1 Introduction

The finite element method generalizes to two dimensional and three dimensional spaces without a glitch! In fact, it is in two, three, and sometimes higher dimensions that the finite element method provides its key advantage: ability to represent irregular geometries.

2 2D Finite Element Method

2.1 Geometry representation

In one dimension, the geometry (the segment [0, 1]) was represented by a collection of small segments. In two dimensions, the simplest variety of the finite element method represents the geometry by adjacent triangles. For example, here is how a disk would be represented:

Creating a mesh for a given geometry – a process called tessellation or, in this special case, triangulation – is not an easy task. In real life problems, the
tessellation step usually takes a long as the rest of the solution. In three dimensions, tessellation is even more challenging. The most common triangulation algorithm is Delaunay triangulation. (In this class, the mesh will always be provided.)

2.2 Formulation of the method

The Laplace equation

\[ \Delta u = 0 \]

subject to the boundary conditions

\[ u|_S = U_0 \]

is solved by minimizing the objective function

\[ E = \frac{1}{2} \int_\Omega |\nabla u|^2 d\Omega \]

subject to the same boundary condition (constraints!).

The finite element method minimizes \( E \) with respect to continuous function that a linear over each triangle (in all – piecewise linear). Such a function is shown here:

2.3 The basis

We will once again express such functions as linear combinations of elements of an atomic basis. An atomic basis consists of functions \( \psi_s \) which are piecewise linear, equal one at a single node and zero everywhere else. A few members of this basis are shown below.
Number the nodes of the mesh arbitrarily 1 through $N$ and number the elements of the basis so that $\psi_s(x,y)$ equals 1 at the node $s$ and 0 everywhere else. Represent the function $u(x,y)$ with respect to this basis

$$u(x,y) = \sum_{r=1}^{N} c_r \psi_r(x,y)$$  \hspace{2cm} (1)

2.4 Forming the FEM matrix

Rewrite the energy $E$ in cartesian coordinates

$$E = \frac{1}{2} \int_{\Omega} \left( \frac{\partial u}{\partial x} \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} \frac{\partial u}{\partial y} \right) d\Omega$$

and substitute equation (1)

$$E = \frac{1}{2} \int_{\Omega} \left( \sum_{r=1}^{N} c_r \frac{\partial \psi_r}{\partial x} \sum_{s=1}^{N} c_s \frac{\partial \psi_s}{\partial x} + \sum_{r=1}^{N} c_r \frac{\partial \psi_r}{\partial y} \sum_{s=1}^{N} c_s \frac{\partial \psi_s}{\partial y} \right) d\Omega$$

$$= \frac{1}{2} \sum_{r,s=1}^{N} c_r c_s \int_{\Omega} \left( \frac{\partial \psi_r}{\partial x} \frac{\partial \psi_s}{\partial x} + \frac{\partial \psi_r}{\partial y} \frac{\partial \psi_s}{\partial y} \right) d\Omega$$

As in the one dimensional case, we end up with a quadratic form

$$E = \frac{1}{2} \sum_{r,s=1}^{N} M_{rs} c_r c_s d\Omega,$$
where
\[ M_{rs} = \int_\Omega \left( \frac{\partial \psi_r}{\partial x} \frac{\partial \psi_s}{\partial x} + \frac{\partial \psi_r}{\partial y} \frac{\partial \psi_s}{\partial y} \right) d\Omega \] (2)

The two remaining tasks is to compute the matrix \( M_{rs} \) and to impose the boundary conditions either by direct substitution or by Lagrange multipliers.

### 2.5 Computing the FEM matrix

We are discussing the simplest version of the finite element method: a triangulated domain with piecewise linear element functions. Therefore, the derivatives that appear in equation (2) are piecewise constant – and zero over the majority of the domain.

As in the one dimensional case we will build the "matrix finite element by finite element" (rather than "matrix entry by matrix entry"). If the finite element, a triangle in our case, includes nodes 3, 7, and 9 then that element will contribute to entries \( M_{33}, M_{77}, M_{99}, M_{37}, M_{73}, M_{97}, M_{39}, \) and \( M_{93} \). In order to compute the contributions to these matrix entries, we must evaluate \( \frac{\partial \psi_r}{\partial x} \) and \( \frac{\partial \psi_s}{\partial x} \), \( \frac{\partial \psi_r}{\partial y} \) and \( \frac{\partial \psi_s}{\partial y} \). Let us say this in a more generic way: we need the partial derivatives of the linear function (plane) that is 1 and node 3 and 0 at nodes 7 and 9, and the other two planes constructed similarly.

