FELICITY

Finite ELement Implementation and
Computational Interface Tool for You

Class and Function Reference Guide

Shawn W. Walker

August 4, 2014

Overview

This quick reference guide lists most of the functionality of **FELICITY** with *short* descriptions. More information about specific classes, class methods, or functions can be found by using MATLAB's `help` command. For example, typing the following at the MATLAB prompt

```
help MeshTriangle.Quality
```

and pressing `enter` will print the following text to the MATLAB display:

```
Quality

    This computes the quality metric of all elements in the mesh.  This uses the
    ratio of the inscribed sphere radius to circumscribed sphere radius.

    [Qual, PH] = obj.Quality(Num_Bins);

    Num_Bins = number of bins to use in histogram (optional argument).

    Qual = column vector of mesh element qualities.
    PH   = plot handle for the histogram (valid if Num_Bins > 0).
```

Therefore, if you want to know the details of how to create a class object, call a method, or execute a function, then use the MATLAB `help` command.

Mesh Classes

- `MeshInterval`: 1-D triangulations
- `MeshTriangle`: 2-D triangulations
- `MeshTetrahedron`: 3-D triangulations

Note: `MeshTriangle` and `MeshTetrahedron` are subclasses of `TriRep` (built-in MATLAB class). `MeshInterval` is a subclass of `EdgeRep` (1-D version of `TriRep`).

Methods For All Mesh Classes

- `Angles`: interior angles of each cell (element).
- `Append_Subdomain`: define sub-domain of mesh.
- `Create_Embedding_Data`: lower level version of `Generate_Subdomain_Embedding_Data`.
- `Create_Subdomain`: lower level version of `Append_Subdomain`.
- `Delete_Subdomain`: remove specific sub-domain from mesh.
- `Generate_Subdomain_Embedding_Data`: return a struct containing embedding information for all sub-domains relative to other (given) sub-domains.
- `Geo_Dim`: return geometric dimension of mesh.
- `Get_Adjacency_Matrix`: adjacency matrix for all mesh vertices.
- `Get_Facet_Info`: embedding and orientation info.
- `Get_Global_Subdomain`: return sub-domain connectivity referenced to global mesh vertex coord.
- `Get_Laplacian_Smoothing_Matrix`: output sparse matrix for "smoothing" the mesh.
- `Get_Subdomain_Cells`: return global mesh cell indices that contain given sub-domain's cells.
- `Get_Subdomain_Index`: internal use only.
- `Num_Cell`: number of cells in global mesh.
- `Num_Vtx`: number of vertices in global mesh.
- `Output_Subdomain_Mesh`: return stand-alone mesh object representation of given sub-domain.
- `Plot`: default plot of global mesh.
- `Plot_Subdomain`: plot sub-domain of global mesh.
- `Quality`: compute quality metric of all mesh cells.
- `Refine`: adaptive refinement of mesh. (not implemented for 3-D meshes.)
- `Remove_Unused_Vertices`: removes all vertices not referenced by the triangulation's connectivity.
- `Reorder`: renumbers the global mesh vertices to give a tighter adjacency matrix.
- `Set_X`: modify global vertex coordinates.
- `Top_Dim`: return topological dimension of mesh.
- `Volume`: compute volume of all cells in the mesh.
- `baryToCart`: convert point coordinates from barycentric to cartesian.
- `baryToRef`: convert point coordinates from barycentric to reference domain coordinates.
- `cartToBary`: convert point coordinates from cartesian to barycentric.
- `circumcenters`: compute circumcenters of cells in the mesh.
- `edges`: return all edges in the triangulation.
- `freeBoundary`: return all "facets" referenced by only one mesh cell.
- `isEdge`: test if pair of vertices is joined by an edge.
- `neighbors`: return the mesh cell neighbor info.
- `refToBary`: convert point coordinates from reference domain coordinates to barycentric.
- `size`: return size of triangulation connectivity.
- `vertexAttachments`: return mesh cells attached to specified vertices.

Methods For `MeshTriangle` And `MeshTetrahedron` Only

- `edgeAttachments`: return cells attached to specified edges.
- `incenters`: return incenters of specified cells.

Methods For `MeshTriangle` Only

- `faceNormals`: return unit normal vectors to specified triangle cells.
- `featureEdges`: return sharp edges of a surface triangulation.

Methods For `MeshInterval` Only

- `edgeTangents`: return unit tangent vectors to specified edge cells.
- `freeBoundary`: return all vertex indices referenced by only one mesh cell (edge).

