IGFEM-Curves-2D: Developer's Guide

Marcus Hwai Yik Tan

Created on January 27th, 2016. Last revised on May 15, 2021

1 Structure of program

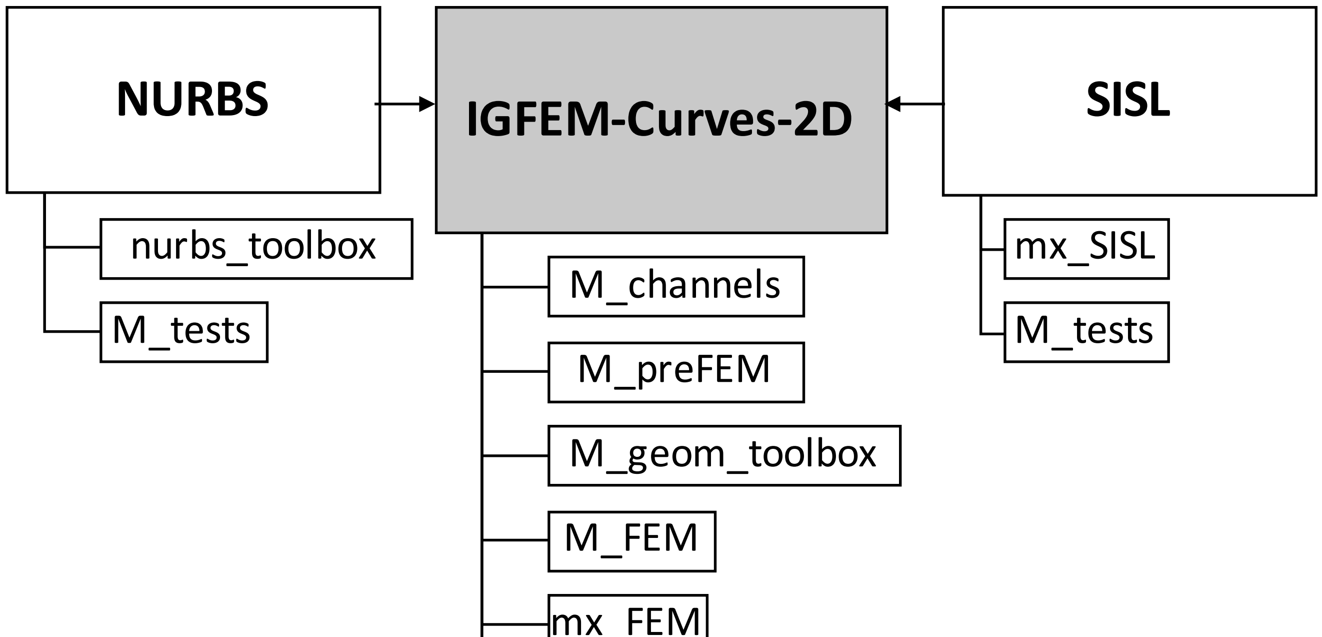

Figure 1: The directories (in bold) and subdirectories of the program.

The program follows a straightforward functional programming paradigm. It consists of a main directory IGFEM-Curves-2D and two other directories NURBS and SISL at the same level as the main directory, as well as other subdirectories as shown in Fig. 1. The scripts are organized into subdirectories based on the types of functions they perform. Subdirectories starting with capital M consist of MATLAB scripts while those starting with mx include MEX functions and C++ functions called by the MEX functions. The IGFEM-curves-2D directory itself contains the `main.m` script for executing the program. The other subdirectories and the types of scripts/codes located in them are described in the subsections Sec. 1.1 to 1.6. The subdirectories M_tests in this directory as well as the other directories (NURBS and SISL) are merely directories that contain MATLAB scripts to test some MATLAB scripts or MEX functions. This is left to the developer to explore on his/her own.

The program also needs the scripts/codes in the directories NURBS and SISL. NURBS contains a NURBS toolbox downloaded from Octave-Forge and based on the original NURBS toolbox by Mark Spink on MATLAB File Exchange. The SISL directory include MEX functions built on the SISL library developed by a group called SINTEF. Some of the MEX functions in the directory also require the Armadillo C++ lineary algebra library. The MEX functions are used for the fast calculation of intersections between common geometrical entities (such as line segment, lines etc) and NURBS or between NURBS entities. The purposes of these functions are clarified in the codes and would not be described here.

1.1 IGFEM-Curves-2D/M_channel

This subdirectory contains scripts that handle the channel networks such as:

1. `create_channels_gui.m` (creates channel input files),
2. `create_channel.m` (an older script that create channels directly),
3. `read_channels.m` (reads channel input file),
4. `write_channel_file.m` (writes channel input files),
5. `plot_channel_network.m` (plots the channels, nodal pressure drops and channel flow rates),
6. `network_pressure_mass_flow_rate.m` (solves for nodal pressures and channel flow rates),
7. `branching_point.m` (finds branching points in a network automatically),
8. `kinks.m` (finds kinks in a network described by linear NURBS so that enrichment nodes can be introduced there).

