PSBC Assignment 3: The Finite Element Method

December 7, 2015

Louis Runcieman

Department of Mathematics

University of Manchester

Academic year 2014-2015

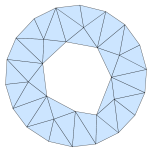
Abstract

The finite element method (FEM) is used to approximate solutions to partial differential boundary value problems. It subdivides the problem domain into finite elements and implements calculus of variations to minimise the associated error function. This paper explores the basic ideas surrounding the theory of FEM with all programming done through GNU Octave (please note, this may induce compatibility issues with MATLAB).

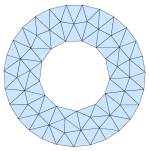
1 Introduction to `Distmesh`

The MATLAB package `Distmesh` developed by Per-Olof Persson and Gilbert Strang [1] creates simple, unstructured triangular (and tetrahedral) meshes of two and three dimensional geometrical surfaces; application of `Distmesh` gives a building tool for the FEM. The surfaces are defined by *signed distance functions* which give the shortest distance from any point to a boundary of the space. `Distmesh` internally distributes nodes in the domain and successively (*iteratively*) relocates them to obtain a force equilibrium; termination occurs when all nodes are approximately fixed in space. This allows the code to be short and simple which in turn enables the code to be easily manipulated for various uses. As with most coding this ease of use and simplicity admits drawbacks; particularly, the code is computationally expensive and needs refining for more complex shapes since it may not return well shaped meshes or properly terminate. Optimisation is available through C++ coding however this is beyond the scope of this paper. `Distmesh` can be easily adapted for use in Octave; to illustrate this, triangular meshes will be made of various examples in the Euclidean plane. Other packages which can give similar results include `Mesh2D` [2] and `Gmsh` [3].

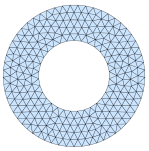
Example 1.1. Consider the cross section of a circular pipe with an circular inner tube of half the pipes radius. Three triangular meshes will be made of this object with varying mesh grains.



(a) 30% of radius.



(b) 20% of radius.



(c) 10% of radius.

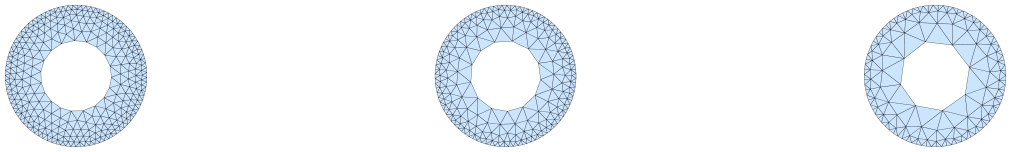
Figure 1: Triangular meshes of annulus with mesh grain as percentage of the radius of the outer circle.

Each triangle is known as an element (of the mesh), with two edges of only meeting at vertices of a triangle. This is the reason for a fine mesh needed to accurately represent a curved surface. Figure 1(a) outlines the correct shape however the inner tube boundary is neither regular nor circular. Decreasing the mesh length by 10% of the radius makes the triangulation finer. Figure 1(b) has the correct shape with a circular outer boundary; the inner boundary appears regular although it is not circular - the mesh must be finer for a more accurate representation of the object. Decreasing the mesh length by another 10% results in a sufficiently smooth representation, as shown in Figure 1(c).

Listing 1: Octave code to produce Figure 1.

```
1 for r=[0.3, 0.2, 0.1]
2 figure;
3 fd=@(p) ddiff(dcircle(p,0,0,1),dcircle(p,0,0,0.5));
4 [p,t]=distmesh2d(fd,@huniform,r,[-1,-1;1,1],[]);
5 end
```

Example 1.2. The previous example uses a uniform scaled edge length, `huniform`, for the mesh elements. This means the meshes themselves are regular. Triangulation may need to be refined at certain points on the surface (which may in turn represent a change in solution of the FEM in three dimensions). Consider again the cross sectional pipe; a refined outer boundary will be illustrated by giving a more coarse triangulation of the inner boundary. So the effects of this can be observed the mesh length will constant.



