CMIS Hand-in 4: Finite Element Method

Nikolai Plambech Nielsen
lpk331@alumni.ku.dk
Niels Bohr Institute, University of Copenhagen

1 THE FINITE ELEMENT METHOD

The essence of the finite element method is to represent the domain as a finite set of elements (hence the name), and compute an approximate solution for each of these elements, adding up to the global solution. The actual elements vary on the dimensionality of the system. Straight-line segments for 1D problems, triangles for 2D and tetrahedra for 3D are the ones we focus on here due to their simplicity, but others can be used as well.

The actual method can be summarised in a series of steps:

(1) Convert the differential equation to a volume integral
(2) Define the elements
(3) Choose a shape and trial function for the elements
(4) Compute the elementwise integrals
(5) Assemble the global matrix and source vector
(6) Apply boundary conditions
(7) Compute the solution

In this hand-in we focus on implementing the method in 1D and 2D, and use it to solve the Poisson equation for a simple source term:

$$\nabla^2 u = c \tag{1}$$

where to begin with we set $c = 0$, and later $c \in \mathbb{R}$, subject to point-wise boundary conditions $u(\mathbf{r}) = a(\mathbf{r}), \forall \mathbf{r} \in \Gamma$.

Step 1. In the first step of the recipe we multiply each side by a function $v(\mathbf{r})$, which must be continuously differentiable and 0 on the boundary of the domain ($v(\mathbf{r}) = 0,\ \forall\ \mathbf{r} \in \Gamma$). Then we integrate over the domain on both sides to get

$$\int_\Omega v(\mathbf{r})\nabla^2 u(\mathbf{r})\ \mathrm{d}\Omega = \int_\Omega v(\mathbf{r})c\ \mathrm{d}x \tag{2}$$

This formulation of the problem is still equivalent to the original form, and is called "Strong Form". Next we perform integration by parts on the left hand side:

$$\int_\Omega v(\mathbf{r})\nabla^2 u(\mathbf{r})\ \mathrm{d}\Omega = \int_\Gamma v(\mathbf{r})\nabla u(\mathbf{r})\ \mathrm{d}\Gamma - \int_\Omega \nabla v(\mathbf{r})\nabla u(\mathbf{r})\ \mathrm{d}\Omega \tag{3}$$

But the first term is zero by definition, leading to the "Weak Form" of the problem:

$$\int_\Omega \nabla v(\mathbf{r})\nabla u(\mathbf{r})\ \mathrm{d}\Omega = -\int_\Omega v(\mathbf{r})c\ \mathrm{d}\Omega \tag{4}$$

This takes care of the first step.

Step 2. In the second step we use the methods from last weeks hand-in on the generation of computational meshes. In particular I define the geometry of the setup in a `.poly` file and relegate the mesh generation to the `Triangle` program written by J. R. Shewchuk.

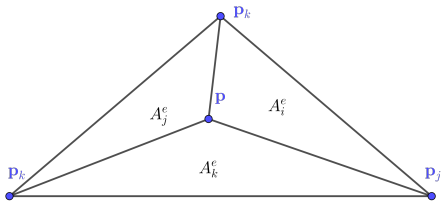

Figure 1: Barycentric coordinates for a triangular element.

Step 3. For the third step we approximate the unknown u by the function

$$u \approx \sum_{e\in\Omega} \mathbf{N}^e \hat{u}^e, \tag{5}$$

where the sum runs over all elements in the system, $\hat{u}^e$ is a vector of the value of u on the 3 vertices in the e'th element (with vertices counted counter clockwise): $\hat{u}^e = [u_i^e\ u_j^e\ u_k^e]^T$, and $\mathbf{N}^e$ is the "shape function" for the element. The shape function must have the following properties

$$N_i(\mathbf{r}) = \begin{cases} 1 & \mathbf{r} = \mathbf{r}_i, \\ 0, & \mathbf{r}_j \wedge j \neq i \end{cases}, \quad \sum_i N_i(\mathbf{r}) = 1. \tag{6}$$

