# CE 133 / ME 180, Lab project #4

Ahmed Bakhaty, Miquel Crusells-Girona

In the next two labs, you will code a finite element program to solve the 2-D stationary heat equation in an easy domain. Your code will be based on the principle of modularity, so that you understand and code the FEM solver step by step. *You will have to provide your report at the end of these two weeks.* The process of coding an FEM solution can be difficult at first, so do not fall behind.

To begin with, recall that the stationary heat equation in 2-D is

$$\boldsymbol{\nabla} \cdot (k \boldsymbol{\nabla} T) + \rho r = 0 \tag{1}$$

where $\boldsymbol{\nabla}$ is the differential operator $\boldsymbol{\nabla} = (\partial_x, \partial_y)$, $T$ is the temperature, $k$ is the conductivity, $\rho$ is the density and $r$ is the heat source per unit mass. Assume, in your code, that $\rho$ is constant, and that $r$ may vary as $r = Ay + B$, where $A$ and $B$ are constants.

The boundary conditions we will consider for the heat equation will be either essential (temperature-based) or natural (flux-based)

$$\begin{aligned} T &= Cx + Dy + E \quad \text{on } \Gamma_T \\ -k\boldsymbol{\nabla} T &= \bar{\mathbf{q}} \quad \text{on } \Gamma_q \end{aligned} \tag{2}$$

where $C$, $D$, $E$ and $\bar{\mathbf{q}}$ are constant.

The process to code an FEM solution is based on the steps shown in Figure 1.

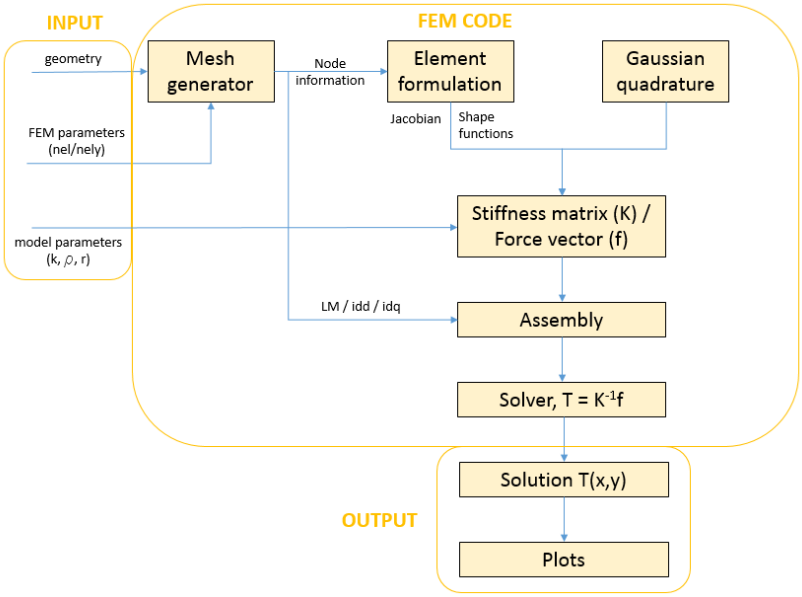


Figure 1: Conceptual diagram to code an FEM solution.

As a result, it is highly recommendable to code independent functions for each module. By doing this, your code becomes very versatile, as it allows you to modify any of the parts in an organised way.

Next sections will give you more information about each module, and what you should provide and expect from each function.

## 1 Weak form and expressions for K and f

First of all, you have to obtain the weak form for the 2-D stationary heat equation. *Provide the weak form in your report.*

Second, suppose an approximation of the solution of the form

$$T(x,y) = \sum_{i=1}^{N} T_i N_i(x,y)$$

and a Bubnov-Galerkin approach. Obtain expressions for the stifness coefficients $k_{ij}^e$ and $f_i^e$ of an element. *Provide your expressions in the report.*

## 2 Mesh generator

Mesh generation is one of the trickiest parts of FEM. When domains are easy, mesh generators are intuitive. However, when domains get complicated, mesh generators can definitely become an art.

In this lab, we are going to assume an easy domain, which will be defined by a paralellogram.

*Create a function called* mesh *that provides the coordinates of all the nodes, the LM array and the idd and idq arrays.* Note that this function will require the four corner nodes of the domain and the total number of elements per mesh edge, *nelx* and *nely*, which are both input parameters. As a result, your function should create a mesh of $nelx \times nely$ elements. Assume that you have the boundary conditions shown in Figure 3. To avoid applying two boundary conditions on the corners, just consider them as with $T$ prescribed.

As a clarifying example, the mesh with $nelx = 2$ and $nely = 2$, with given corner coordinates, should generate 4 elements and 9 nodes, as shown in Figure 2. The enumeration you follow for nodes and elements is not really important as long as you stick to it in all your code. The easiest thing to do here is to begin the enumeration at the bottom left node and go row by row (see Figure 2).

