HiMod Tree

Yves Antonio Brandes Costa Barbosa

January 2017

1 Introduction

The aim of this text is to explain the main components of the HiMod tree, so that the new members of the project can understand the main guidelines to organize their branches and to explain the good practices used to code the methods and classes. This repository contains the first version of the HiMod laboratory code for 2-dimensional applications.

The same material can be found in a shared folder of BitBucket. For the sake of simplicity during version control, BitBucket was chosen as sharing device. If you are not familiarized with BitBucket or the `git` functionality, please refer to the following tutorial.

2 What is inside HiMod Tree?

Once you are set with your BitBucket account and the repository folder in your computer, you can start dig into the code. The HiMod Tree project is composed by the +Core, Demos and other auxiliary folders, but before looking inside it is important to understand how the code is organized.

Usually, a typical matlab simulation is run using several `*.m` files, that implement a different function necessary of perform the computations. The problem with this approach is that the notion of hierarchy is lost when the project starts to get bigger (which is the case of HiMod) and eventually the organization is lost due to the huge amount of individual files. This problem motivated the implementation of the HiMod code in an object oriented framework. Although, the object oriented functionality of Matlab is not optimized, by programming using this methodology we are able to keep track of what is happening during the simulation and to detect where to act when modifications are required.

If you are not familiar with the object oriented framework used in Matlab, please refer to the this other tutorial. Here you can find the explanation for all the structures used in the definition of classes, properties and methods of

HiMod Tree. In more details, the folders found inside the project and their content are summarized bellow.

2.1 +Core

It contains the classes representing the main structures necessary to keep the simulations running. They are AssemblerADRHandler, BasisHandler, EvaluationHandler, FilterADRHandler, IntegrateHandler, BoundaryConditionHandler, SolverHandler and UtilityHandler. Specifically, we have:

1. **AssemblerHandler** is responsible for the build methods that creates the main components necessary to create the stiffness matrix, mass matrix and RHS vector. It contains the required methods to solve steady and unsteady ADR partial differential equations. For the sake of organization, when the Stokes and Navier-Stokes assembler methods will be included in the +Core, a second and third assembler class need to be created, respectively;

2. **BasisHandler** is responsible for the evaluating the discretization points in the desired basis function, either for the modal components in the transverse fiber or the nodal components in the supporting fiber. Whenever a set of point need to be evaluated in a different basis, the method must be inserted here;

3. **EvaluationHandler** is responsible for taking the solution of the linear system representing the discretized partial differential equation and return the solution in the display grid. These methods are necessary to plot the solution and compute the resulting error;

4. **FilterADRHandler** contains all the methods performing the Data Assimilation process. This includes both filtering algorithms and necessary methods to perform the algorithm. In the current version of the code, only the linear Kalman Filter is implemented;

5. **IntegrateHandler** contains all of the integration methods used to assemble the matrices of the linear system. In particular, we have the 'Gauss' and 'Gauss-Legendre' methods, that supply the integration nodes depending on the method used, and the 'Quadrature Rule', that insert the quadrature nodes in the mesh of the discretized problem. One important thing to notice is that the current 'Gauss' and 'Gauss-Legendre' methods supply the quadrature nodes inside a predefined interval, so whenever these functions are called we must guarantee, before or after the insertion of the node, that the integration is being made in the right interval;

6. **BoundaryConditionHandler** contains the methods responsible for imposing the boundary conditions in the final matrix. It is also used to define the methods used in the Domain Decomposition approach to well define the divided domains. The current version of the code does not use any of

these methods because it is defined for homogeneous Dirichlet boundary conditions. However, in future application, whenever a different kind of boundary condition need to be applied, the method will be defined here;

7. **SolverHandler** class contains the methods that are called in the demonstration files. In each case, the solver method called will contain all the secondary methods in the remaining classes in order to solve the problem. An usual solver method contains the sequence of actions: call assembler, call linear system solver, call error calculator and call the plotting function. In some special cases, some additional step may be included. For example, when simulating the Data Assimilation process, a filtering step is added right after the solution;

2.2 Demos

The 'Demos' folder contains all of the demonstration files necessary to run the simulations. As specified in more details afterwards, for each new branch of HiMod it is necessary to create an additional file, also unique. An usual demonstration file contains all of the necessary information to run the simulations, such as:

