convert wgs84 utm to latitute and longitude
```

63.14 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
```

63.15 xy2sn

```
convert cartesian to streamwise coordiantes
```

63.16 xy2sn_java

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

63.17 xy2sn_old

```
transform points from cartesian into streamwise coordinates

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

64 lib/mathematics/linear-algebra

64.1 det2x2

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

64.2 det3x3

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

64.3 det4x4

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

64.4 diag2x2

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

64.5 eig2x2

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

65 lib/mathematics/linear-algebra/eigenvalue

65.1 eig_bisection

65.2 eig_inverse

65.3 eig_inverse_iteration

65.4 eig_power_iteration

66 lib/mathematics/linear-algebra/eigenvalue/jacobi-davidson

66.1 afun_jdm

66.2 davidson

66.3 jacobi_davidson

66.4 jacobi_davidson_qr

66.5 jacobi_davidson_qz

66.6 jacobi_davidson_simple

66.7 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'
```

66.8 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'
```

66.9 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
```

66.10 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
    ===========================================
%==========================================================================

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

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

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

66.11 mfunc_jdm

66.12 mgs

66.13 minres_

66.14 mv_jacobi_davidson

67 lib/mathematics/linear-algebra

67.1 first

67.2 gershgorin_circle

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

67.3 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
```

67.4 ieig2x2

```
reconstruct matrix from eigenvalue decomposition
```

67.5 inv2x2

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

67.6 inv3x3

67.7 inv4x4

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

68 lib/mathematics/linear-algebra/lanczos

68.1 arnoldi

68.2 arnoldi_new

68.3 eigs_lanczos_man

68.4 lanczos

68.5 lanczos_

68.6 lanczos_biorthogonal

68.7 lanczos_biorthogonal_improved

68.8 lanczos_ghep

68.9 mv_lanczos

68.10 reorthogonalise

68.11 test_lanczos

69 lib/mathematics/linear-algebra/linear-systems

69.1 gmres_man

```
break on convergence
```

69.2 minres_recycle

70 lib/mathematics/linear-algebra

70.1 lpmean

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

70.2 lpnorm

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

70.3 matvec3

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

70.4 max2d

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

70.5 mpoweri

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

70.6 mtimes2x2

70.7 mtimes3x3

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

70.8 nannorm

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

70.9 nanshift

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

70.10 nl

```
number rows (lines) of a matrix

analogue to unix nl command
```

70.11 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)
```

70.12 normalize1

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

70.13 normrows

70.14 orth2

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

70.15 orth_man

```
orthogonalize the columns of A
```

70.16 orthogonalise

```
make x orthogonal to Y
```

70.17 paddext

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

70.18 paddval1

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

70.19 paddval2

```
padd values to x
```

71 lib/mathematics/linear-algebra/polynomial

71.1 chebychev

```
chebycheff polynomials
```

71.2 piecewise_polynomial

```
evaluate piecewise polynomial
```

71.3 roots1

```
roots of linear functions
```

71.4 roots2

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

71.5 roots2poly

71.6 roots3

71.7 roots4

71.8 test_roots4

71.9 vanderi_1d

```
vandermonde matrix of an integral
```

72 lib/mathematics/linear-algebra

72.1 randrot

```
random rotation matrix
```

72.2 right

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

72.3 rot2

```
rotation matrix from angle
```

72.4 rot2dir

```
rotation matrix from direction vector
```

72.5 rot3

72.6 rotR

72.7 rownorm

72.8 simmilarity_matrix

72.9 spnorm

```
frobenius norm
```

72.10 spzeros

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

72.11 test_roots3

72.12 transform_minmax

72.13 transpose3

```
transpose stacked 3x3 matrices
```

72.14 transposeall

73 lib/mathematics/logic

```
bitwise operations on integers
```

73.1 bitor_man

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

input:
      A (positive integer)
```

74 lib/mathematics/master/plot

74.1 attach_boundary_value

74.2 cartesian_polar

74.3 img_vargrid

74.4 plot_basis_functions

74.5 plot_convergence

74.6 plot_dof

74.7 plot_eigenbar

74.8 plot_error_estimation

74.9 plot_error_estimation_2

74.10 plot_error_fem

74.11 plot_fdm_kernel

74.12 plot_fdm_vs_fem

74.13 plot_fem_accuracy

74.14 plot_function_and_grid

74.15 plot_hat

74.16 plot_hydrogen_wf

74.17 plot_mesh

74.18 plot_mesh_2

74.19 plot_refine

74.20 plot_refine_3d

74.21 plot_runtime

74.22 plot_spectrum

74.23 plot_wavefunction

75 lib/mathematics/master/ported

75.1 assemble_2d_dphi_dphi

75.2 assemble_2d_phi_phi

75.3 assemble_3d_dphi_dphi

75.4 assemble_3d_phi_phi

75.5 dV_2d_

75.6 derivative_2d

75.7 derivative_3d

75.8 element_neighbour_2d

75.9 prefetch_2d_

75.10 promote_2d_3_10

75.11 promote_2d_3_15

75.12 promote_2d_3_21

75.13 promote_2d_3_6

75.14 promote_3d_4_10

75.15 promote_3d_4_20

75.16 promote_3d_4_35

75.17 vander_2d

75.18 vander_3d

76 lib/mathematics/master/sandbox

76.1 adapt

76.2 assoc_laguerre

76.3 assoc_legendre

76.4 c23

77 lib/mathematics/master/sandbox/cg

77.1 cg

77.2 cg_coef_to_poly

77.3 errmat

77.4 lanczos

77.5 laplacian_2d

77.6 test_cg_eigs

77.7 test_lanczos

78 lib/mathematics/master/sandbox

78.1 condition_number_higher_order

78.2 confinement_dat

78.3 convergence_2d_3d

78.4 convergence_matrix_powers

78.5 cut_out

78.6 derivative_2d

78.7 derivative_3d

78.8 dummy

78.9 eig_error

78.10 eigs_fix

78.11 energy_level

78.12 equalise

78.13 example_int64

```
Basic operations
```

```
Matrix multiplication
Timing
```

79 lib/mathematics/master/sandbox/fem-matlab

79.1 boundary_circle

79.2 boundary_rectangle

79.3 geometry_circle_with_hole

79.4 geometry_rectangle

80 lib/mathematics/master/sandbox

80.1 fem_2d_estimate_error

80.2 fem_assemble_scratch

80.3 fem_s

80.4 fourier_h

80.5 grad_2d

80.6 grad_3d

80.7 gradient

80.8 harmonic_oscillator

80.9 hydrogen_2d_analytic

80.10 hydrogen_boxed

80.11 hydrogen_boxed_old

80.12 hydrogen_wave

```
% Hydrogen atom
```

80.13 hydrogen_wf

80.14 ichol_man

80.15 known_eigenvalue

80.16 kron_man

80.17 laguerre

80.18 laplacian_arbitrary_order_old

80.19 laplacian_convergence

80.20 laplacian_cut_out

80.21 laplacian_cylindrical

80.22 laplacian_non_uniform_old

80.23 laplacian_polar

80.24 laplacian_simple

80.25 lderivative_3d

80.26 list_dat

80.27 matlab-horner

80.28 mesh_to_grid_2d_3

80.29 mg_mat

80.30 mv

80.31 orth2

80.32 partial_derivative_2d

80.33 partition_function

80.34 partition_function_old

80.35 poisson

80.36 poisson_fem

80.37 potential

80.38 powerc

80.39 quick_newihbour

80.40 radial

80.41 radial_convergence

80.42 radial_wafefunction

80.43 refine_2d

80.44 refine_3d

80.45 relerr

80.46 restore_cw

80.47 runtime_bm

80.48 rydberg

80.49 s_old

80.50 snorm

80.51 spherical_harmonic

80.52 split_eig

80.53 sum1

80.54 sum3

81 lib/mathematics/master/sandbox/summation

81.1 acc

81.2 add

81.3 ape

81.4 mmul_accurately

81.5 sum_kahan

81.6 sum_pairwise

81.7 test_sum

82 lib/mathematics/master/sandbox

82.1 test_convergence_ill_conditioned

82.2 test_fem_1d

82.3 test_fem_2d

82.4 test_fem_3d

82.5 test_increase

82.6 test_lanczos_shift

82.7 test_ldl

82.8 test_power

82.9 trefethen_p8_fdm

82.10 wavefunc

82.11 xgrid

83 lib/mathematics/number-theory

83.1 ceiln

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

83.2 digitsb

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

83.3 floorn

```
floor to n-digits
```

83.4 iseven

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

83.5 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
```

83.6 nchoosek_man

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

83.7 pythagorean_triple

```
pythagorean triple
```

83.8 roundn

```
round to n digits
```

84 lib/mathematics/numerical-methods/differentiation

84.1 derivative1

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

84.2 derivative2

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

85 lib/mathematics/numerical-methods/finite-difference

85.1 cdiff

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

85.2 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
```

85.3 cmean

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

85.4 derivative_matrix_1_1d

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

85.5 derivative_matrix_2_1d

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

85.6 derivative_matrix_2d

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

85.7 derivative_matrix_curvilinear

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

85.8 derivative_matrix_curvilinear_2

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

85.9 difference_kernel

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

85.10 distmat

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

85.11 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
```

85.12 laplacian

85.13 laplacian_fdm

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

85.14 left

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

85.15 lrmean

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

86 lib/mathematics/numerical-methods/finite-difference/maste

86.1 fdm_adaptive_grid

86.2 fdm_adaptive_refinement_old

86.3 fdm_assemble_d1_2d

86.4 fdm_assemble_d2_2d

86.5 fdm_confinement

86.6 fdm_d_vargrid

86.7 fdm_h_unstructured

86.8 fdm_hydrogen_vargrid

86.9 fdm_mark_unstructured_2d

86.10 fdm_plot

86.11 fdm_plot_series

86.12 fdm_refine_2d

86.13 fdm_refine_3d

86.14 fdm_refine_unstructured_2d

86.15 fdm_schroedinger_2d

86.16 fdm_schroedinger_3d

86.17 relocate

87 lib/mathematics/numerical-methods/finite-difference

87.1 mid

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

87.2 pwmid

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

87.3 ratio

```
ratio of two subsequent values
```

87.4 steplength

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

87.5 swapoddeven

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

87.6 test_derivative_matrix_2d

87.7 test_derivative_matrix_curvilinear

87.8 test_difference_kernel

88 lib/mathematics/numerical-methods/finite-element

88.1 Mesh_2d_java

88.2 Tree_2d_java

88.3 assemble_1d_dphi_dphi

88.4 assemble_1d_phi_phi

88.5 assemble_2d_dphi_dphi_java

88.6 assemble_2d_phi_phi_java

88.7 assemble_3d_dphi_dphi_java

88.8 assemble_3d_phi_phi_java

88.9 boundary_1d

88.10 boundary_2d

88.11 boundary_3d

88.12 check_area_2d

88.13 circmesh

88.14 cropradius

88.15 display_2d

88.16 display_3d

88.17 distort

88.18 err_2d

88.19 estimate_err_2d_3

88.20 example_1d

88.21 example_2d

88.22 explode

88.23 fem_2d

88.24 fem_2d_heuristic_mesh

88.25 fem_get_2d_radial

88.26 fem_interpolation

88.27 fem_plot_1d

88.28 fem_plot_1d_series

88.29 fem_plot_2d

88.30 fem_plot_2d_series

88.31 fem_plot_3d

88.32 fem_plot_3d_series

88.33 fem_plot_confine_series

88.34 fem_radial

```
adaptive grid
constant grid
```

88.35 flip_2d

88.36 get_mesh_arrays

88.37 hashkey

89 lib/mathematics/numerical-methods/finite-element/int

89.1 int_1d_gauss

89.2 int_1d_gauss_1

89.3 int_1d_gauss_2

89.4 int_1d_gauss_3

89.5 int_1d_gauss_4

89.6 int_1d_gauss_5

89.7 int_1d_gauss_6

89.8 int_1d_gauss_lobatto

89.9 int_1d_nc_2

89.10 int_1d_nc_3

89.11 int_1d_nc_4

89.12 int_1d_nc_5

89.13 int_1d_nc_6

89.14 int_1d_nc_7

89.15 int_1d_nc_7_hardy

89.16 int_2d_gauss_1

89.17 int_2d_gauss_12

89.18 int_2d_gauss_13

89.19 int_2d_gauss_16

89.20 int_2d_gauss_25

89.21 int_2d_gauss_3

89.22 int_2d_gauss_33

89.23 int_2d_gauss_6

89.24 int_2d_gauss_7

89.25 int_2d_gauss_9

89.26 int_2d_nc_10

89.27 int_2d_nc_15

89.28 int_2d_nc_21

89.29 int_2d_nc_3

89.30 int_2d_nc_6

89.31 int_3d_gauss_1

89.32 int_3d_gauss_11

89.33 int_3d_gauss_14

89.34 int_3d_gauss_15

89.35 int_3d_gauss_24

89.36 int_3d_gauss_4

89.37 int_3d_gauss_45

89.38 int_3d_gauss_5

89.39 int_3d_nc_11

89.40 int_3d_nc_4

89.41 int_3d_nc_6

89.42 int_3d_nc_8

90 lib/mathematics/numerical-methods/finite-element

90.1 interpolation_matrix

90.2 mark

90.3 mark_1d

90.4 mesh_1d_uniform

90.5 mesh_3d_uniform

90.6 mesh_interpolate

90.7 neighbour_1d

90.8 old

90.9 pdeeig_1d

90.10 pdeeig_2d

90.11 pdeeig_3d

90.12 polynomial_derivative_1d

90.13 potential_const

90.14 potential_coulomb

90.15 potential_harmonic_oscillator

90.16 project_circle

90.17 project_rectangle

90.18 promote_1d_2_3

90.19 promote_1d_2_4

90.20 promote_1d_2_5

90.21 promote_1d_2_6

90.22 quadrilaterate

90.23 recalculate_regularity_2d

90.24 refine_1d

90.25 refine_2d_21

90.26 refine_2d_structural

90.27 regularity_1d

90.28 regularity_2d

90.29 regularity_3d

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

90.30 relocate_2d

90.31 test_circmesh

90.32 test_hermite

90.33 tri_assign_points

90.34 triangulation_uniform

90.35 vander_1d

```
van der Monde matrix
```

90.36 vanderd_1d

90.37 vanderi_1d

91 lib/mathematics/numerical-methods/finite-volume/@Advection

91.1 Advection

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

91.2 dot_advection

```
advection equation
```

92 lib/mathematics/numerical-methods/finite-volume/@Burgers

92.1 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
```

92.2 dot_burgers_fdm

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

92.3 dot_burgers_fft

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

93 lib/mathematics/numerical-methods/finite-volume/@Finite

93.1 Finite_Volume

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

93.2 apply_bc

```
apply boundary conditions
```

93.3 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);
```

93.4 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
```

93.5 step_unsplit

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

94 lib/mathematics/numerical-methods/finite-volume/@Flux_Limiter

94.1 Flux_Limiter

```
class of flux limiters
```

94.2 beam_warming

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

94.3 fromm

```
fromme limiter
low res
```

94.4 lax_wendroff

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

94.5 minmod

```
min-mod schock limiter
```

94.6 monotized_central

```
monotonized central flux limiter
```

94.7 muscl

```
muscl flux limiter
```

94.8 superbee

```
superbee limiter
```

94.9 upwind

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

94.10 vanLeer

```
van Leer limiter
```

95 lib/mathematics/numerical-methods/finite-volume/@KDV

95.1 dot_kdv_fdm

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

95.2 dot_kdv_fft

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

95.3 kdv_split

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

96 lib/mathematics/numerical-methods/finite-volume/@Reconstruct_

96.1 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;
```

96.2 advect_highres

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

96.3 advect_lowress

```
single time step
low resolution
```

97 lib/mathematics/numerical-methods/finite-volume

97.1 Godunov

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

97.2 Lax_Friedrich

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

97.3 Measure

97.4 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
```

97.5 fv_swe

```
wrapper for solving SWE
```

97.6 staggered_euler

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

97.7 staggered_grid

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

98 lib/mathematics/numerical-methods

98.1 grid2quad

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

99 lib/mathematics/numerical-methods/integration

99.1 cumintL

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

99.2 cumintR

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

99.3 int_trapezoidal

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

100 lib/mathematics/numerical-methods/interpolation/@Kriging

100.1 Kriging

```
class for Kriging interpolation
```

100.2 estimate_semivariance

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

100.3 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
```