It also contains scripts related to the channel networks needed by the project Opt-IGFEM-2D such as:

1. `generate_sample_file.m` (generates random initial designs using a few method like Latin hypercube sampling in bounding boxes, random shuffling in bounding boxes, random generation of quads for parallel networks and random generation of vertices followed by subgraph monomorphism to connect to vertices so that the required topology is obtained),
2. `channel_polygons.m` (constructs triangles/quads for imposing geometrical constraints),

3. `update_channels.m` (updates channel control points and diameters in the optimization),

4. `design_params2channel_params_map.m` (maps the channel control point coordinates to the design parameters),

5. `plot_channels_and_design_parameters.m` (plots channels and design parameters).

1.2 IGFEM-Curves-2D/M_preFEM

This subdirectory contains scripts that perform the usual preparation for FEM analysis such as:

1. `generate_uniform_mesh.m` (generates a structural mesh),

2. `set_boundary_conditions.m` (sets Dirichlet and Neumann boundary conditions. Note: in the event that there are enrichment nodes along a Dirichlet boundary, this script also sets up the additional equations for applying the Lagrange multiplier method).

It also contains scripts that prepare the required data for IGFEM analysis such as:

1. `single_edge_curves_intersect.m` (finds intersection between a single edge and the channel network),

2. `edges_curves_intersect.m` (finds intersection between all edges of the mesh and the channel network),

3. `elem_branching_points.m`, `elem_kinks.m`, `kinks_branch_edges_intersect.m` (handles branching points and kinks of the channel network)

4. `element_intersections.m` (determines intersections of all elements with the channel network based on the edge intersections),

5. `parent_element_nurbs.m` (determines which elements with intersections are parent elements, i.e., those that require enrichments, extract the NURBS curve segment within the parent elements, introduce enrichment nodes at branching points and kinks etc),

6. `child_element_nurbs.m` (subdivides each parent element into integration subdomains/child elements),

7. `child_elem_nurbs_surface.m` (constructs the NURBS surface for each integration subdomain)

8. `create_nurbs_enrichment_function.m` (extracts the basis functions associated with the enrichment control points of the subdomain NURBS surfaces and use them as enrichment functions),

There are scripts for mesh refinements/modification such as:

1. `elem_sharing_edge_nodes.m` (calculates sparse matrix the size of number of original nodes $\times$ number of original nodes where the (i,j) and (j,i) entries are the elements sharing the edge with nodes i and j. The matrix is used for mesh refinement by bisection.)

2. `bisection.m` (refines mesh by bisecting triangular elements)

3. `flip_edge.m` (swaps edges of a pair of triangles to avoid certain scenarios or improve mesh quality)

1.3 IGFEM-Curves-2D/M_geom_toolbox

This subdirectory contains more generic geometrical tools used by the scripts in M_preFEM such as:

1. `intersect_edges.m` (finds intersection between edges/line segments quickly),

2. `adjacency_matrix.m` (finds adjacency matrix of the mesh),

3. `inTriangle.m` (determines if a set of points lie within or on the boundaries of a triangle within a specified tolerance. Note: inpolygon could be used but it does not allow for the specification of tolerance).

1.4 IGFEM-Curves-2D/M_FEM

This subdirectory contains scripts that perform NURBS-based IGFEM or polynomial IGFEM such as:

1. `gauss_points_and_weights.m` (provides the gauss points and weights. Note that quadratures for tri elements are stored in the script while those for quad elements are calculated using `lgwt.m`),

2. `estimate_nff_nfp_npp.m` (estimates the sizes of the different partitions (free-free, free-prescribed and prescribed-prescribed) of the stiffness matrix),

3. `assemble.m` (assembles the stiffness matrix and load vector. Note that the stiffness matrix is assembled by constructing triple vectors respectively containing the row indices, the column indices and the entry values. These vectors are then fed to the sparse function to construct the sparse stiffness matrix),

4. `assemble_prescribed_node.m` (assembles the vector of prescribed nodal values),

5. `force_assemble.m` (assembles the load vector including the entries due to the Neumann BC),

6. `compute_regular_element.m` (computes regular element stiffness matrix),

7. `compute_parent_element_nurbs.m` (computes parent element stiffness matrix when NURBS-based IGFEM is selected),

8. `compute_polyIGFEM_element.m` (computes parent element stiffness matrix for polynomaial IGFEM. However, it is no longer in used as assembly is done by the mex function in the mx_FEM subdirectory),

9. `local_coord.m` (calculates local coordinates in a tri/quad element given the global coordinates),

10. `shape_funct.m` (evaluates shape functions of tri/quad elements given local coordinates),

11. `initialize.m` (maps the node number to the equation number. Presribed nodes are mapped to negative equation numbers while free nodes are mapped to positive equation numbers),
12. `streamwise_elem_length.m` (calculates streamwise element lengths for applying SUPG),
13. `solve_matrix_eqn.m` (solves the assembled equations),
14. `interpolate_soln.m` (interpolates the FEM solution for direct plotting and for error analysis),
15. `update_enrichment_node_value.m` (updates the enrichment node values by adding the offsets to the background values. Note that for NURBS-based IGFEM, linear interpolation is used as it is found to be sufficiently accurate for visualization).