1. **Simulation case:** selects what simulation case will be run;
2. **Discretization parameters:** contains the number of vertical nodes, of horizontal elements, of horizontal nodes and discretization step of the supporting fibre;
3. **Basis properties:** contains the properties of the basis used to discretize the supporting fibre. This includes the degree of the basis and continuity properties;
4. **Boundary conditions:** defines the boundary conditions in the 4 boundaries of the domain (up, down, inflow and outflow). The current version of the code works for homogeneous Dirichlet in the up, down and inflow boundaries and Neumann in the outflow boundary;
5. **Physical domain:** defines the properties of the physical domain where the problem is defined. In the current version of the code contain the profile of the central line and its derivative, the expression of the thickness of the domain (now it only works for $L = 1$), and the expression of the map (Jacobian J) from the physical domain to the computational domain;
6. **Quadrature properties:** sets the number of quadrature nodes that will be used in the build methods to create the system matrices;
7. **Coefficients of the bilinear form:** defines the expression of the diffusion coefficients, transport term and reaction coefficient, that define the ADR problem, as a function of the physical domain.

8. **Exact solution:** expression of the exact solution of the ADR problem used to compute the simulation error of approximated solution.

9. **Force and Dirichlet profile at the inflow:** defines the forcing term and the inflow Dirichlet profile necessary to result the exact solution defined in the previous step;

10. **Solver:** insert all of the simulation specifications as properties of the solver object and call the respective solver method to compute the approximated solution and analyse it;

Depending on the branch, other specifications may be inserted to run the simulation. For example, in the case of unsteady ADR problems, the time domain must be specified.

2.3 Matlab Documentation

Contains the publish files generated automatically using the Matlab interface. The code must be always well specified and follow the correct templates given by MathWorks in order to facilitate the organization when the publish is used.

2.4 Simulation Results

Contains all of the results coming from the simulation. For the same of organization, each HiMod branch must have only one folder, that can be organized. Along the code, whenever the export methods are called to export simulation data, make sure that the receiving address is set to the correct branch sub folder inside the 'Simulation Results' folder.

3 Contribution guidelines

If you just joined the project, it is important to be aware of the general guidelines used to modify the current version of the code. First of all, it is necessary to create a personal branch of work from the master code present in the BitBucket shared folder. Do not, under any circumstances commit a piece of code into the master copy without being sure of the modifications and without cleaning up all the debug operations. This will save time and will preserve the work flow of the other members of the project.

Once your branch is created, you can start implement modifications. To do so, please flow the instructions present in the *Good practices guide for HiMod Tree*, found in the same folder of this text. Besides the guide, we also ask new contributors to account for the following orientations through out their work:

1. Create one, and only one, test file in the 'Demos' folder for your branch of work. For the sake of organisation follow the general programming guidelines and structure already used in other test files;

2. Check the existing classes in the +Core folder and place correctly the methods you need to perform your computations;

3. In the case you see fit to create a new class to include a set o special methods, you may do it. However, if the new methods you want to create are only auxiliary, they should be included in the Util class. ***In the current version of the code, the UtilHandler class has not been created yet, so any auxiliary methods should be inserted in the Util folder. However, we insist that the new methods should follow the programming guidelines detailed in the guide***;

4. Make sure that all your simulation results are organized in the 'Simulation results' folder. If you use an automatic file generation to export data during the simulations, make sure that the export address is set inside your assigned simulation folder;

5. Once you have finished implementing all the modifications and cleaning your code, make sure to update the Matlab documentation present in the 'Matlab Documentation' folder. Make sure there are no further modifications required before you change the documentation and commit your code;

An import consideration that all contributors should have in mind is to make reference the material used to produce the code. By material we mean specific scientific papers, tutorials and pre-existing code. This way, when the code will be revisited in the future, it will be possible to identify the exact source of the algorithms.

4 Who do I talk to?

In case you have any doubts about the structure or functioning of the code or if you want to report a bug, please contact the code administrator.

CODE ADMINISTRATOR

- Name: Yves Antonio BRANDES COSTA BARBOSA
- Email: yvesantonio.brandes@mail.polimi.it

PROJECT ADMINISTRATOR

- Name: Simona PEROTTO
- Email: simona.perotto@polimi.it