The first ensures proper interpolation for the points, whilst the second ensures that the approximation of $\mathbf{u}$ is exact on all vertices in the system. In one dimension we use a hat-function:

$$N_i(x) = \max\left(0, 1 - \left|\frac{x_i - x}{\Delta x}\right|\right) \tag{7}$$

which are 1 on the i'th vertex, and 0 on all other (assuming equal spacing of nodes in the domain Δx). And for triangles in two dimensions we use barycentric coordinates:

$$N_i^e(\mathbf{r}) = \frac{A_i^e}{A^e} \tag{8}$$

where the geometry is as seen in figure 1. Where the areas can be found as the magnitudes of cross products:

$$N_i^e = \frac{\left|(\mathbf{p}_k - \mathbf{p}_j)\times(\mathbf{p} - \mathbf{p}_j)\right|}{2A^e}, \quad A^e = \frac{1}{2}\left|(\mathbf{p}_j - \mathbf{p}_i)\times(\mathbf{p}_k - \mathbf{p}_i)\right| \tag{9}$$

Now, the points of the triangles are two dimensional, so the cross products are also just the determinant of the 2×2 matrix $[(\mathbf{p}_k - \mathbf{p}_j),\ (\mathbf{p} - \mathbf{p}_j)$ and equivalently for the total area, and other barycentric coordinates.

We also need the spatial derivatives of the shape functions, which in one dimension, for the straight line between nodes i and j gives

$$\frac{\partial N_i^e}{\partial x} = \frac{-1}{\Delta x}, \quad \frac{\partial N_j^e}{\partial x} = \frac{1}{\Delta x} \tag{10}$$

And in two dimensions

$$\frac{\partial N_i^e}{\partial x} = -\frac{p_{k,y} - p_{j,y}}{2A^e}, \quad \frac{\partial N_i^e}{\partial y} = \frac{p_{k,x} - p_{j,x}}{2A^e} \tag{11}$$

All of which are independent of the free variable $\mathbf{r}$.

Next we choose the trial function to be $v(\mathbf{r}) = \mathbf{N}(\mathbf{r})\delta\mathbf{u}$, where $\delta\mathbf{u}$ is a vector of arbitrary values (a choice reminiscent of the techniques used in the principle of least action). Substituting the shape and trial functions into the volume integral gives

$$\int_\Omega \nabla(\mathbf{N}(\mathbf{r})\delta\mathbf{u})\nabla\mathbf{N}(\mathbf{r})\hat{u}\ \mathrm{d}\Omega = -\int_\Omega (\mathbf{N}(\mathbf{r})\delta\mathbf{u})c\ \mathrm{d}\Omega \tag{12}$$

or utilizing $v(\mathbf{r})^T = v(\mathbf{r})$ since v is a scalar and contracting the integral

$$\delta\mathbf{u}^T \int_\Omega \nabla\mathbf{N}^T(\mathbf{r})\nabla\mathbf{N}(\mathbf{r})\hat{u} + \mathbf{N}^T(\mathbf{r})c\ \mathrm{d}\Omega = 0 \tag{13}$$

Since this must hold for all choices of $\delta\mathbf{u}$, the integral itself must be zero. We can then manipulate the integral further to get a linear set of equations for $\hat{u}$:

$$\left[\int_\Omega \nabla\mathbf{N}^T(\mathbf{r})\nabla\mathbf{N}(\mathbf{r})\ \mathrm{d}\Omega\right]\hat{u} = -c\int_\Omega \mathbf{N}^T(\mathbf{r})\ \mathrm{d}\Omega, \quad \mathbf{K}\hat{u} = \mathbf{f} \tag{14}$$

where $\mathbf{K}$ is the left hand side integral, and $\mathbf{f}$ is the right hand side integral. Next we split up the integral in the individual elements:

$$\mathbf{K} = \int_\Omega \nabla\mathbf{N}^T(\mathbf{r})\nabla\mathbf{N}(\mathbf{r})\ \mathrm{d}\Omega = \sum_{e\in\Omega}\int_{\Omega^e} \nabla(\mathbf{N}^e)^T\nabla\mathbf{N}^e\ \mathrm{d}\Omega = \sum_{e\in\Omega}\mathbf{K}^e. \tag{15}$$

and likewise for the source vector. Really, the individual element matrices are $N\times N$, where N is the number of vertices in the domain, but only 9/4 elements are non-zero, so we just calculate a 3×3 (2×2) matrix, and add the values in the right place of $\mathbf{K}$.