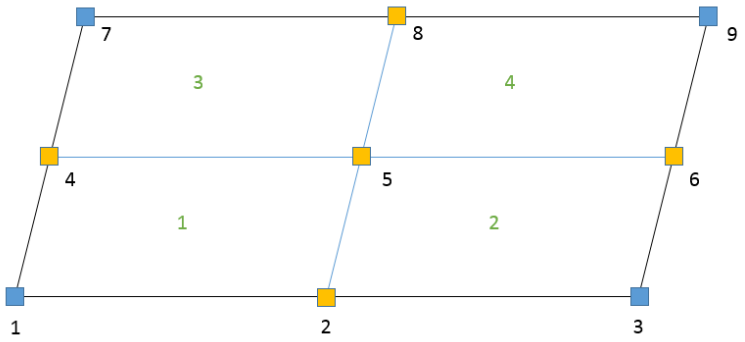


Figure 2: Mesh example for $nelx = 2$ and $nely = 2$.

The boundary conditions will be important to define the *idd* and *idq* arrays. The *idd* array will contain all the boundary nodes where the temperature is prescribed (essential boundary condition), whereas the *idq* array will contain all the nodes where the flux is prescribed (natural boundary condition). These matrices will be important for the assembly process. You can find more detailed information about these matrices here (Page 27).

The *LM* array is an array that contains the relationship between the (local) nodal numbers of the bilinear quad and the global nodal numbers of the mesh. As a result, the *LM* array is an $nen \times nel$ array, where *nen* is the number of nodes per element (4 in our case).

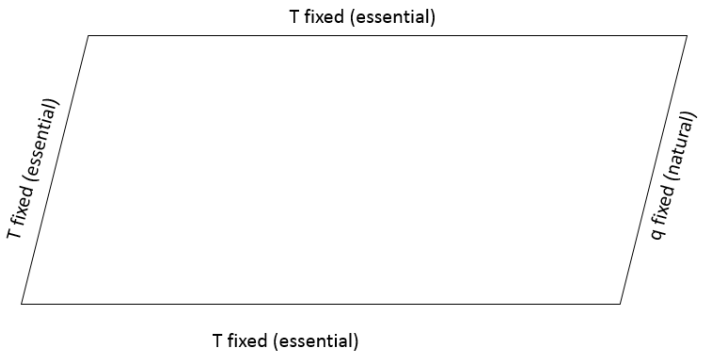


Figure 3: Boundary conditions for the problem.

## 3 Element formulation

In order to build up our FEM solution, we are going to use 2-D bilinear isoparametric elements. *Provide the isoparametric shape functions of this element.*

*Create a Matlab Function called* shapef4nq *that provides the isoparametric shape functions and the Jacobian of the isoparametric mapping.* Note that this function requires the nodal coordinates of the element for the Jacobian matrix $J$. You can check your answer by using the definition of the shape functions.

Even though this is not common in real FEM codes, code your functions as symbolic (*syms x y*), as it will be more insightful for you. In reality, FEM solvers only store values at Gauss points. At this point, you can choose whether you want this function to export also the derivatives with respect to the physical domain. These can be easily obtained in Matlab by using the function $diff$.

## 4 Gaussian quadrature

The way in which integrals are computed on the computer is via Gaussian quadratures

$$\int_{y_1}^{y_2}\int_{x_1}^{x_2} f(x,y)\,dxdy \approx \sum_{i=1}^{nip} f(x_i,y_i)w_i$$

where $nip$ is the number of integration points.

The simplest integration scheme to integrate cubic functions exactly in one dimension is the Gaussian quadrature with two points, weights $w_1 = w_2 = 1$ at $\xi_1 = -\frac{1}{\sqrt{3}}$ and $\xi_2 = \frac{1}{\sqrt{3}}$. In two dimensions, even though it's not optimal, one just uses the same nodes and weights for each dimension (Figure 4), i.e. $(\xi_1, \eta_1) = (-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}})$, etc.

Recalling that

$$\int_{y_1}^{y_2}\int_{x_1}^{x_2} f(x,y)\,dxdy = \int_{\eta_1}^{\eta_2}\int_{\xi_1}^{\xi_2} f(\xi,\eta)\ \det(J)d\xi d\eta$$

*create a Matlab function called* gausseval *that provides the value of an integral by using the symbolic integrand and the determinant of the Jacobian.* Note that, because your shape functions are symbolic, you will have to use the Matlab function *subs*.

Because the 2nd order Gaussian quadrature integrates cubic functions exactly, you can test your function by providing the input arguments (with linear to cubic integrand) and comparing the result against the exact integral.

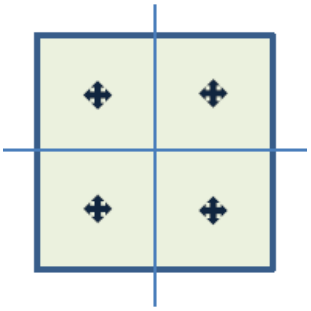

Figure 4: Gauss points in a 2-D isoparametric quad.

