# Sheet 10 – FEM in 2-D

December 12, 2016

The aim of this sheet is to write some auxiliary functions that can assist with automatically generating a stiffness matrix. The problem that we will consider is

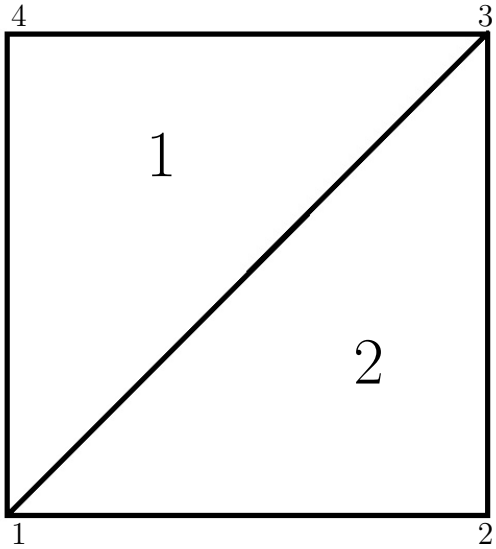
$$u_{xx} + u_{yy} = f(x, y)$$

in some domain $D$ along with (arbitrary) boundary conditions. In this case, the stiffness matrix $A$ has $(i, j)$ entry given by

$$\iint_D \nabla\phi_i \cdot \nabla\phi_j \mathrm{d}x\mathrm{d}y$$

where $\phi_i$ and $\phi_j$ are the shape functions that take 1 at the $i$-th and $j$-th nodes respectively.

We will use triangular elements and piecewise-linear shape functions. As a test case, we will consider the triangulation of the unit square $(x, y) \in [0, 1] \times [0, 1]$ shown below, although the code that we write will be adaptable to different domains and numbers of elements.



**Exercise 1.** *Calculating the Coefficients*
Our shape functions $\phi$ can be represented by their coefficients. On each element $k$, we can define the function associated with node $i$ by

$$\phi_i(x, y) = a_{i,k}x + b_{i,k}y + c_{i,k}.$$

So our first task is to write a function that calculates these coefficients, and stores them in a sensible way. There are many ways to do this. You may follow your own approach, or be guided along the following path.

Finite element codes usually involve what is known as a "'connectivity matrix"' $K$ that identifies which nodes belong to which elements. For example, the $(i, j)$-th entry of $K$ could be 1 if node $i$ is part of element $j$, and 0 otherwise. We will provide our function with this connectivity matrix, and two other matrices $R$ and $L$ that together define the grid.

For a grid with $N$ nodes and $M$ elements, we will create $R$, an $N \times 2$ matrix that lists the co-ordinate points of each node, and $L$, an $M \times 6$ matrix that lists the co-ordinate points of each element.

In the example grid, we have

$$R = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$L = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

where we can see that the rows of $R$ correspond to the different nodes, and the rows of $L$ correspond to different elements.

1. Write out the connectivity matrix K

   Given these inputs, the function should loop over the nodes to define the function $\phi$ associated with that node. We will store the coefficients in the output, an $N \times 3M$ matrix $T$ such that each row of $T$ corresponds to a different function. Within each row, we will have $M$ triples $(a, b, c)$ to define the coefficients for each element. Thus, for each node $i$ we will loop over all of the elements $j$ and calculate the coefficients on each one. Note that we only need to calculate the coefficients if the connectivity matrix tells us that the node $i$ lies on the element $j$.

2. Write out the general system that needs to be solved to compute coefficients $(a, b, c)$ for $\phi_i$ on element $j$.

3. Now write a code that creates the matrix $T$ as described above.

For the given grid, the output should be

$$T = \begin{bmatrix} 0 & -1 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

**Exercise 2.** *The Stiffness Matrix*
We will now write a script that calls our coefficient function to compute the stiffness matrix.

As stated, in this simple case the integrands are piece-wise constant and so we have

$$A_{ij} = \Sigma_k |\Delta_k| (a_{i,k} a_{j,k} + b_{i,k} b_{j,k})$$

where $|\Delta_k|$ is the area of element $k$, and the constants are those referred to in the piecewise definition of $\phi$ above.

Now we need three nested for-loops, the usual two to go over all the entries of the matrix and then a third for loop to add up the contributions from all of the elements. However, we only need to do anything on this final loop if the connectivity matrix tells us to – *i.e.* if both nodes $i$ and $j$ are placed on element $k$.

Again, for this example grid the output should be

$$A = \begin{bmatrix} 1 & -0.5 & 0 & -0.5 \\ -0.5 & 1 & -0.5 & 0 \\ 0 & -0.5 & 1 & -0.5 \\ -0.5 & 0 & -0.5 & 1. \end{bmatrix}$$

**Exercise 3.** *Extension*

1. Use your code to solve the given problem with homogeneous Neumann boundary conditions and forcing function $f(x, y) = \cos(x)$.

2. Observe convergence as you increase the number of elements.

3. Now try the same problem but with homogeneous Dirichlet boundary conditions. You will need to over-write some of the coefficients in order to preserve the boundary values.