101 lib/mathematics/numerical-methods/interpolation/@Regu

101.1 RegularizedInterpolator1

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

101.2 init

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

102 lib/mathematics/numerical-methods/interpolation/@Regu

102.1 RegularizedInterpolator2

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

102.2 init

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

103 lib/mathematics/numerical-methods/interpolation/@Regu

103.1 RegularizedInterpolator3

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

103.2 init

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

104 lib/mathematics/numerical-methods/interpolation

104.1 IDW

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

104.2 IPoly

```
polynomial interpolation class
```

104.3 IRBM

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

104.4 ISparse

```
sparse interpolation class
```

104.5 Inn

```
nearest neighbour interpolation
```

104.6 Interpolator

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

104.7 fixnan

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

104.8 idw1

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

104.9 idw2

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

104.10 inner2outer

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

104.11 inner2outer2

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

104.12 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
```

104.13 interp1_man

```
interpolate
```

104.14 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
```

104.15 interp1_slope

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

104.16 interp1_smooth

104.17 interp1_unique

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

104.18 interp2_man

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

104.19 interp_angle

```
interpolate an angle
```

104.20 interp_fourier

```
interpolation by the fourier method
```

104.21 interp_fourier_batch

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

104.22 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_);
```

104.23 interp_sn2

```
interpolation in streamwise coordinates
```

104.24 interp_sn3

104.25 interp_sn_

104.26 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)
```

104.27 resample1

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

104.28 resample_d_min

```
resample a function
```

104.29 resample_vector

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

104.30 test_interp1_limited

105 lib/mathematics/numerical-methods

105.1 inverse_complex

106 lib/mathematics/numerical-methods/ode

106.1 bvp2_check_arguments

106.2 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])
```

106.3 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
```

106.4 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])
```

106.5 bvp2wavetrain

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

106.6 bvp2wavetwopass

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

106.7 ivp_euler_forward

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

106.8 ivprk2

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

106.9 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]
```

106.10 ode2characteristic

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

106.11 step_trapezoidal

```
single trapezoidal step
```

106.12 test_bvp2

107 lib/mathematics/numerical-methods/optimisation

107.1 armijo_stopping_criterion

```
armijo stopping criterion for optimizations
```

107.2 astar

```
astar path finding alforithm
```

107.3 binsearch

```
binary search on a line
```

107.4 bisection

```
bisection
```

107.5 box1

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

107.6 box2

107.7 cauchy

107.8 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
```

107.9 directional_derivative

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

107.10 dud

```
optimization by the dud algorithm
```

107.11 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
```

107.12 extreme_quadratic

107.13 ftest

107.14 fzero_bisect

107.15 fzero_newton

107.16 grad

```
numerical gradient
```

107.17 hessian

```
numerical hessian
```

107.18 hessian_from_gradient

```
numerical hessian from gradient
```

107.19 hessian_projected

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

107.20 line_search

```
bisection routine
```

107.21 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
```

107.22 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
```

107.23 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
```

107.24 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
```

107.25 line_search_quadratic2

```
quadratic line search
```

107.26 line_search_wolfe

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

107.27 ls_bgfs

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

107.28 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
```

107.29 ls_generalized_secant

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

107.30 nlcg

```
non-linear conjugate gradient
input:
x   : nx1 start vectort
opt : struct options
fdx : gradient constraint
```

107.31 nlls

```
non-linear least squares
```

107.32 picard

```
picard iteration
```

107.33 poly_extrema

```
extrema of a polynomial
```

107.34 quadratic_function

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

107.35 quadratic_programming

```
optimize by quadratic programming
```

107.36 quadratic_step

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

107.37 rosenbrock

```
rosenbrock test function
```

107.38 sqrt_heron

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

107.39 test_directional_derivative

107.40 test_dud

107.41 test_fzero_newton

107.42 test_line_search_quadratic2

107.43 test_ls_generalized_secant

107.44 test_nlcg_6_order

107.45 test_nlls

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

108 lib/mathematics/numerical-methods/pde

108.1 laplacian2d_fundamental_solution

109 lib/mathematics/numerical-methods/piecewise-polynomials

109.1 Hermite1

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

109.2 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)
```

109.3 hp2_predict

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

109.4 hp_predict

```
predict with piecewise hermite polynomial
```

109.5 hp_regress

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

109.6 lp_count

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

109.7 lp_predict

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

109.8 lp_regress

109.9 lp_regress_

110 lib/mathematics/regression/@PolyOLS

110.1 PolyOLS

```
class for polynomial least squares
```

110.2 coefftest

110.3 detrend

```
detrending by polynomial regression
```

110.4 fit

```
fit a polynomial function
like polyfit, but returns parameter error estimates
TODO automatically activate scaleflag
```

110.5 fit_

```
fit a polynomial function
```

110.6 predict

```
predict polynomial function values
```

110.7 predict_

110.8 slope

```
slope by linear regression
```

111 lib/mathematics/regression/@PowerLS

111.1 PowerLS

```
class for power law regression
```

111.2 fit

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

111.3 predict

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

111.4 predict_

112 lib/mathematics/regression/@Theil

112.1 Theil

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

112.2 detrend

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

112.3 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
```

112.4 predict

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

112.5 slope

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

113 lib/mathematics/regression

```
linear and non-linear regression
```

113.1 Theil_Multivariate

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

113.2 areg

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

113.3 ginireg

```
gini regression
```

113.4 hesssimplereg

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

113.5 l1lin

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

113.6 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)
```

113.7 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
```

113.8 regression_method_of_moments

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

113.9 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
```

113.10 theil2

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

113.11 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
```

113.12 total_least_squares

```
total least squares
```

113.13 weighted_median_regression

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

114 lib/mathematics/set-theory

114.1 issubset

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

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

115 lib/mathematics/signal-processing

115.1 acf_effective_sample_size

```
effective sample size from acf
```

115.2 acf_genton

```
autocorrelation function
```

115.3 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
```

115.4 acfar1_2

```
autocorrelation of the ar1 process
```

115.5 acfar2

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

115.6 acfar2_2

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

115.7 ar1_cutoff_frequency

115.8 ar1_effective_sample_size

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

115.9 ar1_mse_mu_single_sample

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

115.10 ar1_mse_pop

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

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

115.11 ar1_mse_range

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

115.12 ar1_spectrum

```
spectrum of the ar1 process
```

115.13 ar1_to_tikhonov

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

115.14 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
```

115.15 ar1_var_factor_

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

115.16 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|
```

115.17 ar1delay

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

115.18 ar1delay_old

```
autocorrelation of the residual
```

115.19 ar2conv

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

115.20 ar2dof

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

115.21 ar2param

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

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

115.22 asymwin

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

115.23 autocorr_fft

```
autocorrelation function
```

115.24 bandpass

```
bandpass filter
```

115.25 bandpass2

```
bandpass filter
```

115.26 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
```

115.27 bartlett_spectrogram

```
bartlet spectrogramm
TODO sliding window
```

115.28 bin1d

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

115.29 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
```

115.30 binormrnd

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

115.31 conv1_man

```
convolutions with padding
```

115.32 conv2_man

```
convolution in 2d
```

115.33 conv2z

115.34 conv30

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

115.35 conv_

```
convolution of a with b
```

115.36 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
```

115.37 convz

115.38 cosexpdelay

115.39 csmooth

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

115.40 daniell_window

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

115.41 danielle_window

```
danielle fourier window
```

115.42 db2neper

```
convert decibel to neper
```

115.43 db2power

```
power ratio from db
```

115.44 derive_danielle_weight

115.45 derive_limit_0_acfar

115.46 detect_peak

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

115.47 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)
```

115.48 doublesum_ij

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

115.49 effective_sample_size_to_ar1

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

115.50 filt_hodges_lehman

115.51 filter1

```
filter along one dimension
```

115.52 filter2

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

115.53 filter_

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

115.54 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
```

115.55 filterp

115.56 filterp1

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

115.57 filterstd

115.58 firls_man

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

115.59 flattopwin

```
the flat top window
```

115.60 frequency_response_boxcar

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

115.61 freqz_boxcar

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

115.62 gaussfilt1

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

115.63 hanchangewin

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

115.64 hanchangewin2

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

115.65 hanwin

```
hanning filter window
```

115.66 hanwin_

```
hanning filter window
```

115.67 highpass

```
high pass filter
```

115.68 kaiserwin

```
kaiser filter window
```

115.69 kalman

```
Kalman filter
```

115.70 lanczoswin

```
Lanczos window
```

115.71 last

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

115.72 lowpass

```
low pass filter
```

115.73 lowpass2

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

115.74 lowpass_iir

```
iir-low pass
```

115.75 lowpass_iir_symmetric

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

115.76 lowpassfilter2

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

115.77 maxfilt1

115.78 meanfilt1

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

115.79 medfilt1_man

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

115.80 medfilt1_man2

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

115.81 medfilt1_padded

```
median filter with padding
```

115.82 medfilt1_reduced

```
median filter with padding
```

115.83 mid_term_single_sample

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

115.84 minfilt1

115.85 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
```

115.86 mysmooth

115.87 nanautocorr

```
autocorrelation with nan-values
```

115.88 nanmedfilt1

```
medfilt1, skipping nans
```

115.89 neper2db

```
convert neper to db
```

115.90 peaks_man

```
peaks of a periodogram
```

115.91 polyfilt1

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

115.92 qmedfilt1

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

115.93 randar1

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

115.94 randar1_dual

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

115.95 randar2

```
generate ar2 process
```

115.96 randarp

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

115.97 range_window

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

115.98 rectwin

```
rectangular window
```

115.99 recursive_sum

115.100 select_range

115.101 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
```

115.102 smooth2

```
smooth vectos of X
```

115.103 smooth_man

115.104 smooth_parametric

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

115.105 smooth_parametric2

```
parametrically smooth the curve
```

115.106 smooth_with_splines

115.107 smoothfft

```
filter with fast fourier transform
```

115.108 spectrogram

```
spectrogram
```

115.109 std_window

```
moving block standard deviation
```

115.110 sum_i_lag

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

115.111 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
```

115.112 sum_ii_

115.113 sum_ij

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

115.114 sum_ij_

115.115 sum_ij_partial_

115.116 sum_multivar

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

115.117 test_acfar1

115.118 test_acfar1_2

115.119 test_acfar1_3

115.120 test_acfar1_4

115.121 test_acfar2

115.122 test_ar1_var_factor

115.123 test_ar1_var_factor_2

115.124 test_ar1_var_mu_single_sample

115.125 test_ar1_var_pop

115.126 test_ar1_var_pop_1

115.127 test_ar1delay

115.128 test_bivariate_covariance_term

115.129 test_convexity

115.130 test_lanczoswin

115.131 test_madcorr

115.132 test_randar1

115.133 test_randar1_multivariate

115.134 test_randar2

115.135 test_sum_ij

115.136 test_sum_multivar

115.137 test_trifilt1

115.138 test_wautocorr

115.139 test_wavelet_transform

115.140 test_wordfilt

115.141 test_xar1_mid_term

115.142 tikhonov_to_ar1

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

115.143 trapwin

```
trapezoidal filter window
```

115.144 trifilt1

```
filter with triangular window
```

115.145 triwin

```
triangular filter window
```

115.146 triwin2

```
triangular filter window
```

115.147 varar1

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

115.148 welch_spectrogram

```
welch spectrogram
```

115.149 wfilt

```
filter with window
```

115.150 winbandpass

```
filter with bandpass
```

115.151 window_make_odd

115.152 winfilt0

```
filter with window
```

115.153 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
```

115.154 wmeanfilt

```
mean filter with window
```

115.155 wmedfilt

```
median filter with window
```

115.156 wordfilt

```
weighted order filter
```

115.157 wordfilt_edgeworth

```
weighed order filter
```

115.158 xar1

115.159 xcorr_man

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

116 lib/mathematics/sorting

116.1 sort2

```
sort two numbers
```

116.2 sort2d

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

117 lib/mathematics/special-functions

117.1 bessel_sphere

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

117.2 digamma_man

117.3 hankel_sphere

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

117.4 hermite

```
probabilistic's hermite polynomial by recurrence relation

input :
n : order
x : value

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

117.5 legendre_man

```
legendre polynomials
```

117.6 neumann_sphere

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

118 lib/mathematics/statistics

118.1 atan_s2

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

118.2 beta_mode_to_parameter

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

118.3 coefficient_of_determination

118.4 conditional_expectation_normal

118.5 correlation_confidence_pearson

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

119 lib/mathematics/statistics/distributions

119.1 PDF

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

119.2 binorm_separation_coefficient

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

119.3 binormcdf

```
bio-modal gaussian distribution
```

119.4 binormfit

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

119.5 binormpdf

119.6 edgeworth_cdf

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

119.7 edgeworth_pdf

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

119.8 logn_mode2param

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

119.9 logn_param2mode

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

119.10 lognpdf_

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

119.11 pdfsample

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

119.12 t2cdf

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

119.13 t2inv

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

120 lib/mathematics/statistics

120.1 example_standard_error_of_sample_quantiles

120.2 f_var_finite

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

120.3 gamma_mode_to_parameter

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

120.4 gaussfit3

120.5 gaussfit_quantile

120.6 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
```

120.7 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
```

121 lib/mathematics/statistics/information-theory

121.1 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
```

121.2 bayesian_information_criterion

```
bayesian information criterion
```

122 lib/mathematics/statistics

122.1 kurtncdf

122.2 kurtnpdf

122.3 kurtosis_bias_corrected

```
bias corrected kurtosis
```

122.4 limit

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

122.5 logfactorial

```
approximate log of the factorial
```

122.6 loglogpdf

122.7 lognfit_quantile

122.8 logskewcdf

122.9 logskewpdf

123 lib/mathematics/statistics/logu

123.1 lambertw_numeric

```
lambert-w function
```

123.2 logtrialtcdf

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

123.3 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))))
```

123.4 logtrialtmean

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

123.5 logtrialtpdf

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

123.6 logtrialtrnd

123.7 logtricdf

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

123.8 logtriinv

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

123.9 logtrimean

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

123.10 logtripdf

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

123.11 logtrirnd

123.12 logucdf

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

123.13 logucm

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

123.14 loguinv

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

123.15 logumean

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

123.16 logupdf

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

123.17 logurnd

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

123.18 loguvar

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

123.19 medlogu

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

123.20 test_logurnd

123.21 tricdf

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

123.22 triinv

```
inverse of the triangular distribution
```

123.23 trimedian

```
median of the triangular distribution
```

123.24 tripdf

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

123.25 trirnd

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

124 lib/mathematics/statistics

124.1 maxnnormals

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

124.2 midrange

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

124.3 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
```

124.4 mode_man

125 lib/mathematics/statistics/moment-statistics

125.1 autocorr_man3

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

125.2 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
```

125.3 autocorr_man5

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

125.4 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
```

125.5 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
```

125.6 corr_man

```
correlation of two vectors
```

125.7 cov_man

```
covariance matrix of two vectors
```

125.8 dof

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

125.9 edgeworth_quantile

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

125.10 effective_sample_size

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

125.11 f_correlation

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

125.12 f_finite

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

125.13 lmean

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

125.14 lmoment

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

125.15 maskmean

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

125.16 masknanmean

125.17 mean1

```
mean of x
```

125.18 mean_man

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

125.19 mse

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

125.20 nanautocorr_man1

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

125.21 nanautocorr_man2

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

125.22 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
```

125.23 nancorr

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

125.24 nancumsum

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

125.25 nanlmean

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

125.26 nanr2

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

125.27 nanrms

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

125.28 nanrmse

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

125.29 nanserr

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

125.30 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)
```

125.31 nanwstd

```
weighed standard deviation
```

125.32 nanwvar

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

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

125.33 nanxcorr

125.34 pearson

```
pearson correlation coefficient
```

125.35 pearson_to_kendall

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

125.36 pool_samples

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

125.37 qmean

```
trimmed mean
```

125.38 range_mean

125.39 rmse_

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

125.40 serr

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

125.41 serr1

125.42 test_qskew

125.43 test_qstd_qskew_optimal_p

125.44 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
```

125.45 wcorr

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

125.46 wcov

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

125.47 wdof

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

125.48 wkurt

```
kurtosis with weighted samples
```

125.49 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)
```

125.50 wrms

```
weighted root mean square error
```

125.51 wserr

```
weighted root mean square error
```

125.52 wskew

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

125.53 wstd

```
weighed standard deviation
```

125.54 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
```

126 lib/mathematics/statistics

126.1 nangeomean

126.2 nangeostd

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

127 lib/mathematics/statistics/nonparametric-statistics

127.1 kernel1d

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

127.2 kernel2d

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

128 lib/mathematics/statistics

128.1 normmoment

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

128.2 normpdf2

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

129 lib/mathematics/statistics/order-statistics

129.1 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)
```

129.2 kendall

```
kendall correlation coefficient
```

129.3 kendall_to_pearson

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

c.f. Kruska, 1985
```

129.4 mad2sd

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

129.5 madcorr

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

129.6 median2_holder

129.7 median_ci

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

129.8 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
```

129.9 mediani

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

129.10 nanmadcorr

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

129.11 nanwmedian

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

129.12 nanwquantile

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

129.13 oja_median

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

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

129.14 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
```

129.15 qmoments

```
moments estimated from quantiles
```

129.16 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
```