(a) `b=0.3`. (b) `b=0.15`. (c) `b=0.1`.

Figure 2: Triangular meshes of pipe with refined outer boundary.

Listing 2: Octave code to produce Figure 2.

```
1 a=1; h=0.06;
2 for b=[0.3,0.15,0.1]
3 fd=@(p) ddiff(dcircle(p,0,0,1),dcircle(p,0,0,0.5));
4 fh=@(p) b-a*dcircle(p,0,0,1);
5 figure; [p,t]=distmesh2d(fd,fh,h,[-1,-1;1,1],[]);
6 end
```

The value of `fh` scales the value of the mesh length; it is dependent, in this case, on the ratio between constants `a` and `b` (explaining why `a` has been arbitrarily chosen to be fixed at the value 1). The greater `a/b` is the less the relative refinement at the edges, as dsiplayed in Figure 2(a). This is since a constant term dominates `fh`, so the scaling is constant - meaning no refinement is made. Conversely a smaller value of `a/b`, as in Figure 2(c) gives a stronger refinement since the variable term is dominant. A drawback of the chosen scaling function `fh` is that it makes triangulation near the outer boundary more refined by reducing the refinement near the inner boundary; this means a very fine mesh would be required to represent the object smoothly with a refined outer boundary.

Example 1.3. Another tool of `Distmesh` allows fixed point nodes to be defined as an optional parameter of `distmesh2d`. This is particularly useful when defining polygon domains. Consider a kite domain with a coarse triangulation; by fixing nodes at each vertex of the kite the triangulation has a smoother and quicker stabilisation, as shown in the following diagram.

(a) No fixed points. (b) Fixed points.

Figure 3: Kite with and without the kite's vertices as fixed point nodes.

2 Cotangent Laplacian and the FEM

The FEM is used to find numerical approximations for partial differential boundary value problems; to do so, the continuous domain on which the problem is defined must be discretised. Meshes generated in Section 1 can accurately give such discretisations of the problem domain. The differential operators are then estimated using trigonometry [4]. In particular, the finite element approximation K of the Laplace operator $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ over a discretised domain is given by,

$$\begin{aligned} &K_{ij} = 0, \text{ if no single edge connects nodes } i \text{ and } j, \\ &K_{ij} = \frac{1}{2}(\cot(\theta_i^+) + \cot(\theta_i^-)), \text{ where } \theta_i^+ \text{ and } \theta_i^- \text{ are interior angles at vertices opposite edge } ij, \\ &K_{ii} = -\sum_{i \neq j} K_{ij}, \text{ to ensure column and row sums of } K \text{ vanish.} \end{aligned} \tag{1}$$

If the edge ij is on the boundary of the domain then the second term of the second equation in (1) will simply vanish. Hence the *cotangent Laplacian*, K, is defined.

Consider Poisson's equation,

$$\nabla^2 u = q, \tag{2}$$

for real valued functions u and q. Using the approximation of the Laplace operator, ∇^2, defined in (1) a solution to Poisson's equation can be approximated for given boundary data through the linear system of equations $Ku = q$, where values u and q are of discretised also (the same applies for Laplace's equation which is simply the homogeneous version of (2)). Generally K is not square so a solution of the form $u = K^{-1}q$ is not viable; a *least squares solution* can however be obtained since the system is overdetermined. The least square solution minimises the sum of squares of the errors of the system. Hence we derive the finite element method for the Laplace operator from the following steps:

- Divide the solution region into a mesh of finitely many elements through triangulation;
- Calculate the approximation K to ∇^2;
- Determine which nodes in the mesh are boundary nodes and which are interior nodes;
- Define solution to satisfy boundary data at boundary nodes;
- Determine a least squares solution of u at interior nodes by the known data of q and the values of u at boundary nodes.

Errors occur in approximating the problem domain, the Laplace operator and the least squares solution to the linear system of equations. Intuitively a mesh which more accurately and smoothly represents the problem domain will reduce all associated errors and give rise to a more accurate solution. The grain of the mesh is easily refined however the process becomes computationally expensive after relatively small refinements and an equilibrium between the two must be chosen. The following examples will give approximation solutions to Laplace's equation and Poisson's equation with Dirichlet and Neumann boundary data respectively; the mesh used will have a range of refinements and approximations will be compared with solutions analytically obtained.

