
Programming for Engineering and Science

Homework 1: "Matlab and FE"

Albert Capalvo and Lisandro Roldan

February 29, 2016

1 Introduction

A Matlab program for solving temperature diffusion problems was developed. Some incomplete code was provided by the professors for the solution of a 2D problem using quadrilateral linear elements. That code was modified and extended in order to solve the same problem with different 2D and 3D finite elements and exporting its results for visualization using Paraview.

2 Code description

The whole set of Matlab files can be downloaded from the following GitHub repository:

https://github.com/lisandroroldan/PES_HW1_Roldan_Capalvo

The principal program consists on a main file, several functions and input data. All of them can be found in the folder *SS_General*.

Additionally folders *SS_GiD* and *FEM.gid* contain the finite element program and GiD Problem type that was used to generate additional meshes and reference results.

2.1 Main program - part 1

File: main.m

When executed, the user will find a message on the console asking for the name of the problem to be solved.

Using the load function, the problem's geometry (node location and connectivity matrices) and groups that define the boundary conditions are imported from data files and converted into Matlab variables. Automatically, the program will identify the element type to be used. The following types of elements are allowed:

2D Elements

- Triangular linear (3 nodes)
- Triangular quadratic (6 nodes)
- Quadrilateral linear (4 nodes)
- Quadrilateral quadratic (8 nodes)

3D Elements

- Tetrahedral linear (4 nodes)
- Tetrahedral quadratic (8 nodes)
- Hexaedron linear (8 nodes)
- Hexaedron quadratic (20 nodes)

From then on, the implemented code will vary depending on the number of dimensions of the problems and the element type selected.

The code executes several functions in order to solve the Finite Element problem, which will be explained below.

2.2 Calculation of number of integration points

File: numberofintegrationpoints.m

This function uses the known number of dimensions and amount of nodes per element to return the number of Gauss quadrature integration points to be used.

2.3 Position of Gauss points in normalized coordinate system and integration weights

Files: integrationpoints.m / integrationweights.m

With the same input variables than the last function plus the number of Gauss points, this functions returns a matrix with the standard coordinates for the element in the normalized space and a vector with the weights needed to perform the integration using Gauss quadrature.

2.4 Shape functions and their derivatives

File: shapefunctions.m / shapefunctionderivs.m

Both functions are run inside a loop over the number of integration points, and their outputs are stored in two matrices.

The input for both are the number of dimensions of the problem, the amount of integration points and their position in the normalized space.

The functions choose over many predefined equations to return the desired shape functions and shape functions derivatives with respect to the space coordinates.

2.5 System resulting of discretizing the weak form

Files: CreateMatrix.m / MatEl.m / Isopar.m

This group of nested functions take the node coordinates, connectivity matrix, space dimensions, gauss integration parameters, shape functions and their derivatives to calculate the components of the system of equations to be solved.

The functions "MatEl" and "Isopar", make the transformation from the normalized space to the real one and create the local stiffness matrices and force vectors (in this case the source term is zero, but it can import the information from a data file). The function "CreateMatrix" assembles them to get the global system of equations.

2.6 Main program - part 2

File: *main.m*

Once the global stiffness matrix, force vector and boundary conditions are obtained, the system is solved using the "Lagrange Multipliers Method".

Finally, with the known value of the temperature for every node, a postprocess function is called.

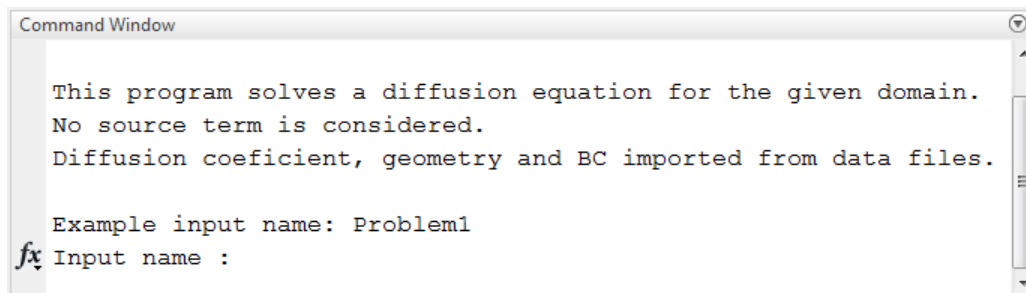
2.7 Post-process

File: *postprocess.m*

This function creates a .vtk file with a structure and format that is readable by Paraview. Once again, the form and information of the file will vary depending on the element type detected at the beginning. The output result is a file that contains information about the coordinates of the nodes, their connectivity, the element types used and the solution of the problem.