## 5 Stiffness and force coefficients

Next, *construct a function called* kijfi *that provides the stiffness coefficients* $k^e_{ij}$ *of element* $e$ *and the force vector at that element.* Note that, in order to code this function, you will need information from the element (derivatives of shape functions with respect to the parent domain, determinant of the Jacobian matrix), together with model parameters ($k$, $\rho$, $r$). This function will require some of the functions created before.

## 6 Assembly

Once one is able to generate the stiffness matrix and force vector of an element, the assembly process needs to be performed. The assembly module will generate the global stiffness matrix of the mesh and its force vector, taking into account the boundary conditions.

To this purpose, *write a function called* assem *that provides the global stiffness matrix and force vector for the problem.* Note that this function will require the *LM* and *idd* arrays, and will loop over all the elements in the mesh.

Regarding the boundary conditions, recall that one can rearrange the vector $\mathbf{T}$ of degrees of freedom in the mesh, according to *idd* as (see more details here, page 27)

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_u \\ \mathbf{T}_k \end{bmatrix}$$

where $\mathbf{T}_u$ contains the unknown temperature degrees of freedom (i.e. not contained in *idd*) and $\mathbf{T}_k$ contains the known temperature degrees of freedom (i.e. where the temperature is known, hence for all the nodes contained in *idd*). As a result:

$$\begin{bmatrix} \mathbf{K}_{uu} & \mathbf{K}_{uk} \\ \mathbf{K}_{ku} & \mathbf{K}_{kk} \end{bmatrix} \begin{bmatrix} \mathbf{T}_u \\ \mathbf{T}_k \end{bmatrix} = \begin{bmatrix} \mathbf{F}_u \\ \mathbf{F}_k \end{bmatrix}$$

Note that, in this description, the coefficients of $\mathbf{F}_k$ are unknown, as they correspond to nodes in which the temperature is prescribed (one does not know the flux at these nodes).

The function assem should, as a result, export the global stiffness matrix, $\mathbf{K}$, in the order defined previously and the force vector $\mathbf{F}_u$, for the nodes where the temperature degree of freedom has to be obtained.

## 7 Solver

Finally, write a script (not necessarily a function) that has the parameters of the model (Figure 1) as inputs, and provides the temperature degrees of freedom at the nodes. This script will essentially be the FEM solver and, in general terms, will contain the sequence:

1. Create the mesh.
2. Formulate the element.
3. Obtain the stiffness matrix and force vector of each element.
4. Assemble the global stiffness matrix according to the boundary conditions.
5. Solve for the unknown degrees of freedom.
6. Solve for the fluxes where the temperature is imposed.

Note that, because $\mathbf{T}_k$ are actually known in the former equation:

$$\mathbf{K}_{uu}\mathbf{T}_u + \mathbf{K}_{uk}\mathbf{T}_k = \mathbf{F}_u \quad \Rightarrow \quad \mathbf{T}_u = \mathbf{K}_{uu}^{-1}(\mathbf{F}_u - \mathbf{K}_{uk}\mathbf{T}_k)$$

This process is known as *static condensation*.

One can obtain the fluxes where $T$ is imposed, element by element, by the well-known relationship

$$q = -k\boldsymbol{\nabla}T = -k\sum_{i=1}^{4} T_i \boldsymbol{\nabla}N_i(\xi,\eta)$$

Note that, because a Gaussian quadrature is used, the most accurate results will be obtained by evaluating this expression at the Gauss points.

Also, observe that one can obtain the force vector at the nodes where $T$ is prescribed by using the second equation in the former matrix expression:

$$\mathbf{F}_k = \mathbf{K}_{ku}\mathbf{T}_u + \mathbf{K}_{kk}\mathbf{T}_k = \mathbf{F}_k$$

Note, however, that this force vector *includes the contribution from the heat supply at the nodes where $T$ is prescribed*.

## 8 Other functions

Because you want your FEM solver to be modular, consider also to code any other auxiliary function that you may need such that it can be used in any module. A good example of these auxiliary functions would be one to provide the coordinates of a particular element $e$.

## 9 Plots

You may want to plot your solution. In order to do that:

1. Make sure you obtain the temperature degree of freedom for all the nodes in your mesh, including the boundary.
2. Use the Matlab function $surf(x, y, T)$ to plot the temperature as a 3d surface.
3. Use the Matlab function $contourf(x, y, T)$ to plot a contour of temperatures.

An example on how to use these functions is included in the following code. This code generates two plots: one surface plot and one contour plot.

```
x = 1:0.2:2;
y = 0:0.1:3;
[X,Y] = meshgrid(x,y);
Z = X.*exp(-X.^2-Y.^2);
subplot(2,1,1)
surf(X,Y,Z)
subplot(2,1,2)
contourf(X,Y,Z)
```

If the domain is not regular, plotting the solution requires some extra steps, but don't worry about that!