129.17 qskewq

```
skewness estimated by quantiles
```

129.18 qstdq

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

129.19 quantile1_optimisation

129.20 quantile2_breckling

```
qunatile regression
```

129.21 quantile2_chaudhuri

```
quantile regression
```

129.22 quantile2_projected

```
quantile in two dimensions
```

129.23 quantile2_projected2

```
spatial qunatile for chosen direction
```

129.24 quantile_envelope

129.25 quantile_regression_simple

```
simple quantile regression
```

129.26 ranking

```
ranking for spearman statistics
```

129.27 spatial_median

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

129.28 spatial_quantile

```
spatial quantile
```

129.29 spatial_quantile2

```
spatial quantile
```

129.30 spatial_quantile3

```
spatial quantile
```

129.31 spatial_rank

```
unsigned rank
```

129.32 spatial_sign

```
spatial sign
```

129.33 spatial_signed_rank

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

129.34 spearman

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

129.35 spearman_rank

129.36 spearman_to_pearson

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

129.37 wmedian

```
weighted median
```

129.38 wquantile

```
weighted quantile
```

130 lib/mathematics/statistics

130.1 qstd

130.2 quantile_extrap

131 lib/mathematics/statistics/random-number-generation

131.1 laplacernd

```
random number of laplace distribution
```

131.2 randc

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

131.3 skewness2param

131.4 skewpdf_central_moments

131.5 skewrnd

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

131.6 skewrnd2

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

132 lib/mathematics/statistics

132.1 range

```
mid range
```

132.2 resample_with_replacement

133 lib/mathematics/statistics/resampling-statistics/@Jackknif

133.1 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
```

133.2 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
```

133.3 matrix1_STATIC

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

133.4 matrix2

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

134 lib/mathematics/statistics/resampling-statistics

134.1 block_jackknife

134.2 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
```

134.3 moving_block_jackknife

```
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
```

134.4 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
```

134.5 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
```

135 lib/mathematics/statistics

135.1 scale_quantile_sd

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

135.2 sd_sample_quantiles

135.3 skewpdf

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

135.4 trimmed_mean

```
trimmed mean
```

135.5 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
```

135.6 ttest_man

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

135.7 ttest_paired

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

135.8 wgeomean

```
weighted geometric mean
function mu = wgeomean(w,x)
```

135.9 wgeovar

```
variance of the weighted geometric mean
```

135.10 wharmean

```
weighted harmonic mean
```

135.11 wharstd

135.12 wharvar

136 lib/mathematics

```
mathematical functions of various kind
```

136.1 ternary_diagram

137 lib/mathematics/test/master

137.1 dat_test_lanczos_3d_k_20_n_40

137.2 poisson2d_blk

137.3 qr_implicit_givens_2

137.4 spectral_derivative_2d

137.5 test_2d_eigensolver_hydrogen

137.6 test_2d_refine

137.7 test_3d_eigensolver_hydrogen

137.8 test_FEM

137.9 test_Mesh_3d

137.10 test_arnoldi

137.11 test_arpackc

137.12 test_assemble

137.13 test_assembly_performance

137.14 test_bc_one_sided

137.15 test_compare_solvers

137.16 test_complete

137.17 test_convergence

137.18 test_convergence_b

137.19 test_df_2d

137.20 test_eig_algs

137.21 test_eig_inverse

137.22 test_eigs_lanczos

137.23 test_eigs_lanczos_1

137.24 test_eigs_lanczos_2

137.25 test_eigs_lanczos_performance

137.26 test_fdm

137.27 test_fdm_d_vargrid

137.28 test_fdm_spectral

137.29 test_fem

137.30 test_fem_1d

137.31 test_fem_1d_higher_order

137.32 test_fem_2d_adaptive

137.33 test_fem_2d_higher_order

137.34 test_fem_3d_higher_order

137.35 test_fem_3d_refine

137.36 test_fem_b

137.37 test_fem_derivative

137.38 test_fem_quadrature

137.39 test_final

137.40 test_fix_substitution

137.41 test_forward

137.42 test_get_sparse_arrays

137.43 test_harmonic_oscillator

137.44 test_high_order_fdm_periodic_bc

137.45 test_hydrogen_wf

137.46 test_ichol

137.47 test_interpolation

137.48 test_inverse_problem

137.49 test_it_vs_exact

137.50 test_jama

137.51 test_jd

137.52 test_jdqz

137.53 test_lanczos_2

137.54 test_lanczos_biorthogonal

137.55 test_laplacian

137.56 test_laplacian_non_uniform

137.57 test_laplacian_simple

137.58 test_mesh_2d_uniform

137.59 test_mesh_2d_uniform_2

137.60 test_mesh_circle

137.61 test_mesh_generation

137.62 test_mesh_interpolate

137.63 test_mg

137.64 test_minres_recycle

137.65 test_multigrid

137.66 test_nc

137.67 test_nonuniform_symmetric

137.68 test_pde

137.69 test_permutation

137.70 test_poison_fem

137.71 test_polar

137.72 test_potential

137.73 test_powers

137.74 test_precondition

137.75 test_project_rectangle

137.76 test_qr

137.77 test_quantum_well

137.78 test_radial_adaptive

137.79 test_radial_confinement

137.80 test_radial_fixes

137.81 test_refine_2d

137.82 test_refine_2d_b

137.83 test_refine_3d

137.84 test_refine_structural

137.85 test_regularisation

137.86 test_round_off

137.87 test_schrdinger_potentials

137.88 test_uniform_mesh

137.89 test_vargrid

138 lib/mathematics/test

138.1 test_gaussfit3

138.2 test_mtimes3x3

139 lib/mathematics/wavelet

139.1 contiuous_wavelet_transform

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

139.2 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
```

139.3 example_wavelets

139.4 phasewrap

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

139.5 test_cwt_man

139.6 test_phasewrap

139.7 test_wavelet

139.8 test_wavelet2

139.9 test_wavelet_analysis

139.10 test_wavelet_reconstruct

139.11 test_wtc

139.12 wavelet

```
wavelet windows
```

139.13 wavelet_reconstruct

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

139.14 wavelet_transform

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

140 lib/mathematics

```
mathematical functions of various kind
```

140.1 wrapphase

141 lib/mesh/@StructuredMesh

141.1 StructuredMesh

```
structured mesh processing
compatible with Delft3D
also provides set-up of discretisation matrices
```

141.2 apply_boundary_condition

```
apply boundary condition and the four sides of the domain
TODO: allow for interior boudaries
```

141.3 bc_from_shp

```
read boundary condition from shape file
```

141.4 bc_index

```
TODO this is deprecated
generate indices for boundary edges
```

141.5 bc_isinvalid

```
check boundary conditions for stacked domains
```

141.6 block

```
stack multiple meshes to complex domain
```

141.7 boundary_chain

```
return chain of boundary points
```

141.8 boundary_direction

```
return direction of boundary segment
```

141.9 boundary_indices

```
indices of boundary segments
id : index of boundary point
jd : index of
```

141.10 cat

141.11 centreline

```
domain (channel) centreline along chosen dimension
```

141.12 child

```
hierarchical mesh generation (for bifurcations)
```

141.13 copy

141.14 corner_indices

```
indices of domain corners
```

141.15 cut_from_domain

```
cut subdomain
```

141.16 export_delft3d_bnd

```
export the boundary in delft3d compatible format
```

141.17 export_delft3d_dep

```
export bathymetry data in Delft3D dep-format
```

141.18 export_delft3d_grd

```
export mesh in deltares delft3D grd file format
```

141.19 export_delft3d_ini

```
export delft3D compatible initial condition file
```

141.20 export_shp

```
export mesh elements as shape file
```

141.21 extend_straight_reach

141.22 extract_elements

```
element indices from grid
```

141.23 flip_dimension

```
flip left and right or top and down
```

141.24 from_1d_mesh

```
convert a 1D mesh to 2D mesh consisting of quadrilaterals
```

141.25 generate_bifurcation

```
creates a mesh for bifurcation with bluff, which is required for
    delft3d grids
TODO do not fix indices
TODO determine p individually
bank : bankline shapefile
nn : number of points across branches
ds: spacing along s
p : fraction of right side branch
level : generate hierarchical mesh,
        grid points in each branch will be 2^n+1,
        and sub meshes until level 1 will be generated

for lower levels the connecting volumes remain narrow,
as the two volumes left and right of the division line are not
    scaled
-> post smoothing required

nn: n=6; for idx=1:5; n(end+1) = 2*(n(end)-3)+3, end
ns: n=18; for idx=1:5; n(end+1) = 2*(n(end)-2)+2, end (should be
    improved to 2*(n-1)+1
```

141.26 generate_disk

```
generate semicircular domain
```

141.27 generate_from_centreline

```
generate a mesh from a given centreline
TODO : avoid crossing of inner bed points in sharp bends
```

141.28 generate_rectangle

```
discretize a rectangular domain
```

141.29 generate_structured_grid

generate a structured mesh consisting of several sub-meshes

141.30 grid_block

mesh a subdomain

141.31 improve

improve (smooth) the mesh

141.32 interp_elem2point

interpolate values sampled at element centres to element corners
TODO allow also interpolation to u and v points

141.33 mesh_polygon

mesh a 1D channel, where boundaries are given as polygon
TODO, this should better use voronoi-tesselation (see centreline class)

141.34 orthogonality

orthogonality of elements

141.35 orthogonalize

orthogonalize mesh
set x of point coordinates to 1/2

141.36 plot

```
plot the mesh
```

141.37 plot_boundary

```
plot the mesh boundary
```

141.38 plot_coupling

```
plot connected vertices, see vertex_connection_matrix.m
```

141.39 plot_orthogonality

```
plot mesh with edges colored by orthogonality condition
```

141.40 quiver

```
quiver plot of velocity
```

141.41 read_delft3d_dep

```
depth in dat file is defined at volume centres (water leve point)
first row, first column and last column are buffer
but nast colum is not (only when outflow?)
```

141.42 read_delft3d_grd

```
read mesh in delft3D grd format
```

141.43 smooth_cubic

```
cubically smooth the mesh coordinates
```

141.44 smooth_curvilinear

```
 smooth the mesh
relax = (10+relax)/11;
relax = min(0.5,relax);
```

141.45 smooth_laplacian

```
 smooth the mesh coordinates

 better than before, but causes dn in inner bends to be narrower
     than in outer bends
 (straightens the lines)
 better smooth p: i.e. fractional distance from left to right,
 this is complicated at the bif
 better: two neighbour smooth: smooth dn and ds with left/right, top
      bottom only
```

141.46 smooth_simple

```
 smooth the mesh coordinates
```

141.47 smooth_sn

```
 smooth the mesh coordinates
```

141.48 snap

```
 snap two meshes that connect at their domain boundaries
```

141.49 statistic

```
 compute mesh statistics
```

141.50 to_unstructured_mesh

```
convert to unstructured mesh
```

141.51 transpose_dimension

```
transpose dimensions
```

141.52 vertex_connection_matrix

```
connectivity of neighbouring vertices
TODO same for elements
```

142 lib/mesh/@UnstructuredMesh

142.1 UnstructuredMesh

```
class containing some meshing functionality
complementary to Mesh_2d, Mesh_3d, Tree_2d and Tree_3d
```

142.2 add_element

```
add an element with vertex indices, vertices already exist
```

142.3 add_vertex

```
add a vertex
```

142.4 angle

```
interior angles of each element
```

142.5 assign_1d

```
assign coordinatex (x0,y0) to containing element
TODO this can fail, if triangulation is not delaunay
```

142.6 assign_2d

```
assign coordinatex (x0,y0) to containing element
```

142.7 assign_3d

```
assign coordinatex (P0,y0) to containing element
```

142.8 bnd_1d

```
left and right end points for 1D meshes
```

142.9 boundary_1d

```
convert 1D mesh to 2D mesh
```

142.10 boundary_chain2

```
get chained indices of boundary segments,
used for setting up higher order polynomials along the boundary
```

142.11 boundary_length_and_direction

```
edge length and direction of boundary segments
TODO, this should be just edge length and direction
```

142.12 cat

```
concatenate two meshes
```

142.13 chain_1d

```
chain 1D elements (segments)
```

142.14 check_dublicate_elements

```
check if elements are duplicate elements
TODO, this does not check if elements cover each other, for example
hierarchical meshes or ABC+BCD and ABD+ACD
TODO check overlap by computation of area
```

142.15 check_edge_intersection

142.16 clip

```
clip mesh to polygonal domain
TODO only works for triangles
```

142.17 compute_elem2elem

```
set up element2element neighbourhood relation
```

142.18 connect_1d_2d

```
auto merge 1d and 2d mesh
this silently requires that 1d segments consist at least of 3
    elements
TODO only implemented for triangles
```

142.19 convert_2d_to_1d

142.20 copy

```
copy constructor
```

142.21 cross_section

```
get cross-sections for 1D elements
```

142.22 delete_element

```
delete an element
```

142.23 derivative_matrix_1d

```
first order first derivative discretisation matrix on the 1d mesh
```

142.24 derivative_matrix_2d

```
first order first derivative discretisation matrix on the mesh
```

142.25 derivative_matrix_2d_2

```
second order derivative matrix on a triangulation
```

142.26 derivative_matrix_3d

```
first order first derivative discretisation matrix on the mesh
```

142.27 distance

```
distance along edges from a point set to all other points

open      : id of start point(s)
countflag : if set use number of hops as distance not the euclidean
     distance
```

142.28 dual_mesh

```
dual mesh formed by the centre of cicumference
the dual mesh consists not only of triangles
TODO rename in generate dual mesh
```

142.29 edge_length

```
euclidean edge length
```

142.30 edge_midpoint

```
edge mid-points
```

142.31 edges_from_elements

```
edges and boundaries from elements
```

142.32 eigs

```
eigenvalues of the lapalcian on the mesh
```

142.33 elem2edge_

```
pointer of element to edge
```

142.34 elem2elem_matrix

```
matrix with neighbourhood relations for each element
```

142.35 element_area

```
area of elements
1d elements have zero area and are not processed
```

142.36 element_centroid

```
centroids of lements
```

142.37 element_midpoint

```
barymetric centre of elements
```

142.38 elements_from_edges

```
2D elements from edges
```

142.39 eval2pval

```
element (centroid) value to vertex value
TODO, use dual mesh or triangulation
```

142.40 export_delft3d_net

```
export into DFLOWFM delft3d net.nc file
```

142.41 export_msh

```
export mesh in GMSH msh format
```

142.42 export_pos

```
export triangles and vertex values to gmsh pos-file format (x,y,z,
    val)
intended for re-meshing with values representing local mesh size
```

142.43 export_shp

```
export edges to GIS shapefile
each element as separate polygon with one z-value
```

142.44 facing_element

```
get triangle ndx that is opposit, e.g. "facing" the vertex vdx of
    triangle tdx
```

142.45 filter_neighbour

```
apply a function on the values on connected vertices
```

142.46 find_encroached_edges

```
find encroached edges in a triangulation,
i.e. edges for which on of the two facing point false into their
    enclosing
circle
```

142.47 flip

```
flip edges between two triangles
	flip
	for each side
		if (connection between opposit points shorter than
		    between edges, swap edge)
		this-> flip
		that-> flip
	end
```

142.48 flip_global

```
recursively flip edges, i.e ABC+BCD -> ABD+ADC,
when new edge (diagonal) is shorter
TODO this is buggy, it cannot be always swapped, only if abcd is
    convex!
```

142.49 flip_quality

```
flip edges, when mesh quality constraint improves
```

142.50 gaussmat_2d

```
matrix for gauss integration on a triangulation
```

142.51 generate_chews_first

```
triangulate domain with chew's first algorithm
```

142.52 generate_from_centreline_1d

```
generate a mesh from centreline
```

142.53 generate_from_centreline_2d

```
generate mesh from centreline
TODO allow number of segments to change

sets up a simple quadrilateral mesh in S-N coordinates
centreline (must be sorted in streamwise direction)
input variables:
cS : S (streamwise) coordinates of centreline
cL : N (spanwise) coordinate of left bank
cR : N (spanwise) coordinate of right bank
input variables controlling ouptut resolution:
S  : S coordinate of slices in S-direction (diff(S) is element
    width)
      must be sorted in s-direction
n  : n number of points per cross section
      (n-1) is number of elements per cross section
output variables:
mesh.{X,Y,S,N} : point coordinates
mesh.T        : point indices of elements (corners of the
    quadrilaterals)
-> make it orthogonal to banks by using a spline along n
```

142.54 generate_frontal

142.55 generate_ghost_elements

```
generate ghost elements, i.e. elements at the domain boundary,
    these
elements can overlap

when the project flag set, ghost points are porjected to the
    boundary,
the project flag is set for dual mesh generation
the project flag is unset for application of the boundary condition
```

142.56 generate_gmsh