3 How of use the program

When the main file is executed, the user will have to introduce via console the name of the problem to solve:



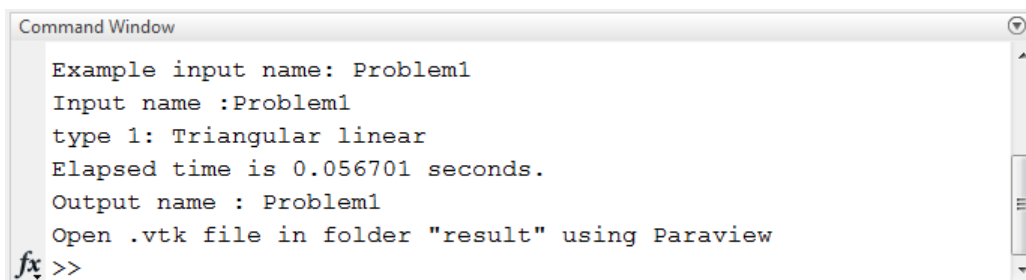
```
Command Window
This program solves a diffusion equation for the given domain.
No source term is considered.
Diffusion coefficient, geometry and BC imported from data files.

Example input name: Problem1
fx Input name :
```

Input of problem name

In case that a different problems with different geometries than the provided ones are needed to be solved, the user will have to copy the input files into the model and element folder. Their name should have the structure: *Name_ nodes*, *Name_ groups*, *Name_ elements* and *Name_ prop*.

Once the program has finished execution, the user will be asked to input the name of the output file



```
Command Window
Example input name: Problem1
Input name :Problem1
type 1: Triangular linear
Elapsed time is 0.056701 seconds.
Output name : Problem1
Open .vtk file in folder "result" using Paraview
fx >>
```

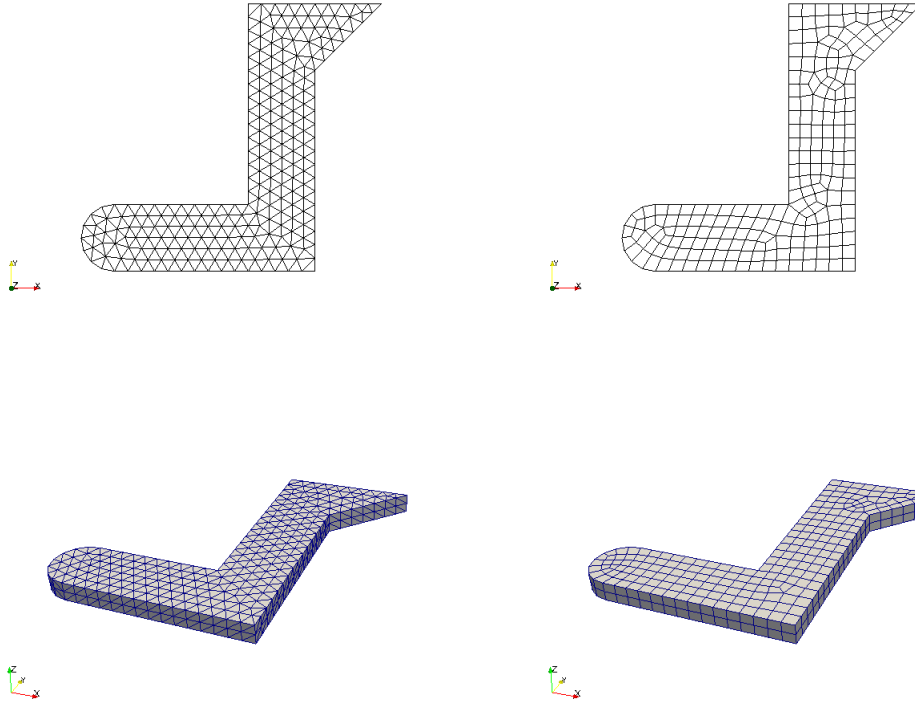
Output file name

The type of element used and the time used for the solving of the problem is informed.

4 Testing and comparison

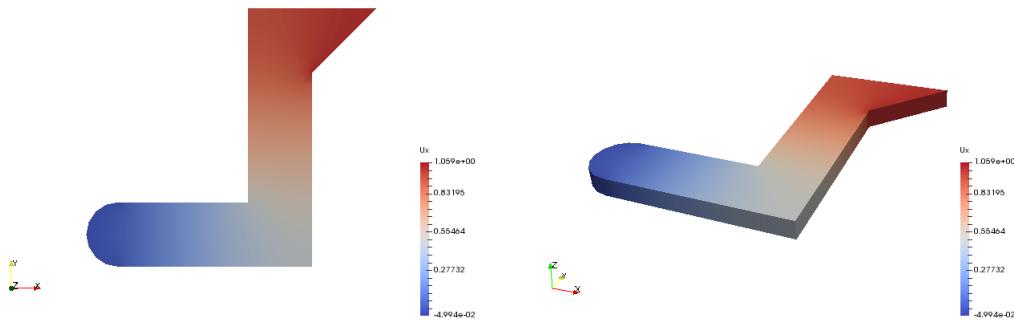
4.1 Results of the supplied meshes

To test the code, all of the eight meshes and groups provided by the professors were used as inputs with the same boundary conditions.