Listing 3: Octave function to compute the cotangent Laplacian

```
function K=cotLap(p,t)
%COTLAP generates the cotangent Laplacian for given mesh data.
%       K:          Finite element approximation to Laplace operator.
%       P:          Node positions (Nx2).
%       T:          Triangle indices (NTx3).
%This function works by exploiting the given order of nodes in p to the
%edges and angles of triangles in t by the following labelling:
%As in p, the ith node's coordinates are p(i,:);
%the ith triangle, t(i,:), has vertices p(a,:), p(b,:), p(c,:) for some a, b, c;
%the first edge of the ith triangle is t(i,1)=p(a,:) -> t(i,2)=p(b,:);
%the second edge of the ith triangle is t(i,1)=p(a,:) -> t(i,3)=p(c,:);
%the third (and final) edge of the ith triangle t(i,2)=p(b,:) -> t(i,3)=p(c,:).
N=size(p,1); %Number of nodes.
s=size(t,1); %Number of triangle elements.
%Distance matrix D betweeen every pair of nodes. D(i,j) is the distance between
%nodes i and j; distance is zero if no single edge connects i and j.
%Limitation: edges measured more than once if shared by more than one triangle.
D=zeros(N); %Base from which the distance matrix is built.
for i=1:s
D(t(i,1),t(i,2))=d(p(t(i,1),:),p(t(i,2),:)); %d is the Euclidean norm.
D(t(i,1),t(i,3))=d(p(t(i,1),:),p(t(i,3),:));
D(t(i,2),t(i,3))=d(p(t(i,2),:),p(t(i,3),:));
D(t(i,2),t(i,1))=D(t(i,1),t(i,2)); %Ensuring the distance matrix is symmetric.
D(t(i,3),t(i,1))=D(t(i,1),t(i,3)); %Ensuring the distance matrix is symmetric.
D(t(i,3),t(i,2))=D(t(i,2),t(i,3)); %Ensuring the distance matrix is symmetric.
end
%Angle matrix A represents angles of each triangle element in the mesh. Cosine
%rule is used to compute angles. A(i,:) contains the three angles in triangle i.
%A(i,j) for j=1,2 or 3 is the interior angle in triangle i at node p(t(i,j),:).
A=zeros(size(t)); %Base from which the angle matrix is built.
for i=1:s
A(i,1)=acos((D(t(i,1),t(i,2))^2 + D(t(i,1),t(i,3))^2 -
       D(t(i,2),t(i,3))^2)/(2*D(t(i,1),t(i,2))*D(t(i,1),t(i,3))));
A(i,2)=acos((D(t(i,2),t(i,1))^2 + D(t(i,2),t(i,3))^2 -
       D(t(i,1),t(i,3))^2)/(2*D(t(i,2),t(i,1))*D(t(i,2),t(i,3))));
A(i,3)=acos((D(t(i,3),t(i,1))^2 + D(t(i,3),t(i,2))^2 -
       D(t(i,1),t(i,2))^2)/(2*D(t(i,3),t(i,1))*D(t(i,3),t(i,2))));
end
%Cotangent Laplacian matrix K. K(i,j) is 1/2 the sum of the cotangents of the
```

```
%angles opposite the edge ij.
%In the following for loop the first term of the RHS of each equation ensures
%addition of both (if more than one) angles opposite the given edge.
K=zeros(N); %Base from which cotangent Laplacian matrix is built.
for i=1:s
  if t(i,1)<t(i,2) %Ensures K is upper triangular. First edge of ith triangle.
  K(t(i,1),t(i,2))=K(t(i,1),t(i,2))+0.5*cot(A(i,3));
  else
  K(t(i,2),t(i,1))=K(t(i,2),t(i,1))+0.5*cot(A(i,3));
  end
  if t(i,1)<t(i,3) %Ensures K is upper triangular. Second edge of ith triangle.
  K(t(i,1),t(i,3))=K(t(i,1),t(i,3))+0.5*cot(A(i,2));
  else
  K(t(i,3),t(i,1))=K(t(i,3),t(i,1))+0.5*cot(A(i,2));
  end
  if t(i,2)<t(i,3) %Ensures K is upper triangular. Third edge of ith triangle.
  K(t(i,2),t(i,3))=K(t(i,2),t(i,3))+0.5*cot(A(i,1));
  else
  K(t(i,3),t(i,2))=K(t(i,3),t(i,2))+0.5*cot(A(i,1));
  end
end
K=K+K'; %Lets K(i,j)=K(j,i) for all i<j, making K symmetric.
for i=1:N %Diagonal element K(i,i) is equal to negative the sum of the row i.
K(i,i)=-sum(K(i,:));
end
```