Step 4. In both one and two dimensions all integrands are independent of the integration variables, and we just get an extra factor of A^e (which is Δx for one dimension), along with an outer product between a vector and itself:

$$\int_{\Omega^e} \frac{\partial(\mathbf{N}^e)^T}{\partial x}\frac{\partial\mathbf{N}^e}{\partial x}\hat{u}^e\ \mathrm{d}\Omega = A^e\frac{\partial(\mathbf{N}^e)^T}{\partial x}\frac{\partial\mathbf{N}^e}{\partial x}\hat{u}^e \tag{16}$$

The source term is

$$\mathbf{f}^e = -\int_{\Omega^e}(\mathbf{N}^e)^T c\ \mathrm{d}\Omega \tag{17}$$

If $c = 0$ then we trivially have $\mathbf{f}^e = 0$. if instead, $c \neq 0$ we get (in two dimensions):

$$f_i^e = -\frac{cA^e}{3} \tag{18}$$

which can be seen by integrating any of the barycentric coordinates over a triangle with points $\mathbf{p}_i = (0,0), \mathbf{p}_j = (\alpha, 0), \mathbf{p}_k = (\beta,\gamma)$. Any triangle can be rotated and translated to have these coordinates if we let $0 < \alpha \leq \beta$, meaning the result is general.

In one dimension we get from straight integration.

$$f_i^e = \Delta x/2 \tag{19}$$

Step 5. Next we need to put all the different element matrices into the global matrix. Here we can leverage the fact since the element matrices consists of an outer product between one vector and itself, the resulting matrices are symmetric. This means we do not have to worry too much about the index order when assembling the system. For each element we take the global index of the vertices in the element, create a meshgrid of these values, and add the 9/4 (2D/1D) values in the element matrix to the corresponding values in the global matrix. Note the adding part: since each vertex may be represented in a multitude of elements, we need to incorporate the contribution from all elements.

The source vector is built similarly, except for each element there are only 3/2 elements to add.

Step 6. Since we are dealing with point-wise / Dirichlet bondary conditions, incorporating these amounts to just clamping the values on the vertices to be the boundary condition value. This means that we just set the row of the n'th element to 0, except for the n'th entry, which we set to a one. We also set the n'th entry in the source term equal to the value of the boundary condition, leading to the equation:

$$\hat{u}_n = a(\mathbf{p}_n) \tag{20}$$

Step 7. Last we just need to solve the linear system of equations. If the system is not uniquely defined (ie, the matrix rank is non-singular) without the boundary conditions, then hopefully by adding these we get a full-rank matrix, and we can employ a direct solver. If not we resort to an iterative method. In our case (both one and two dimensions), our system is uniquely defined and we do indeed get a full-rank, non-singular matrix.

2 EXPERIMENTS

For this weeks experiments we solve the system on a rectangular domain with $x \in [0,6]$ and $y \in [0,2]$, subject to the boundary condition $u(0,y) = a,\ u(6,y) = b$.

To evaluate the accuracy of the method, we compare the numerical solution $\hat{u}_i$ to the analytical solution $u(\mathbf{p}_i)$ using the root mean square:

$$\mathrm{res} = \sqrt{\sum_i (\hat{u}_i - u(\mathbf{p}_i))^2/N_{\mathrm{dof}}} \tag{21}$$

where N_{dof} is the number of degrees of freedom for the system - ie the total number of nodes in the domain, minus the number of nodes subject to boundary conditions. (this ensures we get a nonsensical result in degenerate cases, like ones where all nodes are boundary nodes).