Many of these scripts are not used when polynomial IGFEM is selected as the assembly of the stiffness matrix and load vector is done using the MEX function `mx_assemble_sparse.cpp` together with other C++ codes located in the mx_FEM subdirectory.

1.5 IGFEM-Curves-2D/mx_FEM

This subdirectory contains the C++ codes require to create the MEX function `mx_assemble_sparse.mex*` that can be called directly in MATLAB. The function only assembles the sparse stiffness matrix and the load vector but does not solve the assembled equations since the Armadillo library does not contain a sparse solver. There are two header files here:

1. `armaMex.hpp` (contains functions that allow conversion of inputs and outputs to Armadillo data structures. Note that this header file is obtained from the Armadillo package),
2. `assemble.h` (contains some inline functions and function prototypes for the functions contained in the C++ codes).

The purposes of the C++ codes are self-explanatory from their names and are mostly similar to those of the scripts in M_FEM. Only few points warrant discussion here. First, openmp can be used in `assemble.cpp` by commenting the line `omp_set_num_threads(1)`. Second, if one needs to specify an analytical form of the distributed heat source for use with polynomial IGFEM, one needs to modify the file `body_source_functions.cpp`. (The analytical distributed heat source for NURBS-based IGFEM is specified in the file `distributed_heat_source.m` located in the subdirectory M_channels.) The file `conductance.cpp` is for the constant heat flux model (channels.model == 2).

1.6 IGFEM-Curves-2D/M_error_analysis

The following scripts are included in the subdirectory to calculate the L^2- and H^1- errors and norms given a reference solution:

1. `error_L2_H1.m` (assembles the errors squared and the reference solutions norms squared in all the elements),

2. `error_squared_L2_H1_child_element.m` (calculates the errors squared and the reference solution norms squared in a NURBS-IGFEM integration subdomain),
3. `error_square_L2_H1_regular_element.m` (calculates the errors squared and the reference solution norms squared in a regular element),
4. `compute_polyIGFEM_element_error.m` (calculates the errors squared and the reference solution norms squared in a polynomial IGFEM parent element).

2 Data structures

The purpose of this section is merely to give a general idea of the type of data structure used in the program. The definitions of the variables are best found in the comments placed in the scripts/codes or inferred from the variable names. The data structures used by the program in MATLAB are simple: matrice, vector, struct, cell and various combinations of them. For example, `mesh.elem.parent(.).child(.).channelNodes` is a struct variable that contains vector members (`parent(.)` and `child(.)`) with cells (`channelNodes`).

The main variable of the program is `mesh`. Some important members of this variable are:

1. `edges` with some of the members as follow:
   (a) `edge_nodes` (matrix of end node numbers of the edges),
   (b) `itrsect` (a vector of struct variables, each containing information about the intersections of the channels with an edge).
2. `elem` with some of the members as follow:
   (a) `elem_node` (matrix of node numbers of the elements),
   (b) `elem_edge` (matrix of edge numbers of the elements),
   (c) `junc` (a vector of struct variables the size of the number of elements containing information about branching points when applicable),
   (d) `kinks` (a vector of struct variables the size of the number of elements containing information about kinks when applicable),
   (e) `parent` (a vector of struct variables the size of the number of elements containing information about the enrichment nodes, channel number, integration subdomains etc when the element corresponding to an entry is a parent element),
   (f) `Neumann` (struct variable containing information about the loads on the element boundaries when applicable)
3. `node` with some of the members as follow:

(a) `coords` (matrix of the nodal coordinates, each row corresponding to a node. The first `nOriginalNodes` entries correspond to the original nodes and the rest are the enrichment nodes),

(b) `Dirichlet` (struct variable containing information about the Dirichlet boundary conditions applied to the node),

(c) `nOriginalEnrichNode` (for polynomial IGFEM, this number is the same as the total number of nodes. However, for NURBS-based IGFEM with quadratic and higher NURBS, this number only includes both the original nodes and enrichment nodes at the intersections, branching points and kinks but does not include additional enrichment nodes introduced at interior control points).

Inside `mx_assemble_sparse.mex*` or any of the MEX functions in the SISL directory, the data structures are simply converted into corresponding data structures available in the Armadillo library, std::vector or C variables. No custom object is defined except those that already exist in the Armadillo library.

3 Future development

The efficiency of the intersection finder can be improved by sorting the edges and channels into a tree structure. Only those within the same branch need to be check for intersections. Further speed up can be achieved by using MEX functions.

The NURBS-IGFEM assembly can be sped up by using MEX functions. This requires the use of NURBS library such as SISL.