Short Introduction to Numerical Methods

Finite Element Method & Method of Moments
Application for 2D Electrostatic Problems

Summer Term
2011
Table of Contents

1. Introduction .................................................................................................. 1

2. Finite Element Method (FEM) ....................................................................... 3
   2.1 Discretization of the geometry domain .............................................. 3
   2.2 Postprocessing of the meshing data ................................................... 6
   2.3 Choice of proper basis functions ......................................................... 8
   2.4 Derivation of the linear equation system for a single element .......... 11
   2.5 Formulation of the global matrix by assembling all the elements ..... 17
   2.6 Postprocessing of the results (Calculation of the capacity) .......... 20
   2.7 Display of the numerical results ........................................................ 22
   2.8 References ........................................................................................ 24

3. Method of Moments (MoM) ...................................................................... 25
   3.1 Formulation of integral equation using the $2D$ free space Green’s function ... 25
   3.2 Discretization of the problem ........................................................... 26
   3.3 Formulation of the global matrix ...................................................... 28
   3.4 Set of the reference point for potentials .......................................... 31
   3.5 Computation of the sources and of the capacitance ......................... 31
   3.6 References ........................................................................................ 32
1. Introduction

In electromagnetics there are many problems that are governed by partial differential equations (PDEs). The PDEs can be divided in many different categories according to i.e. their degree (1\textsuperscript{st} order, 2\textsuperscript{nd} order), to their coefficients (constant, special dependent), to their time dependency (elliptic, parabolic). In simple geometries, the PDEs can be solved analytically, but as soon as the structure becomes a little more complicated, then there is no analytical solution and the PDE should be solved in a numerical way. There have been developed plenty of numerical techniques for solving PDEs, but 2 of the most popular are Finite Element Method (FEM) and Method of Moments (MoM).

In the present manuscript, we will try to demonstrate the basic concepts of these methods through a very basic and common example taken from the electromagnetics area. In particular, we will try to solve the electrostatic problem governed by the Laplace equation:

\[ \nabla^2 \varphi = 0 \]  
(1.1)

or in cartesian coordinates:

\[ \frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = 0 \]  
(1.2)

Every step will be clearly described, with as few references to mathematical background as possible. Moreover, a simple Matlab code accompanies each step, so as in the end the whole code for the certain problem is given.

The geometry of the structure can be seen in Figure 1.1, where we force the inner conductor to be in constant voltage \( V=100 \) Volt, while the outer conductor is grounded, which means \( V=0 \) Volt.

Our target will be to compute the potential in all the area between the 2 conductors, or in a more mathematical notation, to solve the problem

\[ \nabla^2 \varphi = 0 \]  
(1.3)

\[ \varphi|_{\text{inner conductor}} = 100 \]
\[ \varphi|_{\text{outer conductor}} = 0 \]

As a postprocessing result we will try to calculate the capacity of the above structure.
It is obvious that no analytical solution can be obtained for the above problem and the given geometry, so our only tool to find the solution will be a numerical method.
2. Finite Element Method (FEM)

The main concept of the FEM is based on subdividing the geometrical domain of the problem into smaller subdomains, called elements. The unknown quantity (the potential in our case) will be the solution of the same PDE as the one of the original problem, but with different boundary conditions on the surface of each element. Then by combining the solution obtained in all the elements, we will be able to get the solution for the original problem.

The basic steps in solving the problem stated above are:

- Discretization of the given geometry using finite elements
- Preprocessing of the meshing data
- Choice of proper basis functions
- Derivation of the linear equation system for a single element
- Formulation of the global matrix by assembling all the elements
- Imposing boundary conditions
- Solving the linear system of equations using linear algebra techniques
- Postprocessing of the results in order to calculate the capacity of the structure
- Display of the numerical results

All the above mentioned steps will be explained in this chapter including, too, a simple Matlab code that performs the described calculations. This chapter is based on [1], [2], [3] and [4], where the reader can also find further details.

2.1 Discretization of the geometry domain

In order to apply the FEM, the domain in which we would like to solve our PDE governed problem should be divided into elements. The elements can be of many types according to the dimension of the domain under study (2D or 3D) and some examples are listed below:

<table>
<thead>
<tr>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>Linear</td>
</tr>
<tr>
<td>Triangle</td>
<td>Tetrahedron</td>
</tr>
<tr>
<td>Isoparametric</td>
<td>Prism</td>
</tr>
<tr>
<td>Hexahedron</td>
<td></td>
</tr>
</tbody>
</table>

*Table 2.1* Different types of elements.
The most flexible and popular elements are triangles and tetrahedral in 2D and 3D, respectively. However, in special cases more sophisticated type of elements, like prisms and hexahedra can also be chosen. Moreover, curved linear elements provide us with better approximation of curved surfaces. Concerning our problem, as it is limited in 2D, we will use triangular meshing.