2.1 The analytical solution

In the case of $c \in \mathbb{R}$ we can solve the system analytically. With these boundary conditions we exclude all dependence on y, giving an analytical solution (found by integrating a constant twice, and solving for two known points to get the factors of the linear and constant term):

$$u(x,y) = \frac{c}{2}x^2 - \frac{18c + a - b}{6}x + a \tag{22}$$

which also includes the straight line solution of $c = 0$:

$$u(x,y) = \frac{b-a}{6}x + a = \frac{\Delta y}{\Delta x}x + a \tag{23}$$

2.2 One dimension

Experiment. In one dimension the $\mathbf{K}$ matrix only has elements along the main and two off diagonals, since each vertex is only part of two neighbouring elements. Because of this, the matrix is equal to what one would get if one uses the finite difference method, without ghost nodes, and using dirichlet boundary conditions. Letting $c = 0$

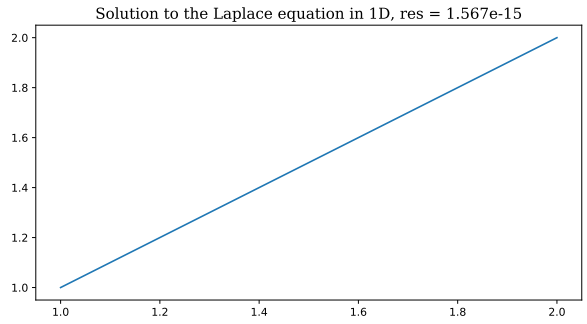

Figure 2: Solution in 1, with $c = 0$. Note the low residual.

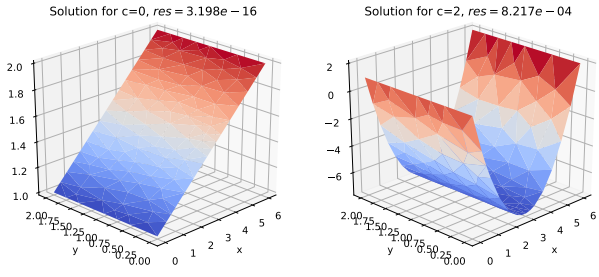

Figure 3: Solutions in 2D, with $c = 0$ (left) and $c = 2$ (right). Note again the low residual for the plane. Triangle settings: minimum angle (30 degrees), maximum area (0.1 pixels squared)

and solving for a computational mesh of 11 equidistant points between 1 and 2 gives the result seen in figure 2. The solution is just $u = x$, and we see that the residual is but an order of magnitude above the machine epsilon, even with just 11 points.

I suspect this is not because of some insane precision in the finite element method, but rather due to the fact, that the shape functions are essentially a piecewise linear interpolation between points, so when the solution *is* a linear function, it will be as close to exact as possible.

Indeed we will see this as we go to the two dimensional case, and set $c \neq 0$.

2.3 Two Dimensions

Experiment. When going to two dimensions we see the same tendency with $c = 0$ (see figure 3, left). The result is more interesting when setting $c \neq 0$ (figure 3, right), where the residual of an appreciable magnitude.

Experiment. Next we analyse the residual dependency on the max triangle area (and therefore the number of vertices in the mesh), by solving the system for a range of maximum triangle areas (between 0.05 and 2 pixels squared). The result is seen in figure 4. Here we see the expected drop in magnitude of the residual as maximum triangle size is decreased (and the number of vertices is increased).

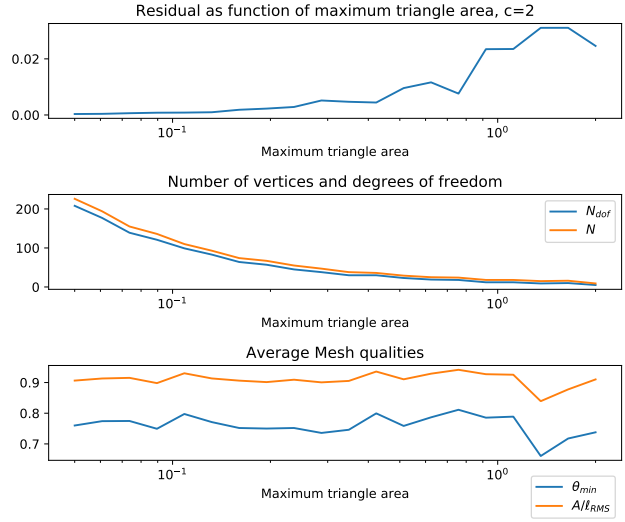

Figure 4: Residual, number of vertices, number of degrees of freedom, and the average of two quality measures (minimum triangle area and aspect ratio), all as a function of maximum triangle area. Note the log scale on the first axis. No meshes had any triangles with any quality measure equal to 0. Minimum triangle area set to 30 degrees