This partial derivatives are very easy to calculate. Let the coordinates of points 3, 7, and 9 be \((x_3, y_3), (x_7, y_7), \) and \((x_9, y_9)\) and let the three planes be

\[
\begin{align*}
\psi_3 (x, y) & = a_3 x + b_3 y + c_3 \\
\psi_7 (x, y) & = a_7 x + b_7 y + c_7 \\
\psi_9 (x, y) & = a_9 x + b_9 y + c_9
\end{align*}
\]

The unknown coefficients \( a_s, b_s, \) and \( c_s \) can be found from the following systems of equations

\[
\begin{bmatrix}
  x_3 & y_3 & 1 \\
  x_7 & y_7 & 1 \\
  x_9 & y_9 & 1
\end{bmatrix}
\begin{bmatrix}
  a_3 \\
  b_3 \\
  c_3
\end{bmatrix}
= 
\begin{bmatrix}
  1 \\
  0 \\
  0
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x_3 & y_3 & 1 \\
  x_7 & y_7 & 1 \\
  x_9 & y_9 & 1
\end{bmatrix}
\begin{bmatrix}
  a_7 \\
  b_7 \\
  c_7
\end{bmatrix}
= 
\begin{bmatrix}
  0 \\
  1 \\
  0
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x_3 & y_3 & 1 \\
  x_7 & y_7 & 1 \\
  x_9 & y_9 & 1
\end{bmatrix}
\begin{bmatrix}
  a_9 \\
  b_9 \\
  c_9
\end{bmatrix}
= 
\begin{bmatrix}
  0 \\
  0 \\
  1
\end{bmatrix}
\]

Let \( a \) be the determinant of the recurring matrix

\[
A = \begin{bmatrix}
  x_3 & y_3 & 1 \\
  x_7 & y_7 & 1 \\
  x_9 & y_9 & 1
\end{bmatrix}
\]
Importantly, \( A \) is twice the area of the triangle! (For the record, \( A = x_3y_7 - x_3y_9 - x_7y_3 + x_7y_9 + x_9y_3 - x_9y_7 \).

The the unknown coefficients can be obtained by solving the above systems

\[
\begin{bmatrix}
a_3 \\
b_3 \\
c_3 \\
a_7 \\
b_7 \\
c_7 \\
a_9 \\
b_9 \\
c_9
\end{bmatrix} = \begin{bmatrix} x_3 & y_3 & 1 \\
x_7 & y_7 & 1 \\
x_9 & y_9 & 1 \\
x_3 & y_3 & 1 \\
x_7 & y_7 & 1 \\
x_9 & y_9 & 1 \\
x_3 & y_3 & 1 \\
x_7 & y_7 & 1 \\
x_9 & y_9 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\
0 \\
0 \\
0 \\
1 \\
1 \\
0 \\
1 \\
1 \end{bmatrix} = \frac{1}{A} \begin{bmatrix} y_7 - y_9 \\
-x_7 + x_9 \\
x_7y_9 - x_9y_7 \\
-y_3 + y_9 \\
x_3 - x_9 \\
x_3y_9 + x_9y_3 \\
y_3 - y_7 \\
-x_3 + x_7 \\
x_3y_7 - x_7y_9 \end{bmatrix}
\]

Therefore,

\[
\frac{\partial \psi_3}{\partial x} = \frac{1}{A} (y_7 - y_9); \quad \frac{\partial \psi_3}{\partial y} = \frac{1}{A} (-x_7 + x_9)
\]

\[
\frac{\partial \psi_7}{\partial x} = \frac{1}{A} (y_3 - y_9); \quad \frac{\partial \psi_7}{\partial y} = \frac{1}{A} (x_3 - x_9)
\]

\[
\frac{\partial \psi_9}{\partial x} = \frac{1}{A} (y_3 - y_7); \quad \frac{\partial \psi_9}{\partial y} = \frac{1}{A} (-x_3 + x_7)
\]

Therefore, this element makes the following contributions to the matrix \( M \)

\[
m = \frac{1}{2A} \begin{bmatrix}
3 & 7 & 9 \\
\downarrow & \downarrow & \downarrow \\
(y_7 - y_9)^2 + (-x_7 + x_9)^2 & (-y_3 + y_9)^2 + (x_3 - x_9)^2 \\
(y_7 - y_9)(y_3 - y_7) & (y_3 - y_9)(y_3 - y_7) & (y_3 - y_7)^2 + (-x_3 + x_7)^2 \\
(x_7 - x_9)(x_3 - x_7) & (x_3 - x_9)(-x_3 + x_7) \end{bmatrix}
\]

\section{Matlab code for building \( M \)}

The matlab code for building \( M \) is much simpler than the above examples because we let Matlab do all the computations.