Regarding the meshing, we choose to use readily available tools, that perform the meshing and provide us directly with the coordinates of the nodes of the elements. Restricted to 2D problems, Matlab Partial Differential Equations toolbox provides a mesher based on Delauney triangulation, which is going to be used in our case. The specific details of this tool are described below.

In order to describe the geometry of our structure, a matrix under the name \( gd \) (geometry description) with all the geometrical data should be implemented. We divide our domain in areas of simple geometrical shapes, which can be circles, polygons, orthogons or ellipses. Every column of the matrix \( gd \) contains all the data that describe the corresponding subdomain. This means that the matrix \( gd \) should have as many columns as the number of the simple geometrical shapes that we used in order to describe our complex geometry.

As each simple shape requires different information in order to be fully described, each column should be filled in the proper way according to Table 2.2.

<table>
<thead>
<tr>
<th>( id )</th>
<th>Circle</th>
<th>Polygon</th>
<th>Orthogon</th>
<th>Ellipse</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Geometrical data</th>
<th>( x ) coordinate of the center</th>
<th>( n ) edges of polygon</th>
<th>‘4’ edges</th>
<th>( x ) coordinate of the centre</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y ) coordinate of the center</td>
<td>( x_1 ) coordinate</td>
<td>( x_1 ) coordinate</td>
<td>( y ) coordinate of the centre</td>
<td></td>
</tr>
<tr>
<td>( r ) radius of the circle</td>
<td>( x_2 ) coordinate</td>
<td>( : )</td>
<td>( a ) length of semimajor axis</td>
<td></td>
</tr>
<tr>
<td>( : )</td>
<td>( x_4 ) coordinate</td>
<td>( b ) length of semiminor axis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_n ) coordinate</td>
<td>( y_1 ) coordinate</td>
<td>( \theta ) angle of rotation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y_1 ) coordinate</td>
<td>( : )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y_2 ) coordinate</td>
<td>( y_4 ) coordinate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( : )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( y_n ) coordinate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[Table 2.2\] Description of geometry matrix \( gd \).
For our problem the matrix $gd$ should have the following form:

```matlab
% Defining the geometry
xmin=-1;
xmax=1;
ymin=0;
ymax=1;
gd=[3 4 xmin xmin 0 0 0.75 ymax ymax 0.75;
    3 4 0 0 xmax xmax 0.25 ymax ymax 0.25;
    3 4 -0.5 -0.5 xmax xmax ymin 0.25 0.25 ymin;
    3 4 xmin xmin -0.5 -0.5 ymin 0.75 0.75 ymin];
gd=gd';
```

The matrix $gd$ created before will be used as an input data for the function $\text{decsg}(dg)$, which will create a new matrix named $dl$, with the same geometrical information with $gd$ but in a form more suitable for the meshing functions to handle with.

In particular, each column of $dl$ describes a different curve of the borders of the geometrical shapes we divided the domain, like the matrix $gd$, but now without any overlapping. As above, each type of curve should be described in a different way. Since we are not going to call any of the elements of the matrix $dl$ directly, no more details about this matrix will be given.

The next step is to call the meshing function that will use the geometrical information about our domain stored in the matrix $dl$. The function's syntax is the following:

```matlab
dl=\text{decsg}(dg);
[p,e,t]=\text{initmesh}(dl);
```

The function $\text{initmesh}()$ returns 3 matrices that contain all the information about the triangles in which our domain is meshed and also some linking data between the different triangles. In more details:

Matrix $p$ has dimension $[2 \times Nn]$, where $Nn$ is the number of the nodes. The $1^{st}$ and the $2^{nd}$ row contain the $x$ and $y$ coordinates of the nodes. Thus, in order to call the $x$ coordinate of the $i^{th}$ element we should type $p(1,i)$.