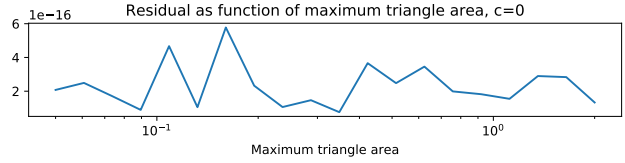

Figure 5: Residual,as a function of maximum triangle area. All other settings are equal to that from figure 4

We also see no big dependence on maximum triangle area, for the averages of the quality measures - `Triangle` produces quality meshes with the input settings. No meshes had any triangle with 0 quality measure.

Experiment. To make sure that the low residual for the plane is not a fluke, we perform the same analysis as before for $c = 0$, the result (only the residual - the number of vertices and the quality does not change with c) is shown in figure 5. Again we see a residual fluctuating around the machine epsilon. Like the shape function in 1D, the barycentric coordinates essentially performs a linear interpolation between points, so I expect this is the reason why we see such a low residual.

Experiment. Lastly we measure the quality of the finite element method as compared to the finite difference method. We create a grid with approximately the same number of points as that for the finite element method. We calculate the number of points in each

direction as follows:

$$\Delta x = \ell_x / N_x, \quad \Delta y = \ell_y / N_y, \quad N = N_x N_y \tag{24}$$

where ℓ_x is the size of the domain in the x-direction and N_x is the number of points in the x-direction (and likewise for y). Setting $\Delta x = \Delta y$ gives

$$N_x = \sqrt{3N}, \quad N_y = \sqrt{\frac{N}{3}} \tag{25}$$

But these are not necessarily integers, so instead we round them to the nearest integer.

$$N_x = \left\lfloor \sqrt{3N} \right\rceil, \quad N_y = \left\lfloor \sqrt{\frac{N}{3}} \right\rceil \tag{26}$$

Of course, since we are using the finite difference method, and are dealing with second order differences, we cannot have $N_y < 3$, so we restrict ourselves to that. We are not given boundary conditions for the top and bottom of the domain, as they are not needed to uniquely determine the solution. However, we need to modify the finite difference formulas for the nodes on the top and bottom boundary so we do not access any non-existing nodes. To do this we employ a forward difference at the bottom, and a backwards difference at the top (as given in https://en.wikipedia.org/wiki/Finite_difference#Higher-order_differences). First we plot the result for $N = 200$ in figure 6, and as expected we get a parabola, but with miniscule residuals. Calculating these for all values of N as in 4 we get the result in figure 7. With the Finite Difference Method we see incredibly small residuals, even for relatively low N (24, 18 degrees of freedom). It might be we need to go nearer the degenerate case for higher residuals to appear, but it does seem suspicious.

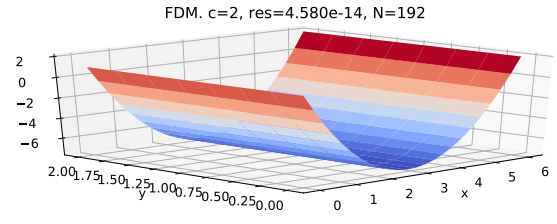

Figure 6: Solution obtained using Finite Difference Method, for $c = 2$. Note the low, low residual.

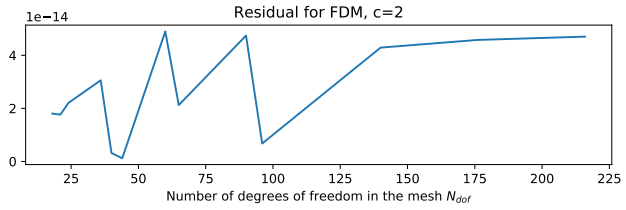

Figure 7: Residuals for solutions found with FDM, for $c = 2$, as a function of N