Manual for Package: mesh
Revision 1:6M

Karl Kästner

January 27, 2020

Contents

1 @StructuredMesh

1.1 StructuredMesh

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

1.2 apply_boundary_condition

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

1.3 bc_from_shp

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

1.4 bc_index

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

1.5 bc_isinvalid

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

1.6 block

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

1.7 boundary_chain

```
return chain of boundary points
```

1.8 boundary_direction

```
return direction of boundary segment
```

1.9 boundary_indices

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

1.10 cat

1.11 centreline

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

1.12 child

hierarchical mesh generation (for bifurcations)

1.13 copy

1.14 corner_indices

indices of domain corners

1.15 cut_from_domain

cut subdomain

1.16 export_delft3d_bnd

export the boundary in delft3d compatible format

1.17 export_delft3d_dep

export bathymetry data in Delft3D dep-format

1.18 export_delft3d_grd

export mesh in deltares delft3D grd file format

1.19 export_delft3d_ini

export delft3D compatible initial condition file

1.20 export_shp

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

1.21 extend_straight_reach

1.22 extract_elements

```
element indices from grid
```

1.23 flip_dimension

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

1.24 from_1d_mesh

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

1.25 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
```

1.26 generate_disk

```
generate semicircular domain
```

1.27 generate_from_centreline

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

1.28 generate_rectangle

```
discretize a rectangular domain
```

1.29 generate_structured_grid

```
generate a structured mesh consisting of several sub-meshes
```

1.30 grid_block

```
mesh a subdomain
```

1.31 improve

```
improve (smooth) the mesh
```

1.32 interp_elem2point

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

1.33 mesh_polygon

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

1.34 orthogonality

```
orthogonality of elements
```

1.35 orthogonalize

```
orthogonalize mesh
set x of point coordinates to 1/2
```

1.36 plot

```
plot the mesh
```

1.37 plot_boundary

```
plot the mesh boundary
```

1.38 plot_coupling

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

1.39 plot_orthogonality

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

1.40 quiver

```
quiver plot of velocity
```

1.41 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?)
```

1.42 read_delft3d_grd

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

1.43 smooth_cubic

```
 cubically smooth the mesh coordinates
```

1.44 smooth_curvilinear

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

1.45 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
```

1.46 smooth_simple

```
 smooth the mesh coordinates
```

1.47 smooth_sn

```
smooth the mesh coordinates
```

1.48 snap

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

1.49 statistic

```
compute mesh statistics
```

1.50 to_unstructured_mesh

```
convert to unstructured mesh
```

1.51 transpose_dimension

```
transpose dimensions
```

1.52 vertex_connection_matrix

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

2 @UnstructuredMesh

2.1 UnstructuredMesh

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

2.2 add_element

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

2.3 add_vertex

```
add a vertex
```

2.4 angle

```
interior angles of each element
```

2.5 assign_1d

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

2.6 assign_2d

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

2.7 assign_3d

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

2.8 bnd_1d

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

2.9 boundary_1d

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

2.10 boundary_chain2

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

2.11 boundary_length_and_direction

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

2.12 cat

```
concatenate two meshes
```

2.13 chain_1d

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

2.14 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
```

2.15 check_edge_intersection

2.16 clip

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

2.17 compute_elem2elem

```
set up element2element neighbourhood relation
```

2.18 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
```

2.19 convert_2d_to_1d

2.20 copy

```
copy constructor
```

2.21 cross_section

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

2.22 delete_element

```
delete an element
```

2.23 derivative_matrix_1d

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

2.24 derivative_matrix_2d

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

2.25 derivative_matrix_2d_2

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

2.26 derivative_matrix_3d

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

2.27 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
```

2.28 dual_mesh

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

2.29 edge_length

```
euclidean edge length
```

2.30 edge_midpoint

```
edge mid-points
```

2.31 edges_from_elements

```
edges and boundaries from elements
```

2.32 eigs

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

2.33 elem2edge_

```
pointer of element to edge
```

2.34 elem2elem_matrix

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

2.35 element_area

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

2.36 element_centroid

```
centroids of lements
```

2.37 element_midpoint

```
barymetric centre of elements
```

2.38 elements_from_edges

```
2D elements from edges
```

2.39 eval2pval

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

2.40 export_delft3d_net

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

2.41 export_msh

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

2.42 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
```

2.43 export_shp

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

2.44 facing_element

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

2.45 filter_neighbour

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

2.46 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
```