Matrix $e$ has dimension $[7 \times Nd]$, where $Nd$ is the number of the boundary edges of the shapes previously defined that form the mesh. For each edge, the $1^{st}$ and $2^{nd}$ row define the starting and the ending node that form this edge. The $3^{rd}$ to $5^{th}$ row include some other information about the position of the edges that will not be used in our case. The $6^{th}$ and $7^{th}$ rows contain the values of the $ids$ of the subdomain areas into which we divided our domain when we were forming the matrix $gd$ and which lay right and left of the edge, respectively. Attention should be paid that the areas that were not defined have $id = 0$. This information will be useful in order to detect the edges, and subsequently the nodes that lie on border surfaces, as for that edge, one of the rows $6^{th}$ or $7^{th}$ should have the value 0.

Matrix $t$ has dimension $[4 \times Ne]$, where $Ne$ is the number of the elements that form the mesh. The first 3 rows of each column include the $id$ number of the nodes that form the $i^{th}$ triangular element and can be obtained as $t(1,i)$, $t(2,i)$ and $t(3,1)$. The $4^{th}$ row contains the $id$ of the subdomain to which the particular element belongs.
If we need to make the mesh denser, which means using more triangles in order to cover the domain of our problem we can call the function `refinemesh()` in the following way:

```plaintext
[p,e,t]=refinemesh(dl,p,e,t);
```

This function divides the triangles created by the `initmesh()`, such as the final mesh consists of more triangles.

At this point, it should be noted that both `initmesh()` and `refinemesh()` and all the other meshing functions and algorithms try to divide the surface in triangles that are almost equilateral. This improves the accuracy of the solution of the numerical method, as the nodes are more uniformly dispersed in the surface.

If we want to mesh more complex geometries in $2D$ or $3D$, or even write scripts that create the geometry, external meshing utilities can be used. One very popular program is Gmsh [5], which provides both a command line and a graphical interface, a printscreen of which is given in Figure 2.1. Just to be noted that the position of the nodes and the connectivity matrix between the meshed elements and their nodes is given in ASCII code in a *.msh file and in order to be processed by Matlab, a M-file that reads this data should be implemented.

2.2 Preprocessing of the meshing data

The matrices $p$, $e$ and $t$ contain all the information about the geometry and the mesh. As a first step of preprocessing, before trying to solve the PDE, we should identify the nodes that lie on boundary surfaces and assign to them the corresponding boundary conditions, in our case Dirichlet.

Regarding the Dirichlet boundary condition, it describes the boundary condition for which the unknown variable (in our case the potential) has a fixed and known value. If we recall that our final target using the FEM analysis is to compute the value of the unknown variable in all the nodes of the
mesh, this means, that the value of this variable on the boundary nodes is already known, so we should exclude these nodes from our unknowns.

The Dirichlet condition is not the only boundary condition that can appear in such problems. However it is the easiest to be implemented, so this first example contains only this boundary condition.

Other often appearing boundary conditions are the Neumann condition (especially in case of symmetry planes), Absorbing Boundary Condition (ABC), Perfect Matched Layer (PML) etc. ABC and PML conditions are used for radiation problems.

In order to assign the Dirichlet boundary condition to the boundary elements, we define 2 arrays $kn$ and $boundary\_val$, with $Nn$ elements each. Regarding the $kn$ array, if the $i^{th}$ element is 0, this means that this node is on the boundary and if it is 1, it means that the node is internal. It is easy to find out the external nodes, as they are the ones for which the edge they belong to lies on the subdomain with $id = 0$.

The numbering and the mesh of the subdomains that the functions $initmesh()$ and $refinemesh()$ produce can be seen in Figures 2.2(a) and 2.2(b) respectively.

![Figure 2.2](image)

(a) Domain enumeration. (b) Mesh of the geometry.

The $boundary\_val$ array keeps the value of the potential that is known for the nodes to which the Dirichlet boundary condition is applied. The following code performs the 2 tasks mentioned before:

```matlab
%kn(i)=0 if the node is boundary and kn(i)=1 if the node is inner
kn=zeros(1,Nn);
boundary_val=zeros(1,Nn);
for id=1:Nn
    kn(id)=1;
    if (p(1,id)==-1)||(p(2,id)==1)||(p(1,id)==1)||(p(2,id)==0)
        boundary_val(id)=0; %external boundary voltage
    else
        boundary_val(id)=100; %inner boundary voltage
    end
end
```
The enumeration of the nodes that initmesh() provides is not very comfortable for the coding of the FEM solution. We would prefer, all the nodes associated with an unknown value to be first, followed by the nodes whose potential value is known due to the Dirichlet condition. Instead of reordering all the matrices, it is more convenient to create a mapping between the initial enumeration and the desirable one described above. This mapping should be implemented in both directions, that is we create an array called index, which, given an id of the old enumeration, it returns the id of the same node but in the new enumeration and another array called index2 that does exactly the opposite. The code for this is shown below:

```matlab
for id=1:size(e,2)
    if e(6,id)==0 || e(7,id)==0
        kn(e(1,id))=0;
        kn(e(2,id))=0;
    end
end
```

At this point the preprocessing of the data is finished.