Note: a mesh cell is the basic unit of a mesh.

- For a 1-D mesh (MeshInterval), the cells are line segments (edges).
- For a 2-D mesh (MeshTriangle), the cells are triangles (faces).
- For a 3-D mesh (MeshTetrahedron), the cells are tetrahedra.

Finite Element Classes

- `FELSymBasisCalc`: for computing and storing symbolic expressions of basis functions.
- `FELSymFunc`: class for manipulating symbolic functions.
- `FEMatrixAccessor`: convenient access of finite element matrices by name.
- `FiniteElementSpace`: define and manipulate a finite element space.
- `ReferenceFiniteElement`: local finite element definition.

Note: we abbreviate finite element as **FE**, Degree-of-Freedom as **DoF**, and Degree-of-Freedom map as **DoFmap**.

Methods For `FELSymBasisCalc`

- `Compose_With_Function`: composes the basis functions (and all its derivatives) with a given function represented by `FELSymFunc`.
- `Fill_Eval`: evaluate basis function (and all its derivatives) at given points.
- `Get_Derivative`: returns a `FELSymFunc` object for the specific (multi-index) derivative you want.
- `Get_Length_Of_Multiindex`: returns the length of the "derivative multi-index" used to store basis function derivatives.
- `Get_Value`: returns basis function evaluations for a given vector component and derivative multi-index.

Methods For `FELSymFunc`

- `Compose_Function`: create a new function from composing two functions, e.g. $h(...) = f(g(...))$, where f is the current "object".
- `Differentiate`: return a function that comes from differentiating the current "object" (function).
- `Eval`: evaluates the function's symbolic expression at given points.
- `input_dim`: number of independent variables the symbolic function has.
- `output_size`: number of components that the function has (can be matrix-valued).

Methods For `FEMatrixAccessor`

- `Get_Matrix`: returns sparse matrix data corresponding to a given "name" of a FE matrix.

Methods For `FiniteElementSpace`

- `Append_Fixed_Subdomain`: store mesh sub-domain where DoFs on that sub-domain are considered *fixed* (Dirichlet condition).
- `Get_DoF_Bary_Coord`: for each DoF in the FE space, returns a cell index in the mesh that contains the DoF and the associated barycentric coordinates.
- `Get_DoF_Coord`: return spatial coordinates of each DoF in the FE space.
- `Get_DoFs`: return list of DoF indices in the FE space.
- `Get_DoFs_On_Subdomain`: return list of DoF indices that are attached to a given sub-domain.
- `Get_Fixed_DoFs`: similar to `Get_DoFs`, except only returns the DoFs that are *fixed* (i.e. by some Dirichlet condition).
- `Get_Free_DoFs`: analogous to `Get_Fixed_DoFs`, except this returns the DoFs that are *free* (not fixed).
- `Get_Zero_Function`: returns a coefficient array (matrix) of all zeros representing the zero function in the FE space.
- `Set_DoFmap`: set the DoFmap, which partially defines the FE space.
- `Set_Fixed_Subdomains`: set several subdomains where the DoFs are considered *fixed*.
- `max_dof`: return largest DoF index in the FE space's DoFmap.
- `min_dof`: return smallest DoF index in the FE space's DoFmap.
- `num_dof`: return number of unique DoF indices in the FE space's DoFmap.

Methods For `ReferenceFiniteElement`

- `Gen_Basis_Function_Evals`: put symbolic basis functions of the reference element into a useful data structure.
- `Gen_Quadrature_Rule`: return quadrature rule on the reference element.
- `Get_Local_DoFs_On_Topological_Entity`: return info on how local DoF indices are associated with topological entities of the reference cell.
- `Get_Nodes_On_Topological_Entity`: return matrix array that specifies the local DoF indices attached to each topological entity of the reference cell; similar to `Get_Local_DoFs_On_Topological_Entity`.

Note: this class is mainly used by the code generator.

Finite Elements Defined In FELICITY

Elements are defined in m-files (scripts). The m-files have the following format, where `Z` is the degree of the element, and `D` is the (topological) dimension of the reference cell on which it is defined.

- Lagrange: `lagrange_degZ_dimD.m`.
- Brezzi-Douglas-Marini: `brezzi_douglas_marini_degZ_dimD.m`.
- Raviart-Thomas: `raviart_thomas_degZ_dimD.m`.

Note:

- The files above can be found in `./FELICITY/Elem_Defn/`.
- The class `FELOutputElemInfo` can be used to print information about an element.