The function `cotLap` exploits the order given to the nodes in `p`, giving an efficient and cheap computation of the cotangent Laplacian. Dirichlet and Neumann BVPs with problem region $\Omega \subseteq \mathbb{R}^2$ and local Cartesian coordinate system (x, y) will now be considered to give example to `cotLap`.

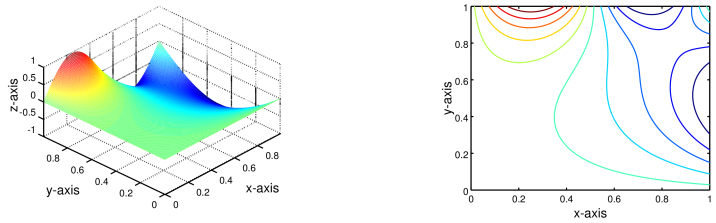
Example 2.1 (Dirichlet problem)**.** Consider the Laplace equation with Dirichlet boundary data on a unit square domain,

$$\begin{aligned} \nabla^2 u &= 0, \\ u(0,y) &= f_1(y) = 0, \\ u(1,y) &= f_2(y) = -sin(\pi y), \\ u(x,0) &= g_1(x) = 0, \\ u(x,1) &= g_2(x) = sin(2\pi x). \end{aligned} \tag{3}$$

The solution to this can be determined analytically by separation of variables, giving,

$$u(x,y) = \frac{\sinh(2\pi y)\sin(2\pi x)}{\sinh(2\pi)} - \frac{\sinh(\pi x)\sin(\pi y)}{\sinh(\pi)}, \tag{4}$$

after applying the boundary conditions in (3). The following figure is a 3 dimensional (3D) and contour graphical representation of the solution.



(a) Analytical solution, 3D. (b) Analytical solution, contour.

Figure 4: Contour and 3 dimensional visualisation of analytical solution to BVP (3).

Listing 4: Octave code to produce Figure 4

```
1 xi=0:0.01:1; [qx,qy]=meshgrid(xi,xi);
2 qz=(sinh(2*pi*qy)/sinh(2*pi)).*(sin(2*pi*qx))-
3     ((sinh(pi*qx)/sinh(pi)).*(sin(pi*qy)));
4 figure; mesh(qx,qy,qz); figure; contour(qx,qy,qz);
```

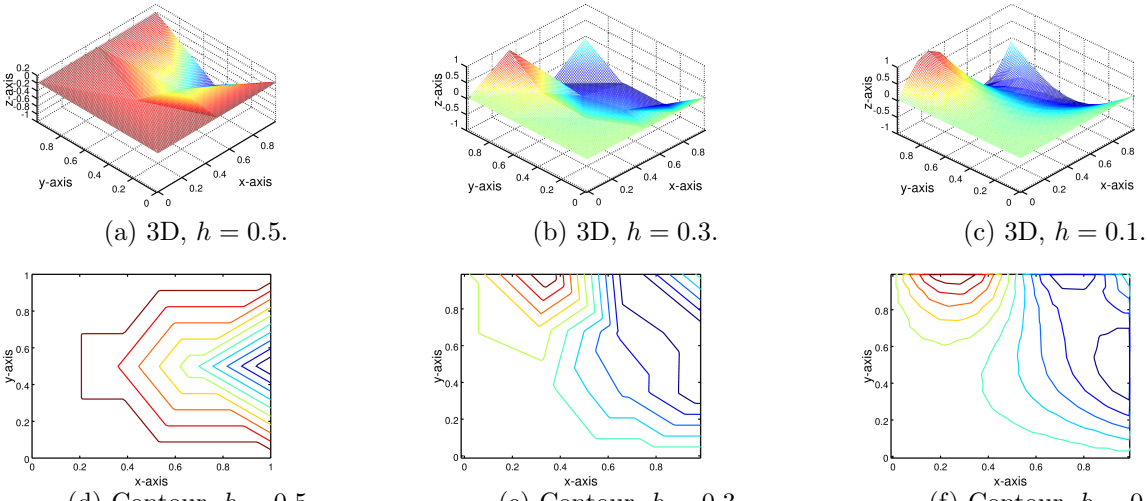
From the FEM, we find an approximation K to the Laplace operator ∇^2. We hence have the linear system of equations to solve,