2.47 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
```

2.48 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!
```

2.49 flip_quality

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

2.50 gaussmat_2d

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

2.51 generate_chews_first

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

2.52 generate_from_centreline_1d

```
generate a mesh from centreline
```

2.53 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
```

2.54 generate_frontal

2.55 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
```

2.56 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
```

2.57 generate_hierarchical

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

2.58 generate_triangle

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

2.59 generate_uniform_1d

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

2.60 generate_uniform_quadrilateral

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

2.61 generate_uniform_tetra

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

2.62 generate_uniform_triangulation

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

2.63 get_facing_and_shared_vertices

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

2.64 grid2tri

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

2.65 import_delft3d_net

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

2.66 import_msh

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

2.67 import_triangle

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

2.68 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);
```

2.69 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);
```

2.70 improve_relocate_global1

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

2.71 improve_relocate_global2

```
improve mesh globally
```

2.72 improve_relocate_global_3

```
improve mesh quality globally
```

2.73 improve_relocate_local

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

2.74 improve_relocate_local_old

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

2.75 improve_topology

```
improve mesh topology
```

2.76 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
```

2.77 insert_steiner_points

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

2.78 integrate_1d

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

2.79 integrate_discharge

```
integrate discharge
```

2.80 interp_1d

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

2.81 interp_2d

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

2.82 interp_fourier

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

2.83 interp_tikhonov_1d

```
interpolation with Tikhonov regularisation
```

2.84 interp_tikhonov_2d

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

2.85 interp_tikhonov_3d

2.86 interpolate_from_boundary

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

2.87 interpolate_point

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

2.88 interpolation_error_1d

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

2.89 interpolation_error_2d

```
interpolate error in 2D
```

2.90 interpolation_error_3d

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

2.91 interpolation_matrix_1d

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

2.92 interpolation_matrix_2d

2.93 interpolation_matrix_3d

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

2.94 isacute

```
determine acute triangles
```

2.95 isobtuse

```
determine obtuse triangles
```

2.96 iterate_smooth2

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

2.97 limit_by_distance

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

2.98 make_elements_ccw

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

2.99 merge_duplicate_points

```
merge duplicate points
```

2.100 merge_facing_blunt_triangles

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

2.101 mesh1

```
mesh in 1D
```

2.102 mesh_1d

```
extract the 1d mesh
```

2.103 mesh_2d

```
extract the 1d mesh
```

2.104 mesh_junctions

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

2.105 nearest_boundary

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

2.106 nedge_

2.107 nonobtuse_refinement

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

2.108 objective_A

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

2.109 objective_T

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

2.110 objective_angle

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

2.111 optimum_angle

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

2.112 orthogonality_quadrilaterals

```
orthogonality condition for quadrilaterals
```

2.113 path

```
path along edges
```

2.114 plot

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

2.115 plot1d

```
plot 1D mesh
```

2.116 plot3

```
plot mesh and values
```

2.117 plotcs

```
plot cross section
```

2.118 project_to_boundary

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

2.119 pval2eval

```
vertex to element value
```

2.120 quad2tri

```
quadrilaterals to triangles
```

2.121 raster_boundary

2.122 recover_edges

```
recover (boundary) edges
```

2.123 refine

```
refine by splitting marked triangles
```

2.124 refine_edge_halving

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

2.125 remove_empty_triangles

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

2.126 remove_isolated_vertices

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

2.127 remove_points

```
remove points and associated elements
```

2.128 remove_quartered_triangles

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

2.129 remove_small_islands

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

2.130 remove_triply_connected_boundary_vertices

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

2.131 remove_trisected_triangles

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

2.132 renumber_point_indices

```
renumber vertex indices
```

2.133 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
```

2.134 restore_acuteness

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

2.135 retriangulate