### 2.3 Choice of proper basis functions

For the time being we focus on an arbitrary triangular element. We will evaluate a linear transformation so as to map this arbitrary triangle to the master triangle. Thus, every triangle that is
created from the meshing procedure will be transformed to the same triangle and all the numerical processing will be implemented for this master triangle.

In order to define the described transformation, we should keep in mind that each of the vertices of the arbitrary triangle should be mapped to one of the vertices of the master triangle.

In other words, we want to define 2 functions

\[ x(\xi, \eta) = a_1^x + b_1^x \xi + c_1^x \eta \]  
\[ y(\xi, \eta) = a_2^x + b_2^x \xi + c_2^x \eta \]  

given the following mapping information:

<table>
<thead>
<tr>
<th>Point 1</th>
<th>(x_1, y_1)</th>
<th>(0,0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point 2</td>
<td>(x_2, y_2)</td>
<td>(1,0)</td>
</tr>
<tr>
<td>Point 3</td>
<td>(x_3, y_3)</td>
<td>(0,1)</td>
</tr>
</tbody>
</table>

Table 2.3 Mapping from initial to master triangle.

This point mapping is illustrated in Figure 2.3:

\[ x(\xi, \eta) = x_1^e + (x_2^e - x_1^e)\xi + (x_3^e - x_1^e)\eta \]  
\[ y(\xi, \eta) = y_1^e + (y_2^e - y_1^e)\xi + (y_3^e - y_1^e)\eta \]  

and

\[ \xi(x, y) = \frac{(y_2^e - y_3^e)(x - x_3^e) + (x_3^e - x_2^e)(y - y_3^e)}{|T|} \]  
\[ \eta(x, y) = \frac{(y_3^e - y_1^e)(x - x_3^e) + (x_1^e - x_3^e)(y - y_3^e)}{|T|} \]  

where
\[ T = \begin{bmatrix} x_1^e - x_3^e & x_2^e - x_3^e \\ y_1^e - y_3^e & y_2^e - y_3^e \end{bmatrix} \]  

(2.4)

and \(|T| = 2A^e, A^e\) being the surface of the \(e^{th}\) element.

We suppose now that somehow we have already solved our problem in this certain triangle, which means that we have computed the value of the potential \(\varphi_1^e, \varphi_2^e, \varphi_3^e\) in the 3 nodes of the triangle.

Then, the potential inside the triangle, given the value of the potential in the 3 nodes, should be computed by the following formula:

\[ \varphi_\xi(\xi, \eta) = \varphi_1^e N_1(\xi, \eta) + \varphi_2^e N_2(\xi, \eta) + \varphi_3^e N_3(\xi, \eta) = \sum_{i=1}^{3} \varphi_i^e N_i(\xi, \eta) \]  

(2.5)

where \(\varphi_i^e\) is the computed potential on the \(i^{th}\) node of the \(e^{th}\) triangle and \(N_i\) are the so called basis or interpolation functions. Their usage is to make an estimation of the potential in the whole area of the triangle given the potential on the 3 points of the triangle. They can be chosen to be a linear combination of the coordinates \(\xi, \eta\), which leads to 1\(^{st}\) order basis functions. However, in many cases, in order to increase the approximation order of our interpolation, we can choose also higher order polynomials as basis functions. At this point it should be mentioned that as the set of basis function should be complete, if we choose higher order interpolation, this increases the number of basis functions and thus the vertices on the element, giving to it more shaping flexibility. But we pay this improvement of accuracy with extra analytical and computational cost. Thus, in this report we will consider 1\(^{st}\) order linear basis functions, i.e.:

\[ N_i(\xi, \eta) = c_{1i} + c_{2i}\xi + c_{3i}\eta \]  

(2.6)

In order to compute these basis functions \(N_i(\xi, \eta)\), we should keep in mind the following:

\[ V_1(0,0) = V_1^e \]  

(2.7a)

\[ V_2(1,0) = V_2^e \]  

(2.7b)

\[ V_3(0,1) = V_3^e \]  

(2.7c)

By taking into account the above equations, we conclude that

\[ N_1(\xi, \eta) = 1 - \xi - \eta \]  

(2.8a)

\[ N_2(\xi, \eta) = \xi \]  

(2.8b)

\[ N_3(\xi, \eta) = \eta \]  

(2.8c)

These functions \(N_i(\xi, \eta)\) are known as simplex or barycentric or natural or space coordinates for 2D.