Suppose that the mesh (with \( P \) nodes and \( T \) triangles) is represented by two arrays \( p \) and \( t \). The array \( p \) is \( P \times 3 \) and contains the locations of the vertices. The array \( t \) is \( T \times 3 \) for each triangle tells us the numbers of the vertices.
3.1 Example

Matlab code

```matlab
function [u, M, b] = fem2D(p, t, e)
P = size(p, 1);
T = size(t, 1);
M = sparse(P, P);
for ii=1:T % for each triangle
    ns = t(ii, :);
    X = [p(ns, :) [1; 1; 1] ];
    A = det(X);
    abc = [X\[1; 0; 0] X\[0; 1; 0] X\[0; 0; 1] ];
    for ii = 1:3
        for jj = 1:3
            M(ns(ii), ns(jj)) = M(ns(ii), ns(jj)) + A/2*(abc(1, ii)*abc(1, jj) + abc(2, ii)*abc(2, jj));
        end
    end
end

4 Imposing boundary conditions: Laplace’s equation on the unit disk

Suppose we need to Laplace’s equation on the unit disk (centered at the origin)
subject to the Dirichlet boundary conditions

\[ U_0(\theta) = \sin 4\theta \]

Let the list of boundary nodes be given in the array e. Then in order to incorporate the boundary conditions using the method of Lagrange multipliers we supplement the above code with the following:

```matlab
E=length(e);
for ii=1:E
    M(P+ii, e(ii)) = 1;
    M(e(ii), P+ii) = 1;
    x = p(e(ii), 1);
    y = p(e(ii), 2);
    theta = atan2(y, x);
    b(P + ii, 1) = sin(4*theta);
end
u = M\b;
u = u(1:P);
```
4.1 Plotting the Solution

The solution can be plotting using the trisurf command:

```matlab
>> trisurf(t, p(:, 1), p(:, 2), u);
```

We get the following plot:

![Plot of the solution](image)

4.2 Comparing to the true solution

The considered example is a special case where the true solution is known. It is given by

\[ u_{\text{true}}(x, y) = 4x^3y - 4xy^3. \]

To confirm that this function satisfies the equation, we need to show two things: 1 that its Laplacian is zero and 2 that it satisfies the boundary condition. To show that the Laplacian vanishes

\[ \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 24xy - 24xy = 0 \]

On the boundary \( x = \cos \theta \) and \( y = \sin \theta \), so

\[ 4x^3y - 4xy^3 = 4\cos^3 \theta \sin \theta - 4\cos \theta \sin^3 \theta = \sin (4\theta) \]

To judge how well we’ve done, let us find the largest deviation from the true solution:

```matlab
>> max(abs(u - (4*p(:, 1).^3.*p(:, 2) - 4*p(:, 1).*p(:, 2).^3)))
ans =
0.01107028372534
```

Not too good, not too bad. To plot the actual error use

```matlab
>> trisurf(t, p(:, 1), p(:, 2), u - (4*p(:, 1).^3.*p(:, 2) - 4*p(:, 1).*p(:, 2).^3))
```
The error improves with a finer mesh:

Here's what the solution looks like on a considerably finer mesh:
The error improves:

```matlab
>> max(abs(u - (4*p(:, 1).^3.*p(:, 2) - 4*p(:, 1).*p(:, 2).^3)))
ans =
  2.011053572200661e-004
```

5 **FEM in 3D**

The finite element procedure in three dimensions is virtually identical to the two dimensional version. What changes is the input geometry. Rather than triangles, the individual elements are tetrahedra. The array \( p \) is now \( P \times 3 \) and the array \( t \) is \( T \times 4 \). A typical tetrahedral mesh looks like this (surface plus a cross section):
5.1 How to test solutions

In order to test the correctness of your numerical procedure you need to pick a problem for which you know the analytical solution and compare the results you obtained numerically with that true solution. There is a simple way produce a problem with a known answer. Start by selecting any function function whose Laplacian vanishes. A few examples:

\[ u(x, y) = x^2 + y^2 - 2z^2 \]
\[ u(x, y) = e^x \cos y + e^y \cos z + e^z \cos x \]
\[ u(x, y) = xy + yz \]

Then, for a given domain \( \Omega \), simply evaluate your chosen \( u(x, y) \) on \( S \), the boundary of \( \Omega \).

Now, call the values you just calculated \( U_0 \) and solve Laplace’s equation on \( \Omega \) subject to the boundary condition

\[ u(x, y)|_S = U_0. \]

The true solution to this problem is \( u(x, y)! \)