$$Ku = 0. \tag{5}$$

In Octave, the approximation `u` to solution u is split into 5 parts to implement the boundary conditions and ensure we have an overdetermined systems. Each of the 4 edges `bvtx1, bvtx2, bvtx3, bvtx4` of the domain and the interior of the domain `invtx` constitute the domains of the 5 parts, giving,

$$\texttt{K(u}_{\texttt{ivtx}}\texttt{+u}_{\texttt{bvtx1}}\texttt{+u}_{\texttt{bvtx2}}\texttt{+u}_{\texttt{bvtx3}}\texttt{+u}_{\texttt{bvtx4}}\texttt{)=0.}$$

and hence by using a least squares solution we find the least squares solution to `u`, on the interior vertices given at line **??** in Listing 6. The solution as determined by the FEM produces the following,



(a) 3D, $h = 0.5$. (b) 3D, $h = 0.3$. (c) 3D, $h = 0.1$.

(d) Contour, $h = 0.5$. (e) Contour, $h = 0.3$. (f) Contour, $h = 0.1$.

Figure 5: Contour and 3 dimensional visualisations of approximate solution to BVP (3) with mesh grain h.

Numerical error is determined by comparing the Euclidean norm and the Infinity norm of the difference in value of the approximate solution and the analytical solution over a meshgrid of argument values (which should vanish), given by the following table.

Table 1: Table comparing norms of a variety of mesh grains for approximate solution to BVP (3).

h	0.5	0.3	0.1
d_∞	73.3982	1.9003	0.46842
d_2	43.6494	2.9353	1.3923

As the table evidences, the error significantly declines with a mesh grain of less than 0.5; a mesh grain of order less than 10^{-1} is gives an stronger approximation to the BVP however this becomes computationally expensive.

Listing 5: Octave code to solve BVP (3).

```
for r=0.1:0.2:0.5
fd=@(p) drectangle(p,0,1,0,1);  %Produces mesh of unit square.
[p,t]=distmesh2d(fd,@huniform,r,[0,0;1,1],[0,0;1,0;1,1;0,1]);
N=size(p,1); %Total number of nodes.
K=cotLap(p,t); %Compute the cotangent Laplacian.
e=boundedges(p,t); %Outputs all bounding edges of the domain.
bvtx=sort(e(:,1)); %Outputs all bounding nodes (in ascending order).
ivtx=[1:N]'; %Outputs a vector containing label of all nodes.
ivtx=setdiff(ivtx,bvtx); %Difference of nodes and bvtx, leaving interior nodes.
%Splitting the 4 edges of the square for each boundary condition.
bvtx1=bvtx; bvtx2=bvtx; bvtx3=bvtx; bvtx4=bvtx;
err=1e-5; %Error from floating arithmetic.
for i=1:size(bvtx,1)
  if abs(p(bvtx(i),1))>err %x entry is approx. 0.
  bvtx1(i)=0; %Cannot used [] since indices would not match.
  end
  if abs(p(bvtx(i),1)-1)>err %x entry is approx. 1.
  bvtx2(i)=0; %Cannot used [] since indices would not match.
  end
  if abs(p(bvtx(i),2))>err %y entry is approx. 0.
  bvtx3(i)=0; %Cannot used [] since indices would not match.
  end
  if abs(p(bvtx(i),2)-1)>err %y entry is approx. 1.
  bvtx4(i)=0; %Cannot used [] since indices would not match.
  end
end
bvtx1(bvtx1==0)=[]; bvtx2(bvtx2==0)=[]; bvtx3(bvtx3==0)=[]; bvtx4(bvtx4==0)=[];
```

```
28 %Removes zero entries. The following is the Dirichlet boundary data.
29 f1=0*bvtx1; f2=-sin(pi*p(bvtx2,2)); g1=0*bvtx3; g2=sin(2*pi*p(bvtx4,1));
30 %Interior solution.
31 U=K(:,ivtx)\(-K(:,bvtx1)*f1-K(:,bvtx2)*f2-K(:,bvtx3)*g1-K(:,bvtx4)*g2);
32 u=zeros(N,1); %Base for our solution u.
33 u(ivtx)=U;% Solution u inside domain.
34 u(bvtx1)=f1; u(bvtx2)=f2; u(bvtx3)=g1; u(bvtx4)=g2;%Solution u on each edge.
```