For a given point \(P(\xi, \eta)\) we draw the 3 triangles that can be formed with the 3 nodes of the initial triangle and the point \(P\). Each coordinate \(N_i\) indicates the normalized surface of the triangle that does not include the \(i^{th}\) node, as it is shown in Figure 2.4.
In other words,

\[ N_1 = \frac{A_1}{A} \] (2.9a)
\[ N_2 = \frac{A_2}{A} \] (2.9b)
\[ N_3 = \frac{A_3}{A} \] (2.9c)

As it is obvious,

\[ N_1 + N_2 + N_3 = 1 \] (2.10)

which is a property of the simplex coordinates in all dimensions.

2.4 Derivation of the linear equation system for a single element

The general form of the PDE (in Cartesian coordinates) that is often asked to be solved in electromagnetic problems is the following:

\[
\frac{\partial}{\partial x} \left( a_x \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( a_y \frac{\partial u}{\partial y} \right) + \beta u = g
\] (2.11)

If we move everything to the left side we get the so called residual formulation which is valid for each element of our meshed structure:

\[
r^e = \frac{\partial}{\partial x} \left( a_x \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( a_y \frac{\partial u}{\partial y} \right) + \beta u - g
\] (2.12)

The exact solution of the differential equation would yield

\[
r^e_{\text{exact}} = 0
\] (2.13)

for each element.
In our approximate solution, we will try to fulfill this target in an average way over the surface of each element with a weight function \( w \):

\[
\iint_{\Omega^e} w \left[ \frac{\partial}{\partial x} \left( a_x \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( a_y \frac{\partial u}{\partial y} \right) + \beta u - g \right] \, dx \, dy = 0
\]  \hspace{1cm} (2.14)

Recalling the identity:

\[
\frac{\partial}{\partial x} \left( w a_x \frac{\partial u}{\partial x} \right) = \frac{\partial w}{\partial x} \left( a_x \frac{\partial u}{\partial x} \right) + w \frac{\partial}{\partial x} \left( a_x \frac{\partial u}{\partial x} \right)
\]  \hspace{1cm} (2.15)

(2.14) can be written as

\[
\iint_{\Omega^e} \left[ \frac{\partial}{\partial x} \left( w a_x \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( w a_y \frac{\partial u}{\partial y} \right) \right] \, dx \, dy - \iint_{\Omega^e} \left[ a_x \frac{\partial w}{\partial x} \frac{\partial u}{\partial x} + a_y \frac{\partial w}{\partial y} \frac{\partial u}{\partial y} \right] \, dx \, dy + \iint_{\Omega^e} \beta w u \, dx \, dy
\]

\[
= \iint_{\Omega^e} w g \, dx \, dy
\]  \hspace{1cm} (2.16)

Now, we will treat the 1st integral in the following way using Green’s theorem:

\[
\iint_{\Omega^e} \nabla \cdot A \, dS = \oint_{\Gamma^e} A \cdot \hat{a}_n \, dl
\]  \hspace{1cm} (2.17)

where \( \Omega^e \) is the surface of the triangle, \( \Gamma^e \) is the boundary line of this surface and \( \hat{a}_n \) is the outward unit normal to the boundary of the surface boundary. Hence, (2.17) can be written in a simpler way:

\[
\iint_{\Omega^e} \left( \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} \right) \, dx \, dy = \oint_{\Gamma^e} \left( A_x \hat{x} + A_y \hat{y} \right) \cdot \hat{a}_n \, dl
\]  \hspace{1cm} (2.18)

Considering

\[
A_x = w a_x \frac{\partial u}{\partial x}
\]  \hspace{1cm} (2.19a)

\[
A_y = w a_y \frac{\partial u}{\partial y}
\]  \hspace{1cm} (2.19b)

and

\[
\hat{a}_n = n_x \hat{x} + n_y \hat{y}
\]  \hspace{1cm} (2.20)

the 1st integral of (2.16) is rewritten as

\[
\iint_{\Omega^e} \left[ \frac{\partial}{\partial x} \left( w a_x \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( w a_y \frac{\partial u}{\partial y} \right) \right] \, dx \, dy = \oint_{\Gamma^e} w \left( a_x \frac{\partial u}{\partial x} n_x + a_y \frac{\partial u}{\partial y} n_y \right) \, dl
\]  \hspace{1cm} (2.21)