Methods For `FELOutputElemInfo`

• `Print_Basis_Functions`: output basis function definitions in either "pretty" or LaTeX format.
• `Print_DoFs`: create a figure that graphically shows the topological layout of the Degrees-of-Freedom (DoFs) on the reference element.

Code Generation For Allocating DoFs

Procedure:

- Define an array of structs, where each struct is an element definition as in `./FELICITY/Elem_Defn/`.
- Use the command `FEL_Compile_DoF_Allocate` to compile the MEX file.
- See the chapter "Automatically Generating Degree-of-Freedom Maps" in the PDF manual for more information.

Code Generation For Assembling Forms/Matrices

Procedure:

- Write an m-file defining the forms/matrices to be created. See keywords and commands below.
- Use the command `Convert_Mscript_to_MEX` to compile the MEX file that will assemble the matrices.
- See the chapter "Assembling Matrices" in the PDF manual for more information.

Matrix Assembly Keywords

- `Bilinear`: define a Bilinear form.
- `Coef`: define finite element coefficient function.
- `Domain`: define spatial domain.
- `Element`: define a finite element space.
- `GeoElement`: define finite element space for specifying how the geometry of the global domain is represented.
- `GeoFunc`: define geometric function for a given `Domain`.
- `Integral`: define an integral to be used for specifying `Bilinear`, `Linear`, or `Real` forms.
- `Linear`: define a Linear form.
- `Matrices`: used for collecting all forms together and outputting them to the code generation routines.
- `Real`: define a Real form, i.e. a small dense matrix.
- `Test`: define finite element test function.
- `Trial`: define finite element trial function.

Symbolic Variable Representations For `Test`, `Trial`, and `Coef`

- `div`: divergence.
- `ds`: arc-length derivative.
- `grad`: gradient.
- `val`: function value.

Symbolic Variable Representations For `GeoFunc`

- `Kappa`: signed scalar curvature (sum of the principle curvatures).
- `Kappa_Gauss`: Gaussian curvature.
- `Mesh_Size`: local mesh size.
- `N`: normal vector.
- `T`: tangent vector.
- `Tangent_Space_Proj`: tangent space projection matrix.
- `VecKappa`: vector curvature (sum of the principle curvatures multiplied by the normal vector).
- `X`: the identity map (position vector).
- `deriv_X`: derivatives of the local parametrization of the domain.

Methods For `Matrices`

- `Append_Matrix`: append a single form (matrix).
- `Include_C_Code`: store info for including external user written C code.

Code Generation For Interpolating FE Functions

Procedure:

- Write an m-file defining the point-wise interpolations to compute. See keywords and commands below.
- Use the command `Convert_Interp_script_to_MEX` to compile the MEX file that will perform the interpolations.
- See the chapter "Interpolating Finite Element Data" in the PDF manual for more information.

Interpolation Keywords

- `Coef`: define finite element coefficient function.
- `Domain`: define spatial domain.
- `Element`: define a finite element space.
- `GeoElement`: define finite element space for specifying how the geometry of the global domain is represented.
- `GeoFunc`: define geometric function for a given `Domain`.
- `Interpolate`: defines an interpolation expression, which may contain FE `Coef` functions as well as `GeoFunc` functions (i.e. domain geometry).
- `Interpolations`: used for collecting all `Interpolate` objects and outputting them to the code generation routines.

Note: some of the above keywords also appear in **Matrix Assembly Keywords**.

Methods For `Interpolations`

- `Append_Interpolation`: appends a single `Interpolate` object (which contains an expression to interpolate).
- `Include_C_Code`: store info for including external user written C code.

Code Generation For Point Searching

Procedure:

- Write an m-file defining a particular point-search of a `Domain`(s). See keywords and commands below.
- Use the command `Convert_PtSearch_script_to_MEX` to compile the MEX file that will execute the point-search.
- See the chapter "Point Searching On A Mesh" in the PDF manual for more information.

Point-Search Keywords

- `Domain`: define spatial domain.
- `GeoElement`: define finite element space for specifying how the geometry of the global domain is represented.
- `PointSearches`: used for collecting all `Domains` (to be searched) and giving it to the code generation routines.

Note: some of the above keywords also appear in **Matrix Assembly Keywords**.

Methods For `PointSearches`

- `Append_Domain`: appends a single `Domain` object (which is a domain to be searched).
- `Include_C_Code`: store info for including external user written C code.

Note: in most situations, one will combine point-search code with interpolation code.