```
generate a mesh from a polygon using gmsh

inshp     : file name of shape file of preloaded shape file
    containing a polygon
obase     : base of output file name
resolution : struct containing default mesh resolution settings
resfile_C : file names of shape files, defining local resolution in
     polygonal regions
opt       : options, see below

this is a Static function
```

142.57 generate_hierarchical

```
generate a hierachical mesh by recursively splitting elements
containing boundary points
```

142.58 generate_triangle

```
generate a mesh from a polygon using the programme "Triangle"
```

142.59 generate_uniform_1d

```
generate a uniformly spaced 1D mesh
```

142.60 generate_uniform_quadrilateral

```
generate a uniform 2D mesh
```

142.61 generate_uniform_tetra

```
uniformly tesselate a rhombic domain in 3D into tetrahedra
```

142.62 generate_uniform_triangulation

```
uniformly tesselate a rectangular (2d) domain into triangles
```

142.63 get_facing_and_shared_vertices

```
for a pairwise list (array) of triangles, determine there common
    and facing edges
```

142.64 grid2tri

```
topologically split a uniform mesh on a rectangular domain into
    triangles
```

142.65 import_delft3d_net

```
import mesh from Delft3d file ( {filanme}_net.nc )
```

142.66 import_msh

```
import mesh from {filename}.msh files as generated by GSMH
```

142.67 import_triangle

```
import a mesh generated with triangle (ele and node)
```

142.68 improve_iterative_relocate_insert

```
iteratively improve the mesh by inserting vertices and smoothing
              fprintf('Iteration %d, %d elements, %d vertices, %d
                  obtuse elements (%g%%)\n', iter, obj.nelem, obj.np
                  , nobtuse, nobtuse./obj.nelem);
```

142.69 improve_iterative_relocate_uniform

```
improve mesh by smoothing following by uniform refinement
              fprintf('Iteration %d, %d elements, %d vertices, %d
                  obtuse elements (%g%%)\n', iter, obj.nelem, obj.np
                  , nobtuse, nobtuse./obj.nelem);
```

142.70 improve_relocate_global1

```
iteratively improve angles to remove obtuse triangles
```

142.71 improve_relocate_global2

```
improve mesh globally
```

142.72 improve_relocate_global_3

```
improve mesh quality globally
```

142.73 improve_relocate_local

```
iteratively improve angles to remove obtuse triangles
```

142.74 improve_relocate_local_old

```
iteratively improve angles to remove obtuse triangles
```

142.75 improve_topology

```
improve mesh topology
```

142.76 insert_mid_points

```
insert mid points into the mesh
the new mesh is of much lower quality, but if all edges are flipped
    ,
this leads to the sqrt(2) refinement
```

142.77 insert_steiner_points

```
refine mesh by inserting steiner points (centre of circumference)
for elements specified by tdx
```

142.78 integrate_1d

```
integrate a quantity val across the mesh
```

142.79 integrate_discharge

```
integrate discharge
```

142.80 interp_1d

```
interpolate on a 1D mesh
```

142.81 interp_2d

```
interpolate on a 2D mesh
```

142.82 interp_fourier

```
interpolate values on the mesh using fourier methods
```

142.83 interp_tikhonov_1d

```
interpolation with Tikhonov regularisation
```

142.84 interp_tikhonov_2d

```
interpolation wiht Tikhonov regularisation in 2D
```

142.85 interp_tikhonov_3d

142.86 interpolate_from_boundary

```
interpolate interior values from the boundary
```

142.87 interpolate_point

```
interpolate from samples to mesh points by IDW method
```

142.88 interpolation_error_1d

```
estimate interpolation error in 1D
```

142.89 interpolation_error_2d

```
interpolate error in 2D
```

142.90 interpolation_error_3d

```
estimate interpolation error in 3D
```

142.91 interpolation_matrix_1d

```
linear interpolation matrix from mesh points to arbitrary
    coordinates P0
```

142.92 interpolation_matrix_2d

142.93 interpolation_matrix_3d

```
interpolation matrix for interpolation in 3D
```

142.94 isacute

```
determine acute triangles
```

142.95 isobtuse

```
determine obtuse triangles
```

142.96 iterate_smooth2

```
iteratively improve the mesh by smoothing
```

142.97 limit_by_distance

```
max edge length
minimum distance
TODO, this will always be zero
```

142.98 make_elements_ccw

```
make all 2D elements clock wise (such that their area is positive)
```

142.99 merge_duplicate_points

```
merge duplicate points
```

142.100 merge_facing_blunt_triangles

```
merge blunt triangles that face each other
```

142.101 mesh1

```
mesh in 1D
```

142.102 mesh_1d

```
extract the 1d mesh
```

142.103 mesh_2d

```
extract the 1d mesh
```

142.104 mesh_junctions

```
mesh junctions of a channel network
              hold on
```

142.105 nearest_boundary

```
determine nearest boundary segment for each input coordindate
```

142.106 nedge_

142.107 nonobtuse_refinement

```
nonobtuse refinement according to Korotov
not feasible for most obtuse triangles
```

142.108 objective_A

```
one objective function value per angle
```

142.109 objective_T

```
wrapper for mesh optimisation objective functions univariate in
    triangles
```

142.110 objective_angle

```
objective function for iterative angle improvement
```

142.111 optimum_angle

```
optimum angle for each vertex = 360^\deg / number of connected
    edges
```

142.112 orthogonality_quadrilaterals

```
orthogonality condition for quadrilaterals
```

142.113 path

```
path along edges
```

142.114 plot

```
plot the mesh (and a discretised function) as a surface and net
```

142.115 plot1d

```
plot 1D mesh
```

142.116 plot3

```
plot mesh and values
```

142.117 plotcs

```
plot cross section
```

142.118 project_to_boundary

```
project a point to the boundary
```

142.119 pval2eval

```
vertex to element value
```

142.120 quad2tri

```
quadrilaterals to triangles
```

142.121 raster_boundary

142.122 recover_edges

```
recover (boundary) edges
```

142.123 refine

```
refine by splitting marked triangles
```

142.124 refine_edge_halving

```
mesh refinement by longest edge bisection
```

142.125 remove_empty_triangles

```
remove degenerated triangles with zero area
```

142.126 remove_isolated_vertices

```
remove points that are not part of the mesh
(gmsh leaves sometimes spurious points in the msh file)
```

142.127 remove_points

```
remove points and associated elements
```

142.128 remove_quartered_triangles

```
point has connectivity 4 and is not on the boundary
```

142.129 remove_small_islands

```
delft3D requires islands to have at least 7 edges
this functions splits edges surrounding small islands
```

142.130 remove_triply_connected_boundary_vertices

```
remove boundary vertices that are connected only to three vertices
```

142.131 remove_trisected_triangles

```
remove trisected trianges
point has connectivity 3 and is not on the boundary
```

142.132 renumber_point_indices

```
renumber vertex indices
```

142.133 resolve_8_vertices

```
improve mesh by removing one edge from vertices with 8-edges
(an interior vertex in a regular triangulation has 6 neighbours,
 and unstructured meshes with local refinement are possible with
 5 and 7 neighbours, 4,3, or 8 and more connected vertices are not
    necessary
```

142.134 restore_acuteness

```
restore acuteness
Laplacian smoothing may at some places decrease the mesh quality,
this locally restores acute elements
```

142.135 retriangulate

```
retriangulate the mesh
```

142.136 ruppert

```
refine the mesh using ruppert's algorithm
```

142.137 scale_to_boundary

```
scale hierarchical mesh to match boundary coordinates
experimental
```

142.138 scatterplot

```
scatterplot of data on mesh
```

142.139 section

142.140 segment

```
segment the mesh into parts according to laplacian eigenvalues
```

142.141 smooth2

```
Laplacian smoothing of vertex coordinates,
replace every point by the average coordinate of its neibghbours
```

142.142 smooth_1d

```
smoothes values in each reach
does not smooth the values at the connection points
```

142.143 smooth_val

```
smooth values on the mesh
TODO allow for smooting boundary only along boundary
```

142.144 smoothness

```
mesh smoothness as ratio of maximum edge length and minimum edge
    length
```

142.145 split3

```
split those triangles that contain a boundary point in three pieces
    ,
for hierrachical mesh generation
```

142.146 split_edge

```
split an edge
```

142.147 split_edge_perpendicular

```
split edge perpendicularly
```

142.148 split_elem_1d

```
split a 1d element
```

142.149 split_encroached_edges

```
recursively split encroached edges
```

142.150 split_obtuse

```
split obtuse elements
```

142.151 split_unsmooth_edges

```
split unsmooth edges
```

142.152 statistics

```
compute mesh statistics
```

142.153 streamwise_derivative_matrix

```
streamwise derivative matrix
```

142.154 thalweg

```
thalweg (deepest point along channel)
```

142.155 to_single

```
TODO, also with indices
```

142.156 uncross_elements

```
make sure, that 4 point elements span an area, and do not form a
    cross
a call to this function should be succeeded by make_ccw
this operator is idempotent
```

142.157 uncross_quadrilaterals

```
make sure, that 4 point elements span an area, and do not form a
    cross
a call to this function should be succeeded by make_ccw
this operator is idempotent
```

142.158 vertex_distance

```
connectivity of directly connected vertices
```

142.159 vertex_to_edge

```
connectivity matrix between vertices and adjacent edges
```

142.160 vertex_to_element

```
connectivity matrix between vertices and elements
```

142.161 vertex_to_vertex

```
connectivity matrix between vertices
```

142.162 vertices_1d

142.163 weighed_laplacian_smoothing

```
weighed Laplacian smoothing
```

142.164 xy2xys

```
for boundary points: convert XY coordinate into a 1Dparametric
    coordinate,
applied in mesh optimization, where movement of boundary points is
constrained on the boundary
```

142.165 xys2xy

```
convert parametric 1D coordinate of boundary point back to
    cartesian XYc oordinate
```

143 lib/mesh/grid/@Grid1

143.1 Grid1

```
lump spatiotemporal data into a 1-dimensional grid
```

143.2 binop

```
operate function fun on data val within the context of a grid cell
(for fitting grid cell values from sampled values)
```

143.3 build_index

```
compute the grid-cell index for samples sampled at points X1

name : name of the index field
X1 : coordinate of source points
R  : cut off radius (if not supplied ident to mesh width)
```

143.4 fit

```
lump (fit) sampled values into the corresponding grid cell
```

143.5 predict

```
interpolate from lumped data to specified location
```

144 lib/mesh/grid/@Grid2

144.1 Grid2

```
lump spatiotemporal data into a 2-dimensional grid
```

144.2 binop

```
operate function fun on data val within the context of a grid cell
(for fitting grid cell values from sampled values)
```

144.3 build_index

```
compute the grid-cell index for samples sampled at points X1
X1 : coordinate along first dimension
X2 : coordinate along second dimension
```

144.4 plot

144.5 predict

```
interpolate from lumped data to specified location
```

145 lib/mesh/grid/@Grid3

145.1 Grid3

```
lump spatiotemporal data into a 3-dimensional grid
```

145.2 build_index

```
compute the grid-cell index for samples sampled at points X1
X1 : coordinate along first dimension
X2 : coordinate along second dimension
X3 : coordinate along third dimension
```

146 lib/mesh/mesh1d

146.1 dxspace

146.2 dxspace2

146.3 dzmesh

146.4 mesh1

146.5 mesh1d

146.6 nlogstep

147 lib/mesh/optimization

147.1 improve_smooth_insert

147.2 objective0_angle1_barycentric

147.3 objective0_angle2_barycentric

147.4 objective0_angle2_barycentric9

147.5 objective0_angle_2_cartesian

147.6 objective0_angle_inf_cartesian

147.7 objective0_barycentric9

147.8 objective0_pythagoras1_barycentric9

147.9 objective0_pythagoras1_cartesian

147.10 objective0_pythagoras2_barycentric9

147.11 objective0_pythagoras2_cartesian

147.12 objective_3_angle

147.13 objective_A_bnd

147.14 objective_P_angle

147.15 objective_P_angle_scaled

147.16 objective_P_angle_scaled_area

147.17 objective_P_midpoint

147.18 objective_angle

147.19 objective_angle2_barycentric

147.20 objective_angle_p

147.21 objective_angle_scaled_area

147.22 objective_angle_scaled_circumference

147.23 objective_cosa

147.24 objective_cosa_p

147.25 objective_cosa_scaled_side_length

147.26 objective_distance_edge_centre

147.27 objective_distance_edge_centre_perpendicular

147.28 objective_distance_orthocentre_excentre

147.29 objective_incentre_excentre

147.30 objective_length_min_max

147.31 objective_length_var

147.32 objective_thales

147.33 objective_thales_difference

147.34 test_objective_cosa_p

148 lib/mesh

mesh generation, manipulation, analysis, refinement and optimization

148.1 preload_msh

149 lib/mesh/sparsemesh/@SparseMesh1

149.1 SparseMesh1

```
lump time series of sampled spatial data in one dimension (
    projected)
```

149.2 assign

```
assign (lump) data "v0" sampled at sample times/location to field "
    field"
```

149.3 assignS

```
lump sequentially sampled data "v0" and assign to field "field"
```

149.4 init

```
initialize, segment sampling locations/times into blocks the
    sampled
data is lumped to
```

149.5 interp

```
interpolate data stored in field "field" to coordinates Xi
ingnore invalid data
TODO, check if convex
```

149.6 interpS

```
interpolate data stored in field "field" to coordinates Xi,
do not ignore invalid data
```

149.7 rmse_interp

```
interpolation part of the error :
e ~ 1/2*d^2v/dx^2 * dx^2 + higher order terms
  ~ 1/2*d^2 v
the other part of the error is the sampling error (gaussian noise)

the mesh is optimal, when e_nois ~ e_interp
```

150 lib/mesh/sparsemesh/@SparseMesh2

150.1 SparseMesh2

```
lump time series of sampled spatial data (track recordings) along
    two dimensions,
e.g 1 projected spatial dimension and one for time time
TODO : better blocks (all neighbours within mahalanobis distance)
TODO : do not use simple mean, but allow for least squares
    regression
TODO : precompute the least squares weights for accummarray
```

150.2 assign

```
assign (lump) data "v0" sampled at sample times/location to field "
    field"
```

150.3 assignS

```
lump sequentially sampled data "v0" and assign to field "field"
```

150.4 init

```
initialize, segment sampling locations/times into blocks the
    sampled
data is lumped to
```

150.5 interp

```
interpolate data stored in field "field" to coordinates Xi
ingnore data outside of the domain (convex interpolation)
```

150.6 interpS

```
interpolate data stored in field "field" to coordinates Xi,
extrapolate beyond domain
```

150.7 rmse_interp

```
interpolation part of the error :
e ~ 1/2*d^2v/dx^2 * dx^2 + higher order terms
  ~ 1/2*d^2 v
the other part of the error is the sampling error (gaussian noise)

the mesh is optimal, when e_nois ~ e_interp

TODO this is e ~ f', not f''
```

151 lib/mesh/sparsemesh

```
lumping and interpolation of spatio-temporal data into a "mesh" that
     is spaced
optimally for the local density of sample points

allows for processing of large data sets with lower memory
    consumption and run time

intended for ADCP data processing

Overcomes the limitation of gridding, where some grid cells can have
     an insufficient
number of samples
```

151.1 SparseMesh

```
SparseMesh superclass
```

152 lib/mesh/test

152.1 test_MMesh_segment

152.2 test_derivative_matrices_curvilinear

153 lib/mesh

```
mesh generation, manipulation, analysis, refinement and optimization
```

153.1 test_nxfun

153.2 trimesh_fast

154 lib/open-channel-flow/@Backwater1D

154.1 Backwater1D

```
solve the gradually varied flow equation (backwater equation)
in one dimension

c.f. Chow, Bresse
```

154.2 backwater_approximation

```
approximation of the backwater curve by an exponential function
note: this is not necessarily a good approximation
in the case of tide, Qt can be given
```

154.3 backwater_curve_iterative

```
analytic solution of the gradually varied flow equation
c.f. Bresse, Chow
```

154.4 backwater_length

```
backwater length
```

154.5 dh_dx

```
change of depth along channel for the backwater equation
beta : momentum coefficient
this is effectively an equation in h^3
```

154.6 dh_dx_

154.7 dzs_dx

```
change of surface elevation along channel
```

154.8 gvf_x_chow

```
analytical solution to the gradually varied flow equation (
    backwater equation)
c.f. Chow, Bresse
```

154.9 invert

```
determine bed level from surface elevation
(inverse backwater equation)
this is ill conditioned, as the surface is smooth for subcritical
    flow,
even if the bed is not smoth

C : chezy
W : width
Q : discharge
S : bed slope
y0 : surface elevation at outflow
lateral inflow
```

154.10 solve

```
solve the gradually varied flow equation (backwater equation)
C : chezy
W : width
Q : discharge
S : bed slope
y0 : surface elevation at outflow
```

154.11 solve_analytic