```
retriangulate the mesh
```

2.136 ruppert

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

2.137 scale_to_boundary

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

2.138 scatterplot

```
scatterplot of data on mesh
```

2.139 section

2.140 segment

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

2.141 smooth2

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

2.142 smooth_1d

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

2.143 smooth_val

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

2.144 smoothness

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

2.145 split3

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

2.146 split_edge

```
split an edge
```

2.147 split_edge_perpendicular

```
split edge perpendicularly
```

2.148 split_elem_1d

```
split a 1d element
```

2.149 split_encroached_edges

```
recursively split encroached edges
```

2.150 split_obtuse

```
split obtuse elements
```

2.151 split_unsmooth_edges

```
split unsmooth edges
```

2.152 statistics

```
compute mesh statistics
```

2.153 streamwise_derivative_matrix

```
streamwise derivative matrix
```

2.154 thalweg

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

2.155 to_single

```
TODO, also with indices
```

2.156 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
```

2.157 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
```

2.158 vertex_distance

```
connectivity of directly connected vertices
```

2.159 vertex_to_edge

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

2.160 vertex_to_element

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

2.161 vertex_to_vertex

```
connectivity matrix between vertices
```

2.162 vertices_1d

2.163 weighed_laplacian_smoothing

```
weighed Laplacian smoothing
```

2.164 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
```

2.165 xys2xy

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

3 grid/@Grid1

3.1 Grid1

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

3.2 binop

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

3.3 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)
```

3.4 fit

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

3.5 predict

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

4 grid/@Grid2

4.1 Grid2

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

4.2 binop

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

4.3 build_index

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

4.4 plot

4.5 predict

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

5 grid/@Grid3

5.1 Grid3

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

5.2 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
```

6 mesh1d

6.1 dxspace

6.2 dxspace2

6.3 dzmesh

6.4 mesh1

6.5 mesh1d

6.6 nlogstep

7 mesh

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

7.1 nxfun

8 optimization

8.1 improve_smooth_insert

8.2 objective0_angle1_barycentric

8.3 objective0_angle2_barycentric

8.4 objective0_angle2_barycentric9

8.5 objective0_angle_2_cartesian

8.6 objective0_angle_inf_cartesian

8.7 objective0_barycentric9

8.8 objective0_pythagoras1_barycentric9

8.9 objective0_pythagoras1_cartesian

8.10 objective0_pythagoras2_barycentric9

8.11 objective0_pythagoras2_cartesian

8.12 objective_3_angle

8.13 objective_A_bnd

8.14 objective_P_angle

8.15 objective_P_angle_scaled

8.16 objective_P_angle_scaled_area

8.17 objective_P_midpoint

8.18 objective_angle

8.19 objective_angle2_barycentric

8.20 objective_angle_p

8.21 objective_angle_scaled_area

8.22 objective_angle_scaled_circumference

8.23 objective_cosa

8.24 objective_cosa_p

8.25 objective_cosa_scaled_side_length

8.26 objective_distance_edge_centre

8.27 objective_distance_edge_centre_perpendicular

8.28 objective_distance_orthocentre_excentre

8.29 objective_incentre_excentre

8.30 objective_length_min_max

8.31 objective_length_var

8.32 objective_thales

8.33 objective_thales_difference

8.34 test_objective_cosa_p

9 mesh

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

9.1 preload_msh

10 sparsemesh/@SparseMesh1

10.1 SparseMesh1

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

10.2 assign

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

10.3 assignS

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

10.4 init

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

10.5 interp

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

10.6 interpS

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

10.7 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
```

11 sparsemesh/@SparseMesh2

11.1 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
```

11.2 assign

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

11.3 assignS

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

11.4 init

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

11.5 interp

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

11.6 interpS

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

11.7 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''
```

12 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
```

12.1 SparseMesh

```
SparseMesh superclass
```

13 test

13.1 test_MMesh_segment

13.2 test_derivative_matrices_curvilinear

14 mesh

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

14.1 test_nxfun

14.2 trimesh_fast