Thus, (2.16) yields

\[
- \iint_{\Omega^e} \left[ a_x \frac{\partial w}{\partial x} \frac{\partial u}{\partial x} + a_y \frac{\partial w}{\partial y} \frac{\partial u}{\partial y} \right] \, dx \, dy + \iint_{\Omega^e} \beta w u \, dx \, dy =
\]
\[
\int_{\Omega^e} \omega g \, dx \, dy - \oint_{\Gamma_e} \omega \left( a_x \frac{\partial u}{\partial x} n_x + a_y \frac{\partial u}{\partial y} n_y \right) \, dt \tag{2.22}
\]

As it was done in the previous section, the unknown variable \( u \) can be approximated by the values of this variable on the nodes of the element multiplied by the corresponding basis functions:

\[
u^e = \sum_{j=1}^{n} u_j^e N_j \tag{2.23}
\]

where \( u_j^e \) corresponds to the value of \( u \) in the \( j^{th} \) node of the element \( e \). It is obvious that for triangular elements \( n = 3 \).

Regarding the choice of the weighting function, there are various methods including Point Matching, Least Squares, Subdomain Collocation and Galerkin. In the present analysis we will choose the Galerkin method, which is the most popular in finite element analysis. In the Galerkin method we simply choose the weighting functions \( \omega \) to be the same with the basis functions \( N_i \) introduced before. Thus,

\[
\omega = N_i, \quad \text{for } i = 1: n \tag{2.24}
\]

In order to apply the Galerkin method we insert (2.23) into (2.22) and upon changing the order of the summation and integration, we get

\[
- \int_{\Omega^e} \sum_{i=1}^{n} \beta N_i \sum_{j=1}^{n} u_j^e N_j \, dx \, dy = \int_{\Omega^e} N_i g \, dx \, dy - \oint_{\Gamma_e} N_i \left( a_x \frac{\partial u}{\partial x} n_x + a_y \frac{\partial u}{\partial y} n_y \right) \, dt \quad \text{for } i = 1: n \tag{2.25}
\]

We remark that in the last integral the variable \( u \) was left unchanged on purpose.

The above equation can be written in matrix form in the following way:

\[
\begin{bmatrix}
K_{11}^e & \cdots & K_{1n}^e \\
\vdots & \ddots & \vdots \\
K_{n1}^e & \cdots & K_{nn}^e
\end{bmatrix}
\begin{bmatrix}
u_1^e \\
\vdots \\
u_n^e
\end{bmatrix} =
\begin{bmatrix}
b_1^e \\
\vdots \\
b_n^e
\end{bmatrix}
\tag{2.26}
\]

where

\[
K_{ij}^e = M_{ij}^e + T_{ij}^e = - \int_{\Omega^e} \left( a_x \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + a_y \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} \right) \, dx \, dy + \int_{\Gamma_e} \beta N_i N_j \, dx \, dy \tag{2.27}
\]

\[
b_i^e = f_i^e + p_i^e = \int_{\Omega^e} N_i g \, dx \, dy - \oint_{\Gamma_e} N_i \left( a_x \frac{\partial u}{\partial x} n_x + a_y \frac{\partial u}{\partial y} n_y \right) \, dt \tag{2.28}
\]

Now, we will calculate these integrals for the case of the 2D triangular elements in our meshing with the help of the basis functions given in (2.8) and repeated here for convenience:

\[
N_1(\xi, \eta) = 1 - \xi - \eta \\
N_2(\xi, \eta) = \xi \\
N_3(\xi, \eta) = \eta
\]
We have to take into account that all the integrals are formed in $xy$ coordinates, thus, we have to map them to their simplex form.