```
 analytical solution to the gradually varied flow equation (bresse
     method)
u_.^(n-m)./(1-u_.^n)
```

154.12 solve_matrix

155 lib/open-channel-flow/bifurcations-and-weirs/@Lateral_Diversion

155.1 Jb

155.2 Lateral_Diversion_Finite_Width

155.3 dR

155.4 derive

155.5 evalk

155.6 lateral_outflow_finite_width1

155.7 load_functions

155.8 stagnation_point

```
fdx   = isnan(x);
```

155.9 streamline

155.10 streamline_radius_of_curvature

155.11 u_far

155.12 v_far

155.13 velocity

155.14 velocity_near_bed

156 lib/open-channel-flow/bifurcations-and-weirs/@Lateral_Diversion

156.1 Jb

156.2 Lateral_Diversion_Finite_Width_Gradual

156.3 coefficients

156.4 condA

156.5 dR

156.6 derive

156.7 evalk

156.8 evalk_

156.9 lateral_outflow_finite_width1

156.10 load_functions

157 lib/open-channel-flow/bifurcations-and-weirs/@Lateral_Div

157.1 coefficients_old

158 lib/open-channel-flow/bifurcations-and-weirs/@Lateral_Div

158.1 stagnation_point

```
fdx   = isnan(x);
```

158.2 streamline

158.3 streamline_radius_of_curvature

158.4 u_far

158.5 uv1

158.6 uv_side_branch

158.7 v_far

158.8 velocity

158.9 velocity_linear

158.10 velocity_near_bed

158.11 xp

159 lib/open-channel-flow/bifurcations-and-weirs/@Lateral_Diversion

159.1 Lateral_Diversion_Wide_Channel

159.2 derive_lateral_outflow

```
derive potential flow solution to lateral outlfow from an
    infinitely
wide main channel
```

159.3 derive_lateral_outflow_finite_width

```
derive coefficients for lateral outflow in the case of potential
    flow
```

159.4 lateral_outflow

```
potential flow solution to the case of lateral outflow from an
    infinitely
wide channel
```

159.5 lateral_outflow_finite_width

```
analytical potential flow solution to lateral outflow from an
    infinitely
wide channel
```

160 lib/open-channel-flow/bifurcations-and-weirs/@Lateral_Div

160.1 Lateral_Diversion_Wide_Channel_Map

```
wrapper to store precomputed streamlines of potential flows
```

160.2 streamline

161 lib/open-channel-flow/bifurcations-and-weirs/@Side_Weir

161.1 Side_Weir

```
side weir, analytical solution to (critical) lateral outflow
```

161.2 dzs_dx

```
side weir, along channel surface gradient
```

161.3 surface_elevation

```
along-channel surface elevation for (critical) lateral outflow over
     a side-weir
```

162 lib/open-channel-flow/bifurcations-and-weirs

162.1 Lateral_Diversion_Finite_Width_Map

163 lib/open-channel-flow

```
functions for open channel flow, sub modules:

@Backwater1D
       gradually varied flow in 1D (backwater)
@Potential_Flow
       depth averaged potential flow, numerical solution
@Potential_Flow_Analytic
       depth averaged potential flow, analytical solution
rating-curve
       empirical rating curves
@Side_Weir
       analytical solution to lateral outflow over a side weir
@SWE
       dynamical solution of the shallow water equation (saint-
           venant-equation)
       in 1D
@SWE_2d
       dynamical solution of the shallow water equation (saint-
           venant-equation)
       in 2D
velocity-profile
       vertical and transverse velocity profiles of the streamwise
           velocity
```

163.1 hfilter

164 lib/open-channel-flow/kinematik-and-diffusion-wave

164.1 diffusion_wave

```
propagation of a diffusion wave (flood wave), c.f. ponce
advection
diffusion
where is the bed slope?
friction slope
eddy slope
chow 1988
d(A+A0)/dt + dQ/dx = q
dQ/dt + d/dx betaQ^2/A + gA(dh/dx + Sf + Se) - beta q_i v_i + Wf B
    = 0
A0 ignored
inflow and wind shear ignored
```

164.2 flood_wave_diffusion_coefficient

164.3 linear_wave

```
linear wave routing (linearised kinematic wave)
```

165 lib/open-channel-flow/meander-bend/@Equilibrium_Bend

165.1 Equilibrium_Bend

```
Transverse profile of the bed level and bed material grain size in
an equilibrium (infintely long) meander bend
```

165.2 bed_profile

```
predict transverse bed profile of an equilibrium meander bend
```

165.3 bed_profile_uniform

```
transverse profile of the bed level of an equilibrium meander bend
with uniform grain size
```

165.4 calibrate

```
calibrate bend geometry to given profile
```

165.5 dD_dr

165.6 dh_dr

```
across channel derivative of flow depth for a meandering river
```

165.7 dh_dr_uniform

```
transverse gradient of the bed level of an equilibrium meander bend
for the case of uniform bed material
```

165.8 grain_size_profile

```
transverse (across channel) profile of the bed material grain size
in a river meander
```

166 lib/open-channel-flow/meander-bend

166.1 Kinoshita

```
% Public properties
% Public get properties
% Private properties
% Constructor
% Setters and getters
% generic methods
```

166.2 bend_transverse_velocity

```
transverse velocity profile in a meander bend
```

166.3 bend_velocity_near_bed

```
near-bed-velocity in a meander bend
```

166.4 kinoshita_

166.5 random_meander

```
generate a pseudo random meander
```

166.6 test_rozovskii

167 lib/open-channel-flow/potential-flow/@Potential_Flow

167.1 Potential_Flow

```
numerical solution of the potential flow on a curvilinear grid
(not necessarilly curvilinear)
```

167.2 apply_boundary_potential_old

167.3 assemble_discretization_matrix_rectilinear

```
assemble the discretisation matrix
```

167.4 assemble_potential_matrix

```
assemble the discretisation matrix for potential flow
```

167.5 bc_dirichlet

```
apply Dirichlet boundary conditions
```

167.6 boundary_condition_side_outflow

```
apply boundary conditions for side outflow
p*phi + (1-p)*d/db phi = rhs
y : along channel coordinate
```

167.7 boundary_condition_side_outflow_1

```
apply boundary conditions
p*phi + (1-p)*d/db phi = rhs
```

167.8 contour

```
contour plot of the potential flow solution
```

167.9 cut_boundary

```
cut the boundary from the domain
wa : width of inlet to side channel
wb : width of side channel
```

167.10 cut_rectangle

```
cut a rectangle from the domain
TODO, this requires also an adaptation of the derivative matrices
     -> step over to semi-unstructured mesh
```

167.11 infer_bed_level

```
note: this is pretty much a broken function for the inference of
    stationary
      morphology

Missing:
- rolling down of transverse slope to balance secondary flow in
    bends
- quasi time steippong

at stationary state:
- changes of discharge along the streamlines of discharge are
    balanced
  by a change in depth, to keep the velocity and sediment transport
       constant along the streamline

dz_b/dt = dqs/dx + dqs/dn = 0               (i)
TODO this only true for infinite bends, as sediment can also move
    to the side
dqs/ds = d/s(q/h) = 1/h dq/ds - q/h^2 dh/ds = 0
TODO this is only true in an ifinite bend (ikeda)
dqs/dn = 0
streamlines along discharge or velocity -> does not matter eq (i)
    is direction independent
```

167.12 infer_bed_level2

```
infer the bed level
```

167.13 infer_bed_level3

167.14 infer_bed_level_loop

```
the bed level does not completely converge but starts to oscillate,
this is presumably due to the non-compact kernel implementation of
    the laplacian oberator
```

167.15 objective_bed_level

```
objective function for determining the bed level
```

167.16 old

167.17 plot

```
surface plot
```

167.18 quiver

167.19 sediment_transport

```
compute the sediment transport
```

167.20 solve_potential

```
solve for the flow potential
```

167.21 streamline

```
compute a streamline
```

167.22 surface_elevation

```
compute surface elevation according to Bernoulli's law
```

167.23 test

167.24 velocity_near_bed

```
determine the velocity near the bed
```

167.25 vertical_velocity

```
determine the vertical velocity from continuity
```

168 lib/open-channel-flow/potential-flow/@Potential_Flow_Ana

168.1 Potential_Flow_Analytic

```
analytical solutions to various depth-averaged potential flow
    problems
```

168.2 streamline

```
numerically follow path along streamline by integrating the
    velocity
```

169 lib/open-channel-flow/rating-curve

169.1 ChezyRatingCurve

```
rating curve, Chezy formalism
```

169.2 DynamicKeuleganRC

```
Dynamic Rating Curve, Keulegan roughness formulation
(dynamic = correction for hysteresis loop)
```

169.3 DynamicManningRC

```
Dynamic Rating Curve, Manning roughness formulation
(dynamic = correction for hysteresis loop)
```

169.4 DynamicPowerRC

```
Dynamic Power Law Rating curve
(dynamic = correction for hysteresis loop)
```

169.5 KeuleganRatingCurve

169.6 ManningRatingCurve

169.7 PolyRatingCurve

169.8 PowerRatingCurve

```
stationary rating curve, power law
```

169.9 PowerRatingCurveOffset

```
stationary rating curve, stage-discharge follows power law
```

169.10 RatingCurve

```
Fri Feb 13 10:02:52 CET 2015
rating curve superclass
```

169.11 csarea

```
predict cross sectional area from transverse bed level profile
and surface elevation
```

169.12 csdischarge

```
compute discharge
```

169.13 csperimeter

```
compute wetted perimeter
```

169.14 csradius

```
compute hydraulic radius of the cross section
```

169.15 cswidth

```
determine cross section width
```

169.16 test_PowerRatingCurve

169.17 wfunc

```
determine channel width
```

170 lib/open-channel-flow/shallow-water/@SWE

170.1 SWE

```
Class to solve the (cross sectionally averaged) shallow water
    equation
(st venant equation)
```

170.2 bc_incoming_non_reflecting

```
set non-reflecting boundary condition for the 1D SWE
```

170.3 bc_inflow

inflow boundary condition

170.4 bc_inflow_low_pass

set low frequency Dirichlet, high frequency pass boundary condition

170.5 bc_inflow_non_reflecting

set non-reflecting boundary condition

170.6 bc_level

set surface level as Dirichlet boundary condition

170.7 bc_level_sommerfeld

set surface level as boundary condition by sommerfeld method

170.8 bc_nonreflecting

set non-reflecting boundary condition
extrapolate 0-order

170.9 bc_reflecting

set reflecting boundary condition
extrapolate 0-order and invert v

170.10 dot

```
time derivative
(only for matlab internal ode-solver)
TODO this is not swe specific
continuity
dA/dt + dQ/dx = I

momentum
dQ/dt + d/dx( Qu + 1/2 gh^2) = gA(S_f - S_b)
S_b = dz_b/dx
S_f = tau_x/rho_w = C_f u|u|
```

170.11 dt_cfl

```
determine time step required by cfl
```

170.12 energy

```
determine total energy as sump of potential and kinetic energy
this is preserved for fricitionless flows
```

170.13 flux

```
st venant's shallow water equation fluw
```

170.14 flux_lin

```
linearised st-venant equation
```

170.15 fluxmateig

```
eigenvalues und vectors of the swe
```

170.16 jacobian

```
Jacobian of the SWE

dq/dt + J dq/dx = sourceterm
note: d/dx(A*q) = J dq/dx
```

170.17 lindot

```
linearised SWE
width variation not included, goes into rhs force term

[      0,  1] [A]    = [    Q]
[ -u^2+gH, 2u] [Q]_dx [Q^2/A+1/2gA^2/w]_dx - 1/2gA^2/w^2 dw/dx
                                           force term
```

170.18 roe_average

```
roe average for the SWE
```

170.19 solve_analytic

```
linearised analytic solution of the swe
```

170.20 solve_stationary

```
stationary solution to the SWE
```

170.21 source_bed_level

```
source term of the SWE caused by a change of the bed level

Note: this term causes splitting and averaging methods to fail to
     give accurate predictions of the smooth surface at steps of
        the bed
```

170.22 source_friction

```
friction source term of the SWE
```

170.23 source_width

```
source term (reaction term) for channels with variable width
```

170.24 swe_geometry

```
predefined functions to set up channel geometry
```

170.25 swe_ic

```
predefined functions of channel geometries
```

171 lib/open-channel-flow/shallow-water/@SWE_2d

171.1 SWE_2d

```
Dynamic solution of the shallow water equation (depth average, 2D)
```

171.2 apply_boundary_condition_stationary

```
apply boundary condition for stationary flow
```

171.3 assemble_stationary

```
TODO, g should be replaced by gx,gy,gz, see chaudhri
assemble discretisation matrix for stationary flow
```

171.4 solve_stationary

```
solve SWE for statinary flow (dU/dt = dQ/dt = 0)
```

172 lib/open-channel-flow/shallow-water

172.1 sw_reflection

```
reflection coefficients of shallow water waves at a sudden change
    of the
cross section (sudden change of admittance)
c.f. lighthill, ippen-harleman
```

172.2 sw_reflection_stepwise

```
time passes and phase shifts
transmission and reflection coefficient depend on direction !
iterative (recursive) reflection and transmission
```

173 lib/open-channel-flow/test/test_Backwater1D

173.1 test_bw1d_solve_matrix

174 lib/open-channel-flow/test

174.1 test_inverse_backwater_curve

174.2 test_normal_flow

174.3 test_nse_nz

175 lib/open-channel-flow/uniform-stationary-flow

175.1 chezy2drag

175.2 chezy2f

175.3 chezy2manning

175.4 chezy2z0

175.5 critical_flow_depth

```
critical flow depth in uniform stationary flow
```

175.6 drag2chezy

```
convert drag coefficient to chezy coefficient
g dz_s/dx + cd w u^2/h = 0 (swe formalism)
    - S  +  1/C^2 U^2/H = 0 (chezy formalism)
```

175.7 f2chezy

175.8 ks2z0

175.9 manning2chezy

175.10 manning2drag

175.11 manning2z0

175.12 normal_flow_depth

```
normal flow depth for uniform stationary flow
function H = normal_flow_depth(Q,W,C,S)
```

175.13 normal_flow_depth_

```
normal flow depth in uniform stationary flow
```

175.14 normal_flow_discharge

```
normal flow discharge for uniform stationary flow
```

175.15 normal_flow_slope

```
energy slope (surface slope) for uniform stationary flow
normal flow slope in uniform stationary flow
```

175.16 normal_flow_velocity

```
normal flow velocity in uniform stationary flow
```

175.17 normal_shear_velocity

175.18 shear_velocity

175.19 z02chezy

175.20 z02ks

175.21 z0tochezy

176 lib/open-channel-flow/velocity-profile/@Log_profile

176.1 Log_profile

```
logarithmic profile of the streamwise velocity
```

176.2 df_dh

```
sensitivity of profile with respect to depth
```

176.3 df_dh_

```
sensitivity of profile with respect to depth
```

176.4 df_dln_z0

```
sensitivity of velocity profile with respect to roughness length
```

176.5 df_dln_z0_

```
sensitivity of profile with respect to roughness length
```

176.6 profile

```
vertical profile of the streamwise velocity
```

176.7 profile_

```
scale of velocity at instrument depth to depth average velocity
roughness length and associated standard error can change in time,
i.e. may be passed as vectors

zs    : [1xn] water surface level
zb    : [1x1] bottom level
za    : [1xn] or [1x1]
        level of velocity measurement,
        i.e. level of HADCP beam bin centre, coincides with
            instrument level,
        if the HADCP is horizontally aligned
        only needs to be passed as vector if instrument is
            redeployed or
        becomes misaligned
ln_z0 : [1xn] or [1x1]
        natural logarithm of the roughness length
s     : [1xn] or [1x1]
        standard error of ln_z0
function [fz_mu fz_s fz_sp fz_bias fz_eps] = log_profile(zs,zb,za,
    ln_z0,s,sp,e)
```

176.8 profile_bias

176.9 regmtx

```
regression matrix
```

176.10 ubar

```
depth averaged velocity
```

177 lib/open-channel-flow/velocity-profile/@Log_profile_with_b

177.1 Log_profile_with_bend_correction

```
vertical velocity profile corrected for bend flow
```

177.2 df_dc

```
sensitivity of the velocity profile with respect to the bend
    correction
parameter c
```

177.3 df_dc_

177.4 du_dz

177.5 fit

```
fit the vertical velocity profile
```

177.6 profile_

```
vertical velocity profile
```

177.7 regmtx

```
regression matrix
```

177.8 u

```
streamwise velocity
```

177.9 u_

streamwise velocity

178 lib/open-channel-flow/velocity-profile/@Log_profile_with_cubic_wa

178.1 Log_profile_with_cubic_wake

log profile with cubic wake

178.2 df_dc

sensitivity of profile with respect to wave parameter

178.3 df_dc_

sensitivity of profile with respect to wake parameter

178.4 profile_

vertical velocity profile

178.5 regmtx

regression matrix

179 lib/open-channel-flow/velocity-profile/@Log_profile_with_dip

179.1 Log_profile_with_dip

Logarithmic profile with dip

179.2 fit