Example 2.2 (Neumann problem)**.** Consider Poisson's equation with non-trivial homogeneous Neumann boundary conditions over domain a unit disc Ω, in polar coordinates (r, θ),

$$\begin{aligned} \nabla^2 u &= \frac{1}{r}e^r + e^r - 4r + \frac{1}{r}, \\ \frac{\partial u}{\partial n} &= 0, \end{aligned} \tag{6}$$

such that $\int_\Omega q = \int_{\partial\Omega} g$, where n is the outward facing norm to the boundary, $\partial\Omega$. The soltion to this can be determined analytically giving,

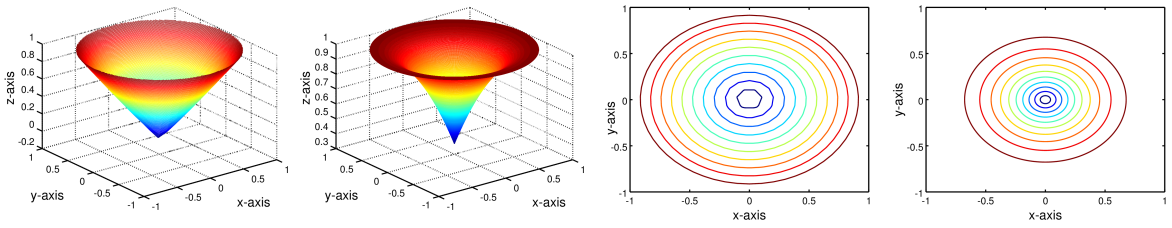
$$u(r, \theta) = e^{r-1} - r^2 + r. \tag{7}$$

By using the same approximation the Laplace operator as in Example 2.1, a FEM solution is found by replacing lines `10` to `31` of Listing 6 with the following listing (the same mesh grains of `0.5, 0.3` and `0.1` have been used).

Listing 6: Octave code to solve BVP 6.

```
1 f=zeros(N,1); %Base for construction of RHS of Poisson's equation.
2 [t,r]=cart2pol(p(:,1),p(:,2)); %Converting Cartesian to Polar coordinates.
3 f=exp(r).*(1+1./r)-4*r+1./r; %The RHS function of Poisson's equation.
4 q=1/6*sqrt(3)*3/2*h^2.*f;
5 %Greater accuracy, based on area of surround traingles for each vertex.
6 u(bvtx)=0; %Solution vanishes on boundary for homogenous Neumann BCs.
7 u(ivtx)=K(ivtx,ivtx)\q(ivtx); %Least squares solution for solution at interior.
```

The following figure visualises the analytical solution and the approximate solution at a mesh grain of `0.1`.



(a) Approx. u, 3D. (b) Analytical u, 3D. (c) Approx. u, contour. (d) Analytical u, contour.

Figure 6: Contour and 3D visualisations of approx. and analytical solution to BVP (6) with mesh grain $h = 0.1$.

Similarly to Example (3), the difference in values of the approximate and analytical solution will now be compared.