It is known from the linear algebra that in order to transform a small surface $dS = dxdy$ to another system of coordinates we have to apply the Jacobian matrix in the following way using (2.2):

\[
dxdy = |J|d\xi d\eta = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{vmatrix} d\xi d\eta = \begin{vmatrix} x^e_2 - x^e_1 & y^e_2 - y^e_1 \\ x^e_3 - x^e_1 & y^e_3 - y^e_1 \end{vmatrix} d\xi d\eta = 2A^e d\xi d\eta \quad (2.29)
\]

Moreover, we need to calculate the terms $\frac{\partial N_i}{\partial x}$ and $\frac{\partial N_i}{\partial y}$ for $i = 1,n$ using the chain derivation:

\[
\frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi} \frac{\partial \xi}{\partial x} + \frac{\partial N_i}{\partial \eta} \frac{\partial \eta}{\partial x} \\
\frac{\partial N_i}{\partial y} = \frac{\partial N_i}{\partial \xi} \frac{\partial \xi}{\partial y} + \frac{\partial N_i}{\partial \eta} \frac{\partial \eta}{\partial y} 
\]

Thus, it is easy to derive the following results using (2.8) and (2.30):

\[
\frac{\partial N_1}{\partial x} = \frac{y^e_2 - y^e_3}{2A^e} \\
\frac{\partial N_1}{\partial y} = \frac{x^e_3 - x^e_2}{2A^e} \\
\frac{\partial N_2}{\partial x} = \frac{y^e_3 - y^e_1}{2A^e} \\
\frac{\partial N_2}{\partial y} = \frac{x^e_1 - x^e_3}{2A^e} \\
\frac{\partial N_3}{\partial x} = \frac{y^e_1 - y^e_2}{2A^e} \\
\frac{\partial N_3}{\partial y} = \frac{x^e_2 - x^e_1}{2A^e} 
\]

Hence, we can now rewrite the integrals $M_{ij}^e$ in $\xi \eta$ coordinates:

\[
M_{11}^e = - \int_0^1 \int_0^{1-\eta} \left( a_x y^e_2 - y^e_3 y^e_2 - y^e_3 \right) \frac{x^e_3 - x^e_2}{2A^e} \frac{x^e_3 - x^e_2}{2A^e} 2A^e d\xi d\eta
\]

\[
= - \left[ a_x \frac{(y^e_2 - y^e_3)^2}{4A^e} + a_y \frac{(x^e_3 - x^e_2)^2}{4A^e} \right] 
\]

\[
M_{12}^e = M_{21}^e = - \left[ a_x \frac{(y^e_2 - y^e_3)(y^e_1 - y^e_2)}{4A^e} \frac{(x^e_3 - x^e_2)(x^e_1 - x^e_2)}{4A^e} \right] 
\]

\[
M_{13}^e = M_{31}^e = - \left[ a_x \frac{(y^e_2 - y^e_3)(y^e_1 - y^e_2)}{4A^e} \frac{(x^e_3 - x^e_2)(x^e_1 - x^e_2)}{4A^e} \right] 
\]

\[
M_{22}^e = - \left[ a_x \frac{(y^e_2 - y^e_3)^2}{4A^e} + a_y \frac{(x^e_3 - x^e_2)^2}{4A^e} \right] 
\]
\[ M_{23} = M_{32} = - \left[ a_x \frac{(y_3^e - y_1^e)(y_1^e - y_2^e)}{4A^e} + a_y \frac{(x_1^e - x_3^e)(x_3^e - x_2^e)}{4A^e} \right] \] (2.33e)

\[ M_{33} = - \left[ a_x \frac{(y_1^e - y_2^e)^2}{4A^e} + a_y \frac{(x_2^e - x_1^e)^2}{4A^e} \right] \] (2.33f)

In the current electrostatic example we want to solve, by comparing (2.11) with (1.2) it is obvious that:

\[ a_x = a_y = 1 \] (2.34)

\[ \beta = g = 0 \] (2.35)

Thus,

\[ T_{ij}^e = 0 \]
\[ f_i^e = 0 \quad \forall i, j \in 1,2,3 \] (2.36)

However, in the general case, given \( \beta \) and \( g \) are constants, it is not difficult to compute these integrals, whose results are given below:

\[ T_{11}^e = T_{22}^e = T_{33}^e = \frac{\beta A^e}{6} \] (2.37a)

\[ T_{12}^e = T_{21}^e = T_{13}^e = T_{31}^e = T_{23}^e = T_{32}^e = \frac{\beta A^e}{12} \] (2.37b)

\[ f_1^e = f_2^e = f_3^e = \frac{g A^e}{3} \] (2.37c)

Regarding the integral \( p_i^e \), we will distinguish 2 cases according to the position of each edge of the triangle that is needed to be handled.