```
fit the vertical velocity profile
```

180 lib/open-channel-flow/velocity-profile/@Log_profile_with_li

180.1 Log_profile_with_linear_bend_correction

```
log profile with linear bend correction
```

180.2 df_dc

```
sensitivity of profile with respect to wake parameter
```

180.3 df_dc_

```
sensitivity of velocity profile with respect to wave parameter
```

180.4 du_dz

```
velocity shear along vertical
```

180.5 profile_

```
velocity profile
```

180.6 regmtx

```
regression matrix
```

181 lib/open-channel-flow/velocity-profile/@Log_profile_with_wake

181.1 Log_profile_with_wake

```
logarithmic velocity profile with wake correction
c.f. coles
```

181.2 df_dc

```
sensitivity of profile with respect to wake parameter
```

181.3 df_dc_

```
sensitivity of velocity profile with respect to wake parameter
```

181.4 du_dz

```
velocity shear
```

181.5 profile_

```
predict velocity profile
```

181.6 regmtx

```
log law with wake
u = us/k ln(z) - us/k ln(z0) + us/k (2/H^2 z - 3/H^3 z^2)
```

182 lib/open-channel-flow/velocity-profile/@VP

182.1 VP

```
velocity profile
```

182.2 process_joint

182.3 process_transverse_profile

```
process the transverse velocity profile
```

182.4 process_vertical_profile

```
predict vertical profile error distribution parameter for HADCP
    error estimate
```

182.5 profile_prediction_error

```
 input :
 U    : [nbin x nens]
        - values for each bin (or across section) and ensemble (or
            reference measurement)
        this are estimates estimates of the discharge or the cross
            sectional averaged
        velocity from the raw values
        - the profile should be limited to the effective profiling
            range,
        abobj 75-100m for a 600kHz ADCP

 dn   : distance between HADCP bins
 width : cross section width

 objput:
       sd_n : expected standard deviation for increasing profiling
            range
function [s_rel s_err s_dat rho res m2 u_pred fdx] =
    velocity_variation(U)
 hadcp_prediction_error
 TODO take scales and unscaled velocity to do combine with harmmean
     estimate
 note: previus versions:
       residual was computed with respect to the predicted local
           velocity
       mse was not upscaled to cs, as profile was expected to cover
           entire cs
       finite width of cs was not considered
```

```
parametric estimate from moments, objliers should be filtered
    beforehand
Note that the median absolute deviation is not a good estimate,
because it may excludes rare events like reverse flow of floods
thus, the only acceptible more robust estimate would be mean
    absolute deviation
```

183 lib/open-channel-flow/velocity-profile/@Vertical_profile

183.1 Vertical_profile

```
vertical profile of the streamwise velocity, superclass
```

183.2 fit

```
fit vertical velocity profile parameter

function obj = fit(obj,U,S,h,binmask)
```

183.3 u

```
predict velocity along the vertical based on profile
```

184 lib/open-channel-flow/velocity-profile

184.1 fit_displacement_profile

```
fit the log profile to the vertical profile of the streamwise
    velocity
```

184.2 lateral_division_method

```
transverse (across channel) profile of the streamwise velocity
in a straight channel
numerical solution
the eps seems incorrect, use better stationary_1d_swe

rho g h S - beta q^2 f / (8 h^2) + d/dy(eps_t dq/dy) = 0
rho g h S - beta q^2 g / (C^2 h^2) + d/dy(eps_t dq/dy) = 0
```

184.3 test_law_of_the_wall_fit

184.4 transverse_profile_parameter

184.5 transverse_velocity_profile

```
transverse profile of the streamwise velocity
c.f. shiono knight
```

184.6 transverse_velocity_profile_olesen

```
transverse profile of the streamwise velocity in a meander bend
```

184.7 transverse_velocity_profile_rozovskii

```
transversal velocity distribution in a bend
Rososkii,
as in the book central differences along the radius and euler
    forward in space
are used, note that since the advent of the computer more advanced
    schemes
could be used (see build in solvers)
cfl condition is not explicitely checked
Rosovsky assumes a constant water level, e.g. does not consider
    superelevation

I_theta = -1/r dz/d_theta   (p. 22)
d_theta = 1/R ds|_R
=> I_theta = -R/r dz/ds = -R/r I_0
It : (1.32) drop of level per unit angle, identical across section
```

184.8 transverse_velocity_profile_shiono_knight

```
transverse profile of the streamwise velocity, determined
    analytically
by the method of shiono and knight
shape of velocity profile only dependent on lambda, f, H, not slope
```

184.9 transverse_velocity_profile_tidal_channel

184.10 transverse_velocity_profile_with_slope

```
stationary 1D shallow water equation across a river section
0 = - g h S0 - tau_b/rho + d/dn (nu h du/dn)
0 = - g h S0 + g u^2/C^2 + d/dn (nu h du/dn)
includes tranvese gradient term

note that shiono/knight 1991 provide an _analytic_ solution,
which takes the form of an expontially decaying side wall effect
```

184.11 vertical_profile_of_velocity_vriend

```
vertical profile of the streamwise velocity, method of de vriend
```

184.12 vertical_velocity_profile

```
vertical profile of the streamwise velocity in non-uniform flow
```

184.13 z2s_rational

185 lib/open-channel-flow/wrapper

185.1 discharge2stage

```
wrapper function
```

185.2 stage2discharge

186 lib/physics/@Constant

186.1 Constant

```
Constant and physical standard quantities
```

186.2 celsius_to_kelvin

```
convert temperature from degree Celsius to Kelvin
function t_K = celsius_to_kelvin(t_C)
```

186.3 depth_to_pressure

```
convert depth to pressure in fresh water at standard temperature

   z = (p - p0)/(rho g)
=> p = rho g z + p0

input :
p0 : nx1 or scalar, pressure at water surface in BAR
d  : depth in metre

output :
p  : nx1, pressure at measurement depth in BAR
```

186.4 kelvin_to_celsius

```
convert temperature degree Kelvin to Celsius
```

186.5 optical_attenuation

186.6 pressure_to_depth

```
convert pressure to depth in fresh water at standard temperature

z = (p - p0)/(rho*g)
```

```
input:
p  : nx1, pressure at measurement depth in BAR
p0 : nx1 or scalar, pressure at water surface in BAR

output:
d  : depth in metre
```

186.7 saturation_vapor_pressure

186.8 sound_absorption_air

186.9 sound_absorption_water

```
sound absrobption in water
following Francois and Garrison, 1982


function alpha = sound_absorption(f,S,D,T)

input:
f : frequency (Hz)
S : salinity
D : depth (m)
T : temperature (degree C)

output:
alpha = sound attenuation in dB/m (not dB/km)

function alpha = sound_absorption(f,S,D,T,model)
```

186.10 sound_velocity_water

```
sound velocity in water
following Lubbers and Graaff (1998)
this formula does not include depth and salinity effects
```

186.11 viscosity_dynamic_water

186.12 viscosity_kinematic_water

187 lib/physics

187.1 beam_bending_deflection

187.2 beam_bending_moment

187.3 beam_bending_strain

187.4 beam_bending_stress

187.5 bolt_stress

187.6 drag_force

188 lib/physics/hydrogen-spectrum

188.1 hydrogen_spectrum_1d

188.2 hydrogen_spectrum_2012_12_02

188.3 hydrogen_spectrum_2d

188.4 hydrogen_spectrum_3d

189 lib/physics

189.1 minimum_cable_diameter

189.2 moment_of_inertia_rectangle

189.3 moment_of_inertia_ring

189.4 parabolic_reflector_gain

190 lib/physics/salinity

190.1 Salinity

190.2 Salinity78

190.3 canter_cremer_number

```
Canter Cremer Number
ratio of fresh water to sea water that flows into the estuary
Qf : fresh water discharge
T  : tidal period
Pt : tidal prism
Savenije, Salinity and tides, eq. 1.1, 2.35 and 5.67
```

190.4 density2salinity

190.5 dispersion_hws_savenije

```
Dispersion at river mouth during high water slack

v0 : tidal velocity scale
E0 : tidal excursion
h0 : depth
a  : convergence length
Nr : Richargson Number

Savenije 1993c, Savenije, Salinity and Tides, eg. 5.70
```

190.6 dispersion_tda_burgh

190.7 richardson_number

```
Estuarine Richardson Number
potential energy due to mixing the entire fresh water with sea
    water
ratio of potential energy and buoyancy
Savenije, Salinity and Tides, 2.36
drho : difference of sea water and fresh water density
rho  : fresh water density
h    : depth
v    : tidal velocity scale
N    : Cramer number
```

190.8 salinity

190.9 salinity_intrusion_length

190.10 sea_water_density

190.11 tidal_discharge

```
specific tidal discharge (discharge per unit width)
```

190.12 tidal_excursion

```
Tidal excursion length
Pt : tidal prism
h0 : depth
w0 : width
```

190.13 tidal_prism_channel

```
Tidal prism
Pt = int_lsw^hws Q_t dt ~ A E
z1 : tidal amplitude
w0 : width of estuary at mouth
b  : length of width convergence
dH_dx = rate of damping of H
c.f. Savenije 2.34, 2.64
```

190.14 tidal_prism_estuary

```
Tidal prism
Pt = int_lsw^hws Q_t dt ~ A E
z1 : tidal amplitude
w0 : width of estuary at mouth
b  : length of width convergence
dH_dx = rate of damping of H
c.f. Savenije 2.34, 2.64
```

190.15 tidal_velocity

191 lib/physics

191.1 test_sound_absorption_air

192 lib/physics/turbulence

192.1 keps2nu

193 lib/physics/wind-wave

193.1 short_wave_length

193.2 short_wave_shear_velocity

193.3 wave_height_from_wind_speed

194 lib/sediment-transport/@GrainSizeDistribution

194.1 GrainSizeDistribution

194.2 assign_channel

194.3 bimodality

194.4 export_csv

194.5 export_shp

194.6 group_channels

194.7 group_curvature

194.8 group_histograms

194.9 load_coordinates

195 lib/sediment-transport/@Hermite_profile

195.1 Hermite_profile

```
suspended sedimen profile in form of a hermite polynomial
```

195.2 fit

```
fit suspended sediment profile
```

195.3 predict

```
predict suspended sediment concentration
```

195.4 regmtx

```
regression matrix
```

195.5 transform

```
hermite profile
```

196 lib/sediment-transport/@Nodal_Point

196.1 Adot

```
ODE of the nodal point relation (time-derivative of branch cs-area)
```

196.2 Nodal_Point

```
Nodal point relation for bifurcations, according to Wang
```

196.3 Qs_in

```
sediment entering branches
```

196.4 Qs_out

```
sediment leaving branches
```

196.5 derive_jacobian

```
derive Jacobian of the nodal point relation
```

196.6 discharge

```
discharge through branches
```

196.7 geometry

```
cross section geometry of branches
```

196.8 jacobian

```
jacobian of the nodal point relation
semi-autogenerated
```

196.9 phase_diagram

```
phase diagram
```

196.10 phase_diagram_wang

```
phase diagram of Nodal point relation
```

196.11 solve

```
solve the nodal point relation for critical points
```

196.12 stability_analysis

```
staility analysis for a given configuration
```

197 lib/sediment-transport/@Parabolic_Constant_Profile

197.1 Parabolic_Constant_Profile

```
parabolic-constant profile
```

197.2 fit

```
fit the suspended sediment concentration profile
```

197.3 predict

```
predict suspended sediment concentration
```

197.4 regmtx

```
regression matrix
```

197.5 transform

```
transformation of vertical coordinate
```

198 lib/sediment-transport/@Rouse_Profile

198.1 Rouse_Profile

```
suspended sediment concentration profile
```

198.2 fit

```
fit the suspended sediment concentration profile
```

198.3 mean_concentration

198.4 predict

```
predict the suspended sediment concentration
```

198.5 regmtx

```
regression matrix
```

198.6 rouse_number

```
rouse number (suspension number) for given grain siye and shear
    velocity
```

198.7 rouse_number_to_grain_diameter

```
convert known rous number (suspension parameter) to grain size
    diameter
```

198.8 set_parameters

198.9 transform

```
transform the vertical coordinate
```

199 lib/sediment-transport

```
analysis and prediction of fluvial sediment transport and
    morphodynamics
```

199.1 Exponential_SSC_Profile

199.2 adaptation_length_bed

```
adaptatoion lenght of bed morphology
```

199.3 adaptation_length_flow

```
adaption length of the flow
```

199.4 bar_mode_crosato

```
bar mode of a river according to crosato
```

199.5 bed_layer_thickness

199.6 bed_load_einstein

```
bed load transport according to einstein jr.
```

199.7 bed_load_engelund_fredsoe

```
bed load transport according to engelund and fredsoe
```

199.8 bed_load_transport_mpm

```
bed load transport rate according to meyer-peter-mueller
```

199.9 bed_load_transport_rijn

```
bed load transport
method of van Rijn (1984)

function [Q_b q_b Phi_b] = bed_load_transport_rijn(C,d50,d90,U,d,b)

d50 [mm] (converted to m)
d90 [mm] (converted to m)

d : depth
b : width
```

199.10 bed_load_transport_wu

```
bed load transport according to Wu
```

199.11 bedform_dimension_rijn

```
bed form dimensions
cf. rijn 1984 iii
```

199.12 bedform_roughness_rijn

```
form drag according to van Rijn
```

199.13 bedform_roughness_rijn_2007

199.14 bedload_direction

```
bedload transport direction
```

199.15 bedload_layer_thickness_mclean

199.16 bifurcation_critical_aspect_ratio

```
critical aspect ratio of a bifurcation
c.f. redolfi and pittaluga
```

199.17 chezy_einstein

```
chezey coefficient according to Einstein
```

199.18 chezy_roughness_engelund_fredsoe

```
chezy rougness according to engelund and fredsoe
```

199.19 chezy_to_manning

```
convert chezy to manning
```

199.20 critical_grain_size

```
critical grain size for a given shear velocity
```

199.21 critical_shear_stress

```
critical shear Stress
```

199.22 critical_shear_stress_ratio

```
critical shields parameter
aka critical shear stress ratio
aka shields curve
```

199.23 critical_shear_stress_wu

critical shear stress, according to wu

199.24 critical_shear_velocity

critical shear velocity

199.25 derive_mpm_foramtive_discharge

199.26 dimensionless_grain_size

dimensionless grain size

199.27 dune_celerity

199.28 dynamic_shear_stress

dynamic shear stress

199.29 fractional_transport_engelund_hansen

fractional sediment transport according to engelund and hansen

199.30 grain_roughness_mpm

199.31 grain_roughness_rijn

grain roughness (skin friction) according to van Rijn

199.32 grain_roughness_wu

199.33 hiding_exposure_wu

199.34 hydraulic_radius

199.35 manning_to_chezy

```
manning to chezy conversion
```

199.36 mobility_parameter_rijn

199.37 mpm2diameter

199.38 mpm_solve_for_dm

199.39 reference_concentration_rijn

199.40 reference_concentration_smith_lean

```
reference concentration according to smith and mclean
```

199.41 reference_height_rijn

199.42 reference_to_flux_averaged_concentration_rijn

199.43 saltation_layer_thickness

199.44 sediment_transport_directed

```
directed sediment transport
```

199.45 sediment_transport_engelund_hansen_2

```
sediment transport according to engelund and hansen
```

199.46 sediment_transport_relation_fit

199.47 sediment_transport_relation_predict

199.48 sediment_transport_scale

199.49 sediment_transport_waves

```
sediment transport by waves
```

199.50 settling_velocity

```
Settling velocity
5.23d in julien-2010
settling velocity in water
settling velocity according to cheng
stokes settling velocity
d : [mm] diameter of sediment particle
ws : [m/s] settling velocity
          signed ws < 0 : falling
(Note: was R, radius in m)

valid for small particles
```

199.51 settling_velocity_to_diameter

```
invert settling velocity to diameter
```

199.52 shields_number

```
normalized shear stress, shear stress ratio
```

199.53 skin_2_total_friction_eh

```
skin friction to total friction conversion according to engelund
    and hansen
function [theta,C] = skin_2_total_friction_eh(theta_t,Ct)
```

199.54 suspended_grain_size

```
suspended grain size distribution based on bed material grain size
    distribution

assumes that probability of suspension is inverse proportional to
    grain diameter
as in Engelund-Hansen transport relation
- no hiding effects considered
- no threshold for large grains applied
- no flocking considered
note: actual distribution varies with the depth
```

```
d    : [1xnd] grain size in arbitrary units (on linear, not on log
    scale)
h_bed : [nsxnd] fractions of sediment of size d
```

199.55 suspended_grain_size_non_linear