Top-left: 2D triangular mesh (linear and quadratic); Top-right: 2D quadrilateral mesh (linear and quadratic); Bottom-left: 3D tetrahedral mesh (linear and quadratic); Bottom-right: 3D hexahedral mesh (linear and quadratic).

Although graphic results were obtained for all of the meshes, no difference can be appreciated graphically, therefore only two examples are included.



Left: results for 2D quadratic linear elements; Right: results for 3D hexahedral linear elements

Using the function tic-toc of Matlab, the execution time was evaluated, producing the following results:

2D_TRI_LIN	0.056 sec	3D_TET_LIN	0.516 sec
2D_TRI_QUAD	0.149 sec	3D_TET_QUAD	4.308 sec
2D_QUAD_LIN	0.068 sec	3D_HEXA_LIN	0.355 sec
2D_QUAD_QUAD	0.189 sec	3D_HEXA_QUAD	1.941 sec

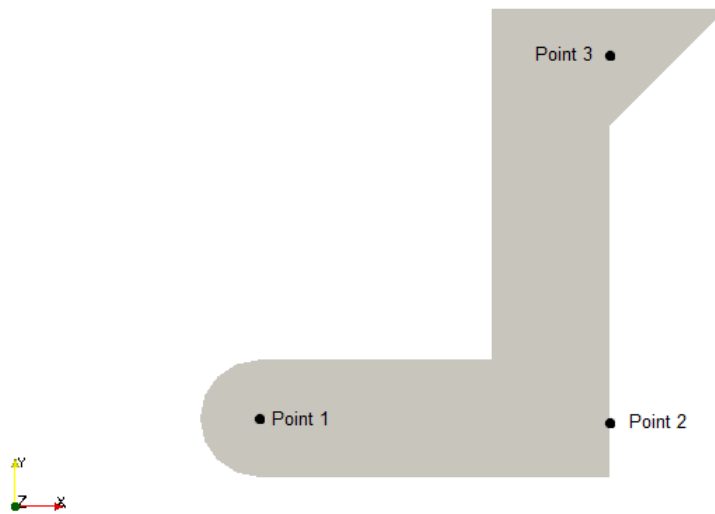
Calculation time

To compare the results another mesh, much denser than the evaluated ones, was used as reference for obtaining the relative error.

Taking advantage that Paraview is able to interpolate the results from the given node values, three points were used to compare results:

Table 4.1: Relative errors, reference mesh: 48.744 tetrahedral linear elements

Problem	Type	Point 1 (-25,-5)		Point 2 (-5,5)		Point 3 (-5,25)	
Ref	3D tet lin	0,04499	Relative error	0,48364	Relative error	0,96662	Relative error
1	2D tri lin	0,04146	7,83%	0,48607	0,50%	0,96950	0,30%
2	2D tri quad	0,04569	1,56%	0,48097	0,55%	0,96478	0,19%
3	2D quad lin	0,04333	3,68%	0,48393	0,06%	0,96756	0,10%
4	2D quad quad	0,04489	0,21%	0,48185	0,37%	0,96562	0,10%
5	3D tet lin	0,04170	7,31%	0,48624	0,54%	0,96960	0,31%
6	3D tet quad	0,04375	2,74%	0,48497	0,27%	0,96797	0,14%
7	3D hex lin	0,04506	0,17%	0,48235	0,27%	0,96584	0,08%
8	3D hex quad	0,04501	0,04%	0,48266	0,20%	0,96613	0,05%