How To Use Hierarchical Search Trees

FELICITY contains built-in C++ code (compiled as MEX files) that implements a binary tree, *quadtree*, and *octree* for efficient nearest neighbor searching of points in space. Technical note: the search trees are implemented as "bucket trees".

Search Tree Classes

- `mexBitree`: binary tree for storing and searching points in 1-D.
- `mexQuadtree`: quadtree for storing and searching points in 2-D.
- `mexOctree`: octree for storing and searching points in 3-D.

Methods For `mexBitree`, `mexQuadtree`, And `mexOctree`

- `Check_Tree`: checks that points in the tree actually belong to their enclosing leaf cell.
- `Get_Tree_Data`: extract data about the tree, such as the enclosing cell (node) dimensions and points contained there in.
- `Plot_Tree`: plots the tree graphically.
- `Print_Tree`: print the tree info to the MATLAB display.
- `Update_Tree`: take a new set of points and update the tree to accommodate the new point coordinates.
- `delete`: destroy the tree (free its memory).
- `kNN_Search`: for a given set of points, this finds the k nearest neighbors in the tree.

Note: see the chapter "Hierarchical Search Trees" in the PDF manual for more information. In addition, the Google-Code page contains some tutorials.

Note: search trees can help make point-searching of meshes very fast.

How To Generate Meshes

FELICITY contains built-in routines for generating meshes and for simple mesh processing. Some are C++ codes (compiled as MEX files) and others are pure MATLAB functions.

Iso-surface Mesh Generation Classes

- `Mesher2Dmex`: Generate 2-D triangle meshes that conform to an iso-contour.
- `Mesher3Dmex`: Generate 3-D tetrahedral meshes that conform to an iso-surface.

Note: these two classes implement the **TIGER** mesh generator.

Methods For `Mesher2Dmex` And `Mesher3Dmex`

- `Get_Cut_Info`: finds all cut edges and cut points that are cut by the zero level set of a given level set function.
- `run_mex`: generate conforming mesh of the interior of a given iso-surface.

Note: see the section "Unstructured Mesh Generation: 2-D and 3-D" in the PDF manual for more information.

Simple Mesh Generator Routines

- `bcc_tetrahedral_mesh`: generate a 3-D BCC (body-centered-cubic) lattice/tetrahedral mesh of the unit cube.
- `bcc_triangle_mesh`: generate a 2-D BCC lattice/triangle mesh of the unit square.
- `Refine_Entire_Mesh`: take a given 2-D mesh and refine the mesh wherever there is a "marked" triangle.
- `regular_tetrahedral_mesh`: generate a regular tetrahedral mesh of the unit cube.
- `triangle_mesh_of_sphere`: generate a 2-D mesh of the surface of a sphere.

Mesh Smoothers

- `FEL_Mesh_Smooth`: run a Gauss-Seidel iterative ODT (Optimal-Delaunay-Triangulation) smoother multiple times on mesh vertex positions. Works for 1-D, 2-D, and 3-D simplicial meshes.

Simulation Management Classes

- `FEL_AbstractSim`: abstract class for creating specific finite element simulation subclasses. Do not use directly; use `FEL_Sim_Template` instead.
- `FEL_SaveLoad`: class for storing time-series, or indexed, simulation data and reloading it.
- `FEL_Sim_Template`: example concrete subclass of `FEL_AbstractSim`.
- `FEL_Visualize`: class for saving simulation plots and making movies.

Methods For `FEL_SaveLoad`

- `Delete_Data`: delete all files saved by the object.
- `Get_Max_Index`: get largest simulation index of files saved in the object's data directory.
- `Load`: load data that was saved previously.
- `Make_FileName`: create a valid filename for a given simulation index.
- `Save`: save data with a specific file index.

Methods For `FEL_Sim_Template`

- `Assemble_Matrices`: assemble matrices on the object's mesh.
- `Build_System`: build the system matrix to be solved.
- `Define_Finite_Element_Space`: define finite element spaces for the simulation.
- `Define_Mesh`: define global mesh for the simulation.
- `Get_LU`: calls MATLAB's `lu` command; to be used with `Solve_With_LU`.
- `Initialize_Solution`: initialize solution variables (finite element coef. arrays) for the simulation.
- `Solve`: solve the system.
- `Solve_With_LU`: solve a linear system $Ax = b$ by a pre-computed LU decomp; to be used with `Get_LU`.

Methods For `FEL_Visualize`

- `Delete_Plots`: delete all plots saved by the object.
- `Make_Movie`: make a movie.
- `Save_Plot`: save figure to a file.