```
suspended grain size distribution based on bed material grain size
    distribution

assumes that probability of suspension is inverse proportional to
    grain diameter
as in Engelund-Hansen transport relation
- no hiding effects considered
- no threshold for large grains applied
- no flocking considered
note: actual distribution varies with the depth

d    : [1xnd] grain size in arbitrary units (on linear, not on log
    scale)
h_bed : [nsxnd] fractions of sediment of size d
```

199.56 suspended_grain_size_rijn

```
grain size of the suspended sediment according to van rijn,
    empirical
```

199.57 suspended_transport_mclean

```
vertical profile of the suspended sediment according to McLean
u := us/kappa*log(z/z0);
I = 1/(int_a^h c dz int_a^h u dz) int_a^h c u dz
```

199.58 suspended_transport_rijn

```
suspended load transport according to van Rijn
```

199.59 suspended_transport_wu

```
suspended sediment transport according to widthu
```

199.60 suspension_parameter_rijn

200 lib/sediment-transport/test

200.1 test_adaptation_length_bed

200.2 test_critical_shear_stress

200.3 test_settling_velocity_to_diameter

201 lib/sediment-transport

```
analysis and prediction of fluvial sediment transport and
    morphodynamics
```

201.1 test_sediment_transport_relation

201.2 total_roughness_engelund_fredsoe

```
 roughness lenght according to engelund and fredsoe
```

201.3 total_roughness_rijn

```
 total roughness according to van rijn
```

201.4 total_transport_ackers_white

201.5 total_transport_bagnold

```
total sediment transport accoding to bagnold
```

201.6 total_transport_eh_distribution

```
total sediment transport according to engelund hansen
for a given graqin size distribution
```

201.7 total_transport_engelund_hansen

```
total sediment transport according to Engelund and Hansen
```

201.8 total_transport_rijn

```
total sediment transport according to van rijn
```

201.9 total_transport_wu

```
total sediment transport according to wu 2000b
```

201.10 total_transport_yang

201.11 transport_stage_mclean

```
transport stage according to McLean
```

201.12 transport_stage_rijn

```
transport stage as defined by van Rijn
```

201.13 vertical_ssc_profile_mclean

```
vertical profile of the suspended sediment according to McLean
```

202 lib/tide/@T_Tide

202.1 T_Tide

```
wrapper for TPXO generated tidal time series
```

202.2 build_index

```
build a structure whose field names contain the index
```

202.3 from_tpxo

```
read TPXO output into tidetable object
```

202.4 get_constituents

```
extract constituents of tpxo object
```

202.5 reorder

```
order constituents as specified by "name"
```

202.6 select

```
select a subsect of constituents
```

202.7 shift_time_zone

```
shift phase according to time zone
```

203 lib/tide/@Tidal_Envelope

203.1 Tidal_Envelope

```
process tidal data to extrac the tidal envelope
```

203.2 init

```
initialize with data
```

204 lib/tide/@Tide_wft

wavelet analysis of tidal data

204.1 Tide_wft

```
wavelet transform of tidal time series
```

204.2 transform

```
wavelet transform tidal time series
input:
time   : [1xn] abszissa of input vector, for example time, must be
    equally spaced
val    : [1xn] signal, input data series (e.g water level or
    velocity)
F      : [1xm] base frequencies, 1, 1, 2, ... for mean level,
    diurnal, semidirunal ...
              base periods from base frequencies T=1/F
n      : [1xm] wavelet window length in multiple of periods
fc, nc : [scalar] low frequency cutoff and window length in periods
winstr : [char] fourier windows (kaiser (recommended), hanning, box
    , etc)
dt_max : [scalar] maximum time to fill gaps in input data series (
    recommended 3/24 for tide)
output:
tide   : struct with fields
        w_coeff  : [1xn] wavelet coefficients (complex)
        amplitude : amplitude
        phase    : phase
        range    :
```

```
h_tide   :
h_low    :
h
```

205 lib/tide/@Tidetable

```
class for generating tidetable data
```

205.1 Tidetable

```
 Tide table
```

205.2 analyze

```
 extract tidal envelope from time series
```

205.3 export_csv

```
 export tide table to csv file
```

205.4 generate

```
 run TPXO to generate time series
```

205.5 generate_tpxo_input

```
 generate tpxo input table
 Note: superseeded by perl script
```

205.6 import_tpxo

```
 import TPXO data into tidetable object
```

205.7 plot_neap_spring

```
plot average neap and spring tide
```

206 lib/tide

```
analysis prediction of tides in rivers and estuaries by empirical
    and theoretical methods
```

206.1 constituents

206.2 doodson

```
frequency of tidal constituents
method of doodson
source: wikipedia
```

206.3 envelope_amplitude

```
compute envelopes of hw and low water
```

206.4 envelope_slack_water

```
slack water envelope of the tide
```

206.5 interval_extrema

```
times and evelations for high and low water
```

206.6 interval_extrema2

```
mimimum and maximum within intervals of constant length,
intended for periodic functions
```

206.7 interval_zeros

```
times of slack water determined frim velocity u
```

206.8 lunar_phase

```
lunar phase
```

206.9 rayleigh_criterion

```
raleigh criterion for resolving tidal constituents
T > 1/|f1-f2|
```

207 lib/tide/river-tide/@River_Tide

```
predict tide in a backwater affected river with a sloping/varying
    bed

Assumptions and capabilities:
       - tidal dynamics follow the 1D-Shallow-Water-Equation
         (depth and cross-sectionally averaged Navier-Stokes-
             Equation)
       - rectangular cross section
       - width can vary along the channel
       - friction coefficient (cd) constant along channel and over
           time (Chezy)
       - advective accelleration term is considered, but can be
           deactivated
       - vertical profile of streamwise velocity is constant
         (Boussinesq coefficient is unity (1))

Limitations / TODO list:
       - single channel dynamics only (no tidal networks)
       - no wind-shear stress (no storm surges)
       - no tidal flats / intertidal areas (width constant in time)
       - no flood-plain during high-river flow
       - no stratification or along-channel salinty gradient
       - negligible head loss in channel bends
       - negligible feed-back of the sediment concentration on the
           propagation of the tide
       - low Froude Number (no hydraulic jumps due to cataracts or
           tidal bores)
```

```
- At present, only two tidal components are supported
  (either D1 with D2 or D2 with D4, in addition to the mean water
      level z0),
  for mixed diurnal-semidiurnal cases with dominant semidiurnal
      component,
  the class has to be extended to support three components (D1, D2
      and D4)

- At present, the tripel overtide is not computed (D3 for diurnal,
    D6 for semindiurnal tide),
  note that this is the main overtide for the case of low river flow

- At present, the 1/h non-linearity is only included in the
    approximations of
  the backwater curve, but not it's influence on the tidal frequency
       components

Method:

This class calls numerical solvers for second order ordinary
    differential
equation boundary value problems

Tides is represented as exponential series in form of total
    discharge Q = sum Q_i = Q_0 + Q_1 + Q_2,
as discharge is conserved (balanced), the equations are simpler than
     for level z and velocity u,
and the frequency components of z are straight forward determined by
     differentiation of Q

Class and function structure:
       River_Tide :
              computes river tide, provides the ode coefficients to
                  the boundary value solver
       bvp2c, bvp2fdm :
              solve the underlying second order boundary value
                  problem
       River_Tide_Map :
              provides convenient batch runs and processing of
                  River_Tide instances

Minimum working example, c.f. example_rive_tide.m and
    example_river_tide_map.m

       input:

       Q0    : scalar, river discharge (m^3/s)
       omega : scalar, angular frequency main tidal species in (1/
```

```
    seconds)
x     : 2x1 vector, left and right end of computational
    domain of the river (m)
w(x)  : function of width along the river (m)
cd(x) : function of drag coefficient along the river (1)
zb(x) : function of bed level along the river (m)

opt   : structure with options
opt.model_str = 'wave' (other solver are not supported at the
     moment)
opt.solver = @bvp2c or @bvp2fdm
opt.nx : number of grid points along channel
opt.ns : base for logarithmic spacing of grid points, 1 :
    linear spacing

bc  : structure array of boundary conditions
       r, row 1..2 : left and right end, respectively
       c, column 1 : mean (river) component
                2..n : condition form column-1 frequency
                    component

       (       q(1)*(p(1) y^-(x0) + p(2) dy^-/dx(x0) ...
            +  q(2)*(p(1) y^+(x0) + p(2) dy^+/dx(x0) ) = bc(c
                ,r).val
                 = val(0)

bc(c,r).var : Quantity, either 'z' or 'Q'
bc(c,r).val : complex amplitude of chosen variable
             (c.f. (1 + 0i) [m] for surface elevation
                 amplitude of 1m)
             (value has to be real for mean component)
             mean component requires z and Q to be specified
                 at opposit ends
bc(c,r).p  : factor for Dirichlet p(1) or Neumann p(2)
    condition
             p = [1,0] : pure Dirichlet
             p = [0,1] : pure Neumann
             sum of abs(p) must be nonzero for each end and
                 each frequency component
bc(c,r).q  : factor for left and right going wave, only
    available for bvp2c
             q = [1,1] : total water level / discharge
             q = [1,0] : only left going wave
             q = [0,1] : only right going wave
             q has no meaning for the mean component and is
                 ignored
             q is only supported by bvp2c,
             bvpfdm uses default q = [1,1]
             sum of abs(q) for each frequency component must
```

```
                    be zero
```

207.1 River_Tide

```
river tide in a single 1D channel
TODO split in two classes:
one that stores data (RT_Solve), one that provides equations (
    RT_Analytic)
```

207.2 bc_transformation

207.3 bcfun

```
Robin (mixed) boundary conditions for the river tide,
supplied for each frequency component,
wrapper that copies values are from the member struct "bc"

      q*(p*Q_1^- + (1-p)*dQ_1^-/dx
input :
       x      : coordinate (left or right end)
       id,ccdx : frequency component index
              (1 = 0 omega (mean), 2 : 1 omega, 3 : 2 omega, ...
columns of bc : frequency
rows of bc, left, right boundary
output :
       p : [2x1] linear combination of Dirichlet and Neumann
           boundary condition
         p(1) -> weight Dirichlet boundary condition
           p(2) -> weight Neumann boundary condition
     q linear combination of left and right travelling (incoming and
          outgoing) wave
           q(1) weight left going wave
         q(2) weight right going wave
       rhs = 0 -> homogeneous boundary condition


function [rhs, p, q, obj] = bcfun(obj,x,y,ccdx)
```

207.4 check_continuity

207.5 check_momentum

207.6 d2au1_dx2

```
second derivative of the tidal velocity magnitude

note: this is for finding zeros,
      the true derivative has to be scaled up by z
```

207.7 d2az1_dx2

```
second derivative of the tidal surface elevation

note: this is for finding zeros,
      the true derivative has to be scaled up by z
```

207.8 decompose

```
decompose the tide into a right and left travelling wave,
i.e. into incoming and reflected wave
TODO subtract forcing term
```

207.9 discharge2level

```
tidal component of surface elevation determined from tidal
    discharge

by continuity :

dz/dt + dq/dx = 0
=> i o z = - dq/dx
=>    z = -1/(io) dq/dx
=>    z = 1i/o dq/dx

TODO allow Q as input
TODO rename into Q1_to_z1
Mon  7 Oct 19:04:14 PST 2019 : added correction for change of width
```

207.10 dkq_dx

```
along-channel derivative of the wave number of the discharge
neglects width variation

TODO, rederive with g as variable
```

207.11 dkz_dx

```
along channel derivative of the wave number of the tidal surface
    elevation
ignores width variation dh/dx and second order depth variation (d^2
    h/dx^2)
TODO rederive with g symbolic
```

207.12 even_overtide_analytic

207.13 friction_coefficient_dronkers

```
friction coefficient according to Dronkers

the coefficients are semi-autogenerated

c.f. dronkers 1964
c.f. Cai 2016

p = [p0,p1,p2,p3];
alpha = Ur/Ut = river velocity / tidal velocity amplitude = (umax+
    umin)/(umax-umin)

function p = friction_coefficient_dronkers(alpha,order)
```

207.14 friction_coefficient_godin

```
friction coefficient according to Godin
these coefficients are identical to Dronker's for U_R = phi = 0

function G = friction_coefficient_godin(obj,phi)
```

207.15 friction_coefficient_lorentz

```
friction coefficient according to Lorent'z
identical to Dronker's coefficient for zero river flow
and a single frequency component
c.f. Cai
c.f. Dronkers

function L = friction_coefficient_lorentz(obj,phi)
```

207.16 friction_dronkers

```
friction determined by Dronker's method

input :
       u    : velocity time series
         Umid : arithmetic mean of mininmum and maximum velocity
              (not the mean of the velocity, usually non-zero even
                  without river flow)
       Uhr  : half-range of the velocity, less than the sum of
              the frequency amplitudes, except at perigean spring
                  tides

function [uau_sum uau p] = friction_dronkers(u,Umid,Uhr,order)
```

207.17 friction_exponential_dronkers

```
friction coefficicients for the frequency components computed by
    Dronkers method
c.f. Dronker's 1964 eq 8.2 and 8.4
Note: Cai dennominates alpha as phi

function [c uau uau_ p] = friction_trigonometric_dronkers(u,dp,Umid
    ,Uhr,order,psym)
```

207.18 friction_godin

```
compute friction with the method of Godin
```

207.19 friction_lorentz

207.20 friction_quadratic

```
friction determined by Dronker's method
```

207.21 friction_trigonometric_dronkers

```
friction computed by the method of Dronkers
expressed as coefficients for the frequency components
c.f. dronkers 1964 eq 8.2 and 8.4
Note: Cai dennominates alpha as phi
```

207.22 friction_trigonometric_godin

```
friction computed by the method of Godin
expressed as coefficients of the frequency components (
    trigonometric form)

 function [c, uau] = friction_trigonometric_godin(obj,u,dp,Umax)
```

207.23 friction_trigonometric_lorentz

```
friction computed by the method of Lorent'z
expressed as coefficients of the frequency components (
    trigonometric form)
```

207.24 generate_delft3d

207.25 init

```
provide initial condition by solving the backwater equation for
    surface level
TODO this should not be solved as a ivp but included in the bvp
    iteration
TODO generate the mesh here and precompute fixed values instead of
    passing functions
TODO Q0 should not be a function
function obj = init(obj, Xi)
```

207.26 mwl_offset

```
offset of the tidally averaged surface elevation caused by tidal
    friction
Linear estimate of the mean water level offset (ignoring feed-back
    of tide)
```

207.27 mwl_offset_2

207.28 mwl_offset_analytic

207.29 odefun

```
coefficients of the backwater and wave equation for river-tides
```

207.30 odefun0

```
coefficients of the backwater equation for the river tide
TODO merge with backwater
```

207.31 odefun_advective_acceleration

207.32 odefun_friction

207.33 odefun_ghof

207.34 odefun_swe_jacobian

207.35 odefun_width

207.36 odefunk

```
coefficients of the ordinary differential equation of the k-th
    freqeuncy
component of the tide

f1 Q'' + f2 Q' + f3 Q + f4 = 0

TODO rename f into c
TODO better pass dzb_dx instead of dz0_dx
TODO aa, oh and gh terms are not tested for width ~= 1
```

207.37 solve

```
call stationary or non-stationary solver respectively

function obj = solve(obj)
```

207.38 solve_swe

```
determine river tide by the fully non-stationary FVM and then
    extract the tide
 this is experimental and not yet fully working
```

207.39 solve_wave

```
solve for the oscillatory (tidal) componets

function obj = solve_wave(obj)
```

207.40 wave_number_analytic

```
analytic expression of the wave number

valid for both tidally, river dominated and low friction conditions
and converging channels

k     : complex wave number in a reach with constant width and bed
    slope
im(k) : damping modulus (rate of amplitude change)
re(k) : actual wave number (rate of phase change)

c.f. derive_wave_number
```

207.41 wave_number_approximation

```
approximate wave number of the left and right traveling wave for
    variable coefficients

TODO merge with wave_number_analytic

function [k, k0, dk0_dx_rel, obj] = wave_numer_aproximation(obj)
```

208 lib/tide/river-tide/@River_Tide_Cai

Prediction of river tide by the method of Cai
c.f. Cai 2013, Cai 2015

208.1 Gamma

```
Gamma parameter for tidal propagation
c.f. Cai 2014
```

208.2 River_Tide_Cai

```
prediction of river tide by the method of Cai (2014)
```

208.3 river_tide_cai_

```
determine the surface amplitude of the river-tide
c.f. Cai
```

208.4 rt_quantities

```
determine the quantities that determine the tidal propagation
c.f. Cai

Note: this computes 4 unknowns following Cai, however,
lambda, mu and epsilon can be substituted
making it an equation in one unknown (delta) only
```

209 lib/tide/river-tide/@River_Tide_Empirical

```
Empirical fit to measurement and prediction (from tide at sea and
   river discharge)
of the river tide
```

209.1 River_Tide_Empirical