Points used for comparison

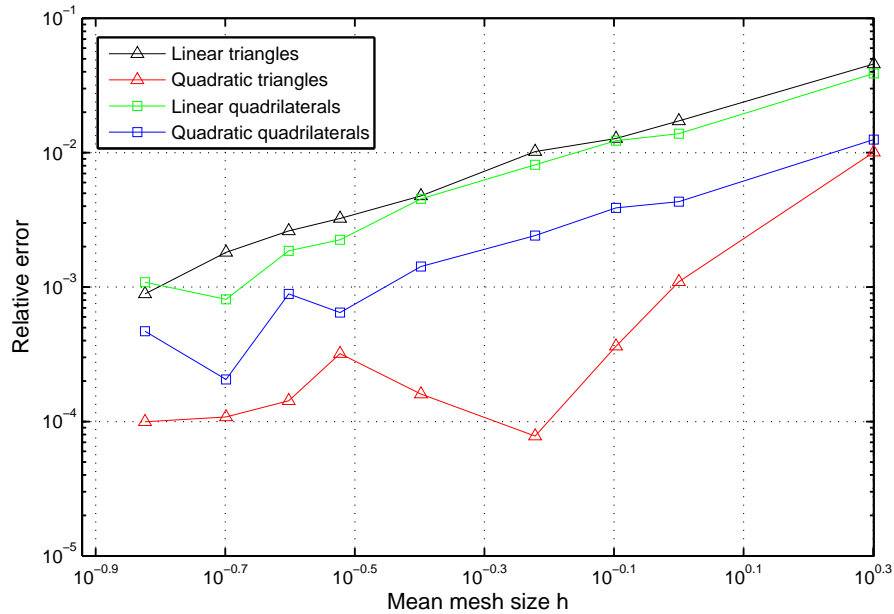
They were chosen in those positions to see the effect that the shape of the boundary produces on the result precision.

It can be observed that the results have bigger error when a point near the curved boundary is evaluated. Specially linear triangles and tetrahedral elements present important differences as the values of the temperature are further from the reference.

4.2 Study of the rate of convergence

In order to guarantee that the solutions provided are correct is important to check that the solution is converging. To do so, meshes were repeatedly refined and for each average size h the relative error was computed, using as reference the results of the finest mesh within the same type of elements.

The variable of interest is the temperature at point 3, and the study was only carried out for 2D elements, mainly because the 3D comparison would be incomplete as the external preprocessor used (GiD) had problems meshing with linear and quadratic hexahedral.



Mesh convergence study

Taking a look on the completeness of the shape functions for all the elements under study, the expected rate of convergence is $\mathcal{O}(h^2)$ and $\mathcal{O}(h^3)$ for linear and quadratic shape functions respectively. Also quadrilaterals are expected to have fairly better convergence due to the fact that their shape functions include some high order terms.

After carrying out the simulations, it can be observed in the picture above that while all types of element have a tendency to converge, the rate at which they do it is not the expected one (all of them present a slope of ~ 1.5). This fact could be explained by the existence of noise sources product of the curved geometry or the accuracy of the solver.

5 Challenges encountered

There were several challenges to be sorted. First, the understanding of the basic algorithm for a simple case of rectangular 2D geometry with linear rectangular elements.

Then, extending the program to work also in 3D and all of the element types described below. It is important to remark that the original files had errors in the functions *shapefunctions.m* and *shapefunctionsderiv.m*, which lead to incoherent results. Their identification and corrections took a considerable amount of time.

Also, a GiD problem type was developed. A new model with the same geometry was created and denser meshes were obtained in order to obtain the reference to calculate the relative error, for finally performing the mesh dependency study. The latest has not been carried out for hexahedral elements as GiD presented problems meshing it.

Finally, it was also coded a routine to compute the exact number of non-zero values that the stiffness matrix would have so that an exact allocation could take place.

6 Further work

Some of the improvements that can be implemented in the code are:

- Change the method for applying the Dirichlet BC. Using Lagrange multipliers method sets the Boundary conditions as additional equations making the system bigger than the original. Applying other methods as system reduction would probably add up to a better performance.
- Implement an iterative solver for large systems of equations, as the current way for solving the system of equations is the Matlab operator backslash which relies on direct methods. At some point during the coding it was tried to implement the conjugate gradient method, and although it gave acceptable results, its lack of performance in time and memory consumption made it be discarded.
- Adding computations in parallel. Taking advantage of the matlabpool commands some improvements in performance could be done through the use of parallel computations. Although this may be easier for computing the stiffness matrix as it can be split in concurrent operations, applying parallelization in the solving process is far from our capabilities at the time being.
- Extending the comparison of the element types to one considering the elapsed time.
- Incorporate a source term handling in the program.

7 Conclusions

In this work a simple FEM code for solving Laplace's problem with 2D quadrilateral elements was studied, understood and further extended to other types of elements. Also some performance improvements, such as the implementation of sparse stiffness matrices to reduce memory allocation, were carried out.

A GiD problemtype was developed to perform a convergence study with different types of elements and mesh densities. From its result, it became evident that the convergence rate was not the expected one, although all the elements do show convergence; which guarantees that the solution obtained after refining the mesh tends to the exact solution.

Finally, its important to note that the project was carried out using Git and Github, which proved to be an outstanding tool for version control and collaborative work.