Table 2: Table comparing norms of a variety of mesh grains for approximate solution to BVP (6).

h	0.5	0.3	0.1
d_∞	33.1987	18.9201	10.3842
d_2	25.6587	7.9258	5.3944

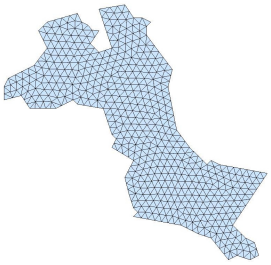
Larger relative errors are apparent for this type of boundary data compared with Dirichlet boundary data; it appears the function `cotLap` maps non-convex points together, which explains why the visualisation of the solution given in Figure 6(a) takes a straight line from the disk's boundary to it's minimum, which consequently effects the contour lines.

A harmonic function u on domain Ω is a twice continuously differentiable real valued function which satisfies Laplace's equation. Within the interior domain for which u is defined u cannot exhibit a true local maximum, meaning u is either constant or for any given point there exists another point arbitrarily close at which u takes larger values. This defines the *maximum principle* for harmonic functions, which states the maximum of u will be found on the boundary $\partial\Omega$ of the domain on which u is defined. The same can be said for u exhibiting a true local minimum, with the choice of any point giving rise to u taking smaller values arbitrarily close to the point; this is known as the *minimum principle*.

The solution obtained in Example 2.1, illustrated by Figure 5, is shown to satisfy the principle.

3 Hypothetical Application of the FEM

(a) Pulau Bangka.



(b) Pulau Bangka triangulated.

Figure 7: Triangulation of Pulau Bangka

Figure 7(a) was printed onto graph paper, an arbitrary origin was picked and all significant features of the coastline of Pulau Bangka were given a node, with entries as the coordinates relative to the origin. These nodes were imported into a matrix `pv` in Octave and using a similar code as used to produce Figure 3(b) the triangulated mesh of Pulau Bangka, Figure 7(b), was produced. For the purpose of this hypothetical situation the island is assumed flat and with homogeneous soil, for which Figure 7(b) suffices to approximate.

The government of Pulau Bangka has a pollution problem; a toxic liquid is being injected inland, away from the coast. The location of the pollution needs to be ascertained before a solution can be made; the concentration at the coastline is known to be zero and the concentration inland is given. Combining this with the assumption of homogeneous soil allows the problem to be modelled similarly to Example 3 but with Poisson's equation $\nabla^2 u = q$, where term q represents the concentration of the pollution. The object is to solve Poisson's equation for u so that the source of the highly concentrated pollution can be located. With a given q, contour plots of the triangulated island domain give the following figure.

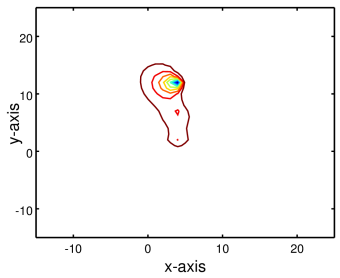


Figure 8: Pollution contour source.

To obtain a clearer picture of this contour plot, the island outline could be overlaid on the contour with the same coordinate axes as used in the triangulation of the island. It is found the highest concentration point has an average value of approximately 0.1, a maximum value of 1; this affects a radius of approximately 1 (this is a natural length scale, where the island's maximum height is 30). This means if approximately 30 test sites where chosen, every possible area of land would be tested and the source would be located when a pollution average of 0.1 is found. This is, however, a very expensive method. If approximately 15 test sites were chosen and any pollutant was registered with a value of greater than 0.1 it can be assumed the pollutant source is with a radius of 1 from that point. Fewer test sites would be cheaper still, however it is less likely the pollutant source would be found on the first attempt.

References

[1] Per-Olof Persson and Gilbert Strang (2012), Distmesh, http://persson.berkeley.edu/distmesh/

[2] Darren Engwirda, http://www.mathworks.com/matlabcentral/fileexchange/25555-mesh2d-automatic-mesh-generation updated 2009.

[3] Christophe Geuzaine and Jean-Franois Remacle, http://geuz.org/gmsh/ updated 2015.

[4] O. C. Zienkiewicz, (2013), The Finite Element Method: Its Basis and Fundamentals