```
class for fitting models to at-a-station time series of tidal
    elevation
```

209.2 fit_amplitude

```
fit the oscillatory components
```

209.3 fit_mwl

```
fit the tidally averaged water level
```

209.4 fit_phase

```
fit the phase of the oscillatory components
```

209.5 fit_range

```
fit the tidal range
```

209.6 predict_amplitude

```
predict the oscillatory components
```

209.7 predict_mwl

```
predict the mean water level
```

209.8 predict_phase

```
predict tidal phase
```

209.9 predict_range

```
predict the tidal range
```

209.10 rt_model

```
select the model for fitting
```

210 lib/tide/river-tide/@River_Tide_JK

```
empirical analysis and prediction of river tides by the method of
    Jay and Kukulka
```

210.1 River_Tide_JK

210.2 damping_modulus

```
damping modulus of the river tide
c.f. Jay and Kukula
function r = damping_modulus(obj,h0,b,Qr)
```

210.3 mean_level

```
tidally averaged surface elevation
c.f. Jay and Kukulka
```

210.4 rivertide_predict

```
predict river tide by the method of jay and kukulka
TODO rename
```

210.5 rivertide_regress

```
Regression of tidal coefficients according to Jay & Kulkulka

coefficients of the r-regression factor 2 apart for specis (jay C7)
this can be repeated for each tidal species (diurnal, semidiurnal)
```

210.6 tidal_discharge

```
tidal discharge
c.f. Jay and Kukulka
function Qt = tidal_discharge(obj,x,R0,h0,b,Qr)
```

210.7 tidal_range

```
predict tidal range
```

211 lib/tide/river-tide/@River_Tide_Map

hash container for a set of River_Tide predictions for different boundary conditions

211.1 River_Tide_Map

```
container class to store individual river tide scenarios
```

211.2 fun

```
compute river tide for a scenario with specific boundary conditions
     and store it in the hash,
or retrive the scenario, if it was already computed
```

211.3 key

```
key for storing a scenario

function [key obj] = key(obj,varargin)
```

211.4 plot

```
quick plot of scenario result

function obj = plot(obj,Xi,Q0,W0,S0,z1_downstream,cd,zb_downstream,
    omega,q,opt)
```

212 lib/tide/river-tide/@River_Tide_Network

predict tides in a fluvial channel network

212.1 River_Tide_Network

```
tide in a fluvial delta channel network, extension of 1D river tide
the network is a directed graph
TODO convert from trig-to exponential form
```

212.2 discharge_amplitude

```
discharge amplitude
```

212.3 mean_water_level

```
predict the mean water level
```

212.4 plot_mean_water_level

```
plot tidally averaged water level
```

212.5 plot_water_level_amplitude

```
plot surface elevation amplitude
```

212.6 solve

```
solve for the tide in a fluvial chanel network

boundary condition at end points not connected to junctions
       [ channel 1 id, endpoint id (1 or 2), s0, c0
       ...
        channel n id, endpoint id (1 or 2), s0, c0]

conditions at junctions are specified as cells
        each cell contains an nx2 array
        n : number of connecting channels
        [channel id1, endpoint id (1 or 2), ...
        channel idn, endpoint id (1 or 2)]

every tidal species for each channel has 4 unknowns
these are 2x2 unknowns for the sin + cos of left and right going
    wave
```

212.7 water_level_amplitude

```
predict the surface elevation amplitude
```

213 lib/tide/river-tide

```
analysis and prediction of river tides

Sub-Classes:

@River_Tide
       - prediction of river tide in a backwater affected river with
            a sloping bed
@River_Tide_Cai
       - prediction of river tide, method of Cai
@River_Tide_Empirical
       - prediction of river tide, empirical
@River_Tide_JK
       - prediction of river tide, empirical after Jay and Kukulka
@River_Tide_Map
       - mulitple-scenaria container for River_Tide
@River_Tide_Network
       - extension of River_Tide to networks
```

213.1 damped_wave_bvp

```
solved damped wave equation
z'' + a z = 0
z(0) = z0, z(L) = 0
```

213.2 damped_wave_ivp

```
linearly damped wave in rectangular channel
solve tide as initial value problem
damped wave approximation

z'' + a z = 0

x_t = Ax + b
```

213.3 damping_modulus_river

```
damping modulus of the tidal wave for river flow only
```

213.4 rdamping_to_cdrag_tide

```
converts damping rate to drag coefficient
c.f. friedrichs, ippen harleman
```

213.5 river_tide_godin

```
analytic solution to the river tide formulated as boundary value
    problem
in a river with finite length

c.f. Godin 1986
```

213.6 rt_celerity

```
celerity of the tidal wave
```

213.7 rt_quasi_stationary_complex

```
quasi-stationary solution of the SWE
TODO staggered grid does not help: q1' needed
```

213.8 rt_quasi_stationary_trigonometric

```
quasi statinary form of the SWE
```

213.9 rt_reflection_coefficient_gradual

```
reflection coefficient for gradual varying cross section geometry
without damping
```

213.10 rt_wave_equation

```
solve river tide as boundary value problem

input:
omega : [nfx1] angluar frequency of tidal component, zero for mean
    flow
reach : [nrx1] struct
```

```
.L   : [1x1] length of reaches
        .width(x,h)   width
        .bed(x,h)     bed level
        .surface(x,h) surface elevation
        .Cd(x,h)      drag coefficient
.bc  : [nd,nf] boundary/junction conditions
        bc(id,if).type : {surface, velocity, discharge} (dirichlet)
        bc(id,if).val : value
opt  : [1x1] struct
      - constant surface elevation
      - deactivative advective acceleration
      .dx : spatial resolution

dimensions:
      nr : nurmber or reaches
      nd : upstream/downstream index
      nf : frequency index
```

213.11 rt_z2q

```
determine tidal discharge from water level for tidal wave
```

214 lib/tide/river-tide/test/test

214.1 test_bvp2c_sym

214.2 test_celerity

214.3 test_characteristic_rate_of_change

214.4 test_dronkers_compound

214.5 test_friction_dronkers

214.6 test_friction_dronkers2

214.7 test_fv_compare_schemes

214.8 test_fv_convergence

214.9 test_power_series

214.10 test_reflection_coefficient_gradual

214.11 test_ricatti

214.12 test_river_tide_models

214.13 test_rt_reflection

214.14 test_rt_zs0

214.15 test_swe

214.16 test_utm2latlon

214.17 test_wave_twopass

215 lib/tide/river-tide/test

215.1 test_bvp2c2

215.2 test_complex_even_overtide

215.3 test_fourier_power_exp

215.4 test_friction

215.5 test_reflection

215.6 test_rt_wave_number

215.7 test_tidal_river_network

215.8 test_tidal_river_network_z0

215.9 test_tide_slack_exp

215.10 test_wave_number_godin

215.11 test_wave_numer_aproximation

216 lib/tide/river-tide

```
analysis and prediction of river tides

Sub-Classes:

@River_Tide
       - prediction of river tide in a backwater affected river with
            a sloping bed
@River_Tide_Cai
       - prediction of river tide, method of Cai
@River_Tide_Empirical
       - prediction of river tide, empirical
@River_Tide_JK
       - prediction of river tide, empirical after Jay and Kukulka
@River_Tide_Map
       - mulitple-scenaria container for River_Tide
@River_Tide_Network
       - extension of River_Tide to networks
```

216.1 tidal_ellipse

```
 tidal ellipse, numerical ode solution
```

216.2 tide_slack_exp

216.3 wave_number_tide

```
wave number of the tide without river flow
c.f. friedrichs, ippen harleman
output :
k : wave number, such that
               z(t,x) = z1(t,0) exp(1i*(omega*t-k*x))

 re(k) : rate of phase change
-im(k) : damping rate

function [k k_low k_high] = damping_modulus_tide(omega,cd,h0,az1)
```

216.4 wavetrainz

```
determine river tide by iterated integration of the surface
    elevation
```

216.5 wavetwopassz

```
two pass solution for the linearised wave equation, for surface
    elevation
```

217 lib/tide/test/river-tide

217.1 example_river_tide

217.2 example_river_tide_map

217.3 river_tide_test

217.4 river_tide_test_01

217.5 river_tide_test_02

217.6 river_tide_test_03

217.7 river_tide_test_04

217.8 river_tide_test_05

217.9 river_tide_test_06

217.10 river_tide_test_07

217.11 river_tide_test_08

```
hold on;
plot(x,abs(z),'--');
hold on;
plot(x,angle(z),'--');
```

217.12 river_tide_test_09

217.13 river_tide_test_10

217.14 river_tide_test_11

217.15 river_tide_test_12

217.16 river_tide_test_13

217.17 river_tide_test_plot

218 lib/tide/test

218.1 test_tidal_harmonic_analysis

219 lib/tide

```
analysis prediction of tides in rivers and estuaries by empirical
    and theoretical methods
```

219.1 tidal_constituents

219.2 tidal_energy_transport_1d

```
 energy transport of a tidal wave
```

219.3 tidal_envelope

```
envelope of the tide

input : t time in days
       f surface elevation
ouput : tl time of low water
       vl surface elevation at low water
       ldx index of low water
       th time of high water
       vh surface elevation at high water
       hdx index of high water
       ndx neap index
       sdx spring index
       dmax:
       drange: range per day
```

219.4 tidal_envelope2

```
surface levelation envelope of the tide
low water, high water and tidal range for lunar each day


input:
      time  :
      L     : surface elevation
      order : interpolation order (default 2)
ouput:
      timei : vector eqispaced
      lmini : minimum level
      lmaxi : maximum level
      rangei : range
      midrangei : (min + max)/2, usually different from mean
      phii  : pseudo phase

Note: the pseudo phase phi jumps, this is because if the tide is
    semidiurnal,
     sometimes the lower hw becomes the next day higher then than
        the
      current high water, e.g. there is no smooth transition by
     51min but a jump by 12h
```

219.5 tidal_harmonic_analysis

```
tidal_harmonic analysis
```

219.6 tidal_range_exp

219.7 tidal_range_tri

220 lib/tide/tide-savenije

220.1 savenije_phase_lag

```
phase lag of high and low water

phi : u_river/u_tide < 1

delta_eps_hw = omega*(t_hws - t_hw)
delta_eps_hw = omega*(t_lws - t_lw)

c.f. savenije
```

220.2 savenije_tidal_range

```
tidal range

based on Savenije 2012

x    : distance to river mouth
eta  : range
eta0 : range at river mouth
hbar : mean water depth
phi  : velocity ratio u_tide/u_river
       note: this varies in strongly convergent estuaries
K    : mannings coefficient
I    : residual surface slope I
```

220.3 savenije_tidal_range1

```
tidal range

based on Horrevoets/Savenije, 2004

H0   : tidal range at river mouth
h0   : initial water depth
v    : velocity scale
b    : convergence length
sine : phase lag
K    : Mannings coefficient
Q_r   : river discharge
```

220.4 savenije_timing_hw_lw

```
time of high water and low water
c.f. savenije 2012
```

220.5 tide-savenije

221 lib/tide

```
analysis prediction of tides in rivers and estuaries by empirical
    and theoretical methods
```

221.1 tide_low_high_exp

221.2 tide_low_high_tri

222 root

Root folder of the source code belonging to the doctoral thesis:

"Multi-Scale Monitoring and Modelling of the Kapuas River Delta", Karl K\"aster, 2019,

and master thesis:

"Computing the Spectrum of the Confined Hydrogen Atom", Karl K\"astner, 2012.

Installation instructions:

1) Install Matlab
2) Install subversion (svn) and add subversion to the search path, so that
   it can be called from Matlab
3) Checkout this umbrella-project:
   svn checkout https://github.com/karlkastner/root/trunk root/
4) Start Matlab
5) Change into this directory ('root/')
6) Run the Matlab script "startup" located in this directory

The script then fetches the sub-repositories and adds them to the
Matlab search path

Note:

The code upload is work in progress, more parts will be subsequently
documented, added and tested.

This is experimental code. Use it wisely and at your own risk.

Licence:

222.1 load_svn_externals

```
script emulating svn:external, which is not supported by GitHub
```

222.2 quick_data_download

223 sanggau

```
processing of sanggau water level, adcp and hadcp measurements
processing of measurements of Sanggau station, Kapuas River km 290,
    Kalimantan Barat, Indonesia
```

223.1 sanggau_align_transect

223.2 sanggau_backscatter_coefficient

223.3 sanggau_batch

223.4 sanggau_boat_velocity

223.5 sanggau_calib_table

223.6 sanggau_check_error

223.7 sanggau_compare_hadcp2rc

223.8 sanggau_compare_hadcp2rc_2

```
ytick(-15:5:15); ylim([-20 15]); yticklabel(sprintf('%d%%\n
    ',(-15:5:15)'))
```

223.9 sanggau_compare_hadcp2rc_3

```
yticklabel(sprintf('%d%%\n',(-15:5:15)'))
```

223.10 sanggau_compare_models

223.11 sanggau_critical_flow_depth

223.12 sanggau_cs_area

223.13 sanggau_dgps_bt_comparison

223.14 sanggau_energy_slope

223.15 sanggau_error_correlation

```
% mode 2 : direct residuals of f_z and not those given by
    lienarisation are used
```

223.16 sanggau_error_h_u

223.17 sanggau_error_in_area

223.18 sanggau_export_coordinates

223.19 sanggau_gsd

223.20 sanggau_hadcp_calibration

223.21 sanggau_hadcp_correction_data

223.22 sanggau_hadcp_velocity_variation

223.23 sanggau_instationary_rating_curve

223.24 sanggau_ivm

223.25 sanggau_load_bed_level_2016

223.26 sanggau_load_gsd

223.27 sanggau_load_hadcp

223.28 sanggau_load_pilot

223.29 sanggau_maximum_vs_average_velocity

223.30 sanggau_mean_discharge

223.31 sanggau_metadata

223.32 sanggau_optimal_filter_length

223.33 sanggau_photmoetric_level_2016_11

223.34 sanggau_plot_backscatter

223.35 sanggau_plot_backscatter_flux

223.36 sanggau_plot_bathymetry_2d

223.37 sanggau_plot_bathymetry_2d_1

223.38 sanggau_plot_bathymetry_curvature

223.39 sanggau_plot_bed_profile_1d

223.40 sanggau_plot_bed_profile_1d_2

223.41 sanggau_plot_bed_profile_1d_3

223.42 sanggau_plot_bottom_track

223.43 sanggau_plot_cross_section

223.44 sanggau_plot_discharge_bart

223.45 sanggau_plot_error

223.46 sanggau_plot_hadcp_discharge

223.47 sanggau_plot_rating_curve

223.48 sanggau_plot_rc_mcmc

223.49 sanggau_plot_stage_change

223.50 sanggau_plot_surface_slope

223.51 sanggau_plot_transverse_velocity_profile

223.52 sanggau_plot_velocity_nz

223.53 sanggau_plot_vertical_profile

223.54 sanggau_plot_vertical_profile_parameter

```
% split
```

223.55 sanggau_plot_z0_2d

223.56 sanggau_plots

223.57 sanggau_process_discharge

223.58 sanggau_process_discharge_bart

223.59 sanggau_quick_plot

223.60 sanggau_rating_curve

223.61 sanggau_rc_mcmc

223.62 sanggau_rc_vs_ivm

223.63 sanggau_redistribution_by_bathymetry

223.64 sanggau_rouse_profile

223.65 sanggau_sdm_scale_vs_depth

223.66 sanggau_stage_acf

223.67 sanggau_std_u_and_z

223.68 sanggau_test_discharge

223.69 sanggau_theoretic_sediment_transport

223.70 sanggau_transverse_velocity_profile

223.71 sanggau_velocity_direction

223.72 sanggau_velocity_profile

223.73 sanggau_veloctiy_profile_rozovskii

223.74 sanggau_z_0_campaigns

223.75 sanggau_z_0_convergence

223.76 sanggau_z_0_optimal_R

```
fprintf('progress: %d%%\n',round(100*(idx-1)/length(R)));
```

224 root

```
Root folder of the source code belonging to the doctoral thesis:

"Multi-Scale Monitoring and Modelling of the Kapuas River Delta",
    Karl K\"aster, 2019,

and master thesis:

"Computing the Spectrum of the Confined Hydrogen Atom", Karl K\"
    astner, 2012.

Copyright (C) 2010-2019 Karl K\"astner

Installation instructions:

1) Install Matlab
2) Install subversion (svn) and add subversion to the search path,
    so that
   it can be called from Matlab
3) Checkout this umbrella-project:
   svn checkout https://github.com/karlkastner/root/trunk root/
4) Start Matlab
5) Change into this directory ('root/')
6) Run the Matlab script "startup" located in this directory

The script then fetches the sub-repositories and adds them to the
Matlab search path

Note:

The code upload is work in progress, more parts will be subsequently
documented, added and tested.

This is experimental code. Use it wisely and at your own risk.

Licence:

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
```

224.1 startup