If this edge, let it be the edge \( 1^e_i \rightarrow 2^e_i \), is not lying on a boundary line, then, as it can be seen in Figure 2.5, there is an edge of another triangle \( e_j \), whose edge \( 3^e_j \rightarrow 1^e_j \) is identical to the edge \( 1^e_i \rightarrow 2^e_i \) of the previous triangle. As these nodes are identical by two, \( 1^e_i \equiv 1^e_j \) and \( 2^e_i \equiv 3^e_j \), this means that the value of the unknown quantity in these 2 nodes will be also identical. Moreover, as the same basis functions are used all over the triangles, they will be also identical in these edges. The only difference in the value of the \( p_i^e \) integral between these 2 edges of these triangles will be in the outward normal, as the 2 vectors have the same direction but opposite senses. And as soon as, in the global matrix assembly, these 2 values of integrals will be added, they will vanish to zero. Thus, if none of the edges of the triangle \( e \) we are analyzing is on the boundary, then there is no need to take care about these \( p_i^e \) integral.

![Figure 2.5 Common edge in inner triangles](image-url)
If the edge, let us assume $1^e \to 2^e$, lies on the boundary, then the value of $p_i^e$, depends on the type of the boundary condition that occurs on that part of the boundary. In case of the Dirichlet boundary condition, the unknown variable $u$ has a fixed value and thus the integral

$$p_i^e = -\int_{1^e \to 2^e} N_i \left(a_x \frac{\partial u}{\partial x} n_x + a_y \frac{\partial u}{\partial y} n_y\right) dl$$

(2.38)

will also vanish to zero, hence even in this case we do not need to take it into account.

In case of mixed boundary conditions, which means

$$a_x \frac{\partial u}{\partial x} n_x + a_y \frac{\partial u}{\partial y} n_y + \gamma u = q$$

(2.39)

the $p_i^e$ integral can be written as

$$p_i^e = -\int_{1^e \to 2^e} N_i (q - \gamma u) dl$$

(2.40)

Attention should be paid to the fact that the result of this integral will contain the unknown value $u$ on the nodes that form this certain edge. This means, that the terms that contain $u$ should be moved in the left part of (2.26). But even in this case the integral (2.40) can be computed analytically. Using (2.23), (2.40) can be rewritten:

$$p_i^e = -\int_{1^e \to 2^e} N_i q dl + \int_{1^e \to 2^e} N_i \gamma (u^e_1 N_1 + u^e_2 N_2 + u^e_3 N_3) dl$$

(2.41)

In order to compute $dl$, we recall that for the edge $1^e \to 2^e$, according to Figure 2.4, $\eta = 0$, thus using (2.2) we get:

$$dl = \sqrt{dx^2 + dy^2} = \sqrt{(x_2^e - x_1^e)^2 + (y_2^e - y_1^e)^2} \, d\xi = l_{12} d\xi$$

(2.42)

where $l_{12}$ is the length of the edge $1^e \to 2^e$ in the initial triangle of Figure 2.4. Using (2.8) and for $i = 1:3$, we can write (2.41) as:

$$p_i^e = -\int_0^1 N_1(\xi,0) q l_{12} d\xi + \int_0^1 N_1(\xi,0) \gamma (u^e_1 N_1(\xi,0) + u^e_2 N_2(\xi,0) + u^e_3 N_3(\xi,0)) l_{12} d\xi =$$

$$= -\frac{ql_{12}}{2} + \frac{\gamma l_{12}}{3} u^e_1 + \frac{\gamma l_{12}}{6} u^e_2$$

(2.43a)

$$p_2^e = -\frac{ql_{12}}{2} + \frac{\gamma l_{12}}{6} u^e_1 + \frac{\gamma l_{12}}{3} u^e_2$$

(2.43b)

$$p_3^e = 0$$

(2.43c)

The terms appearing in $p_i^e$ that contain the unknown variables $u_i^e$ should be moved to the left part of the matrix equation (2.26) leading in updated formulas for $K_{ij}$ and $p_i^e$ as follows:
\[ K_{11}^e = K_{11}^p - \frac{\gamma l_{12}}{3} \]
\[ K_{12}^e = K_{12}^p - \frac{\gamma l_{12}}{6} \]
\[ K_{13}^e = K_{13}^p \]
\[ K_{21}^e = K_{21}^p - \frac{\gamma l_{12}}{6} \]
\[ K_{22}^e = K_{22}^p - \frac{\gamma l_{12}}{3} \]
\[ K_{23}^e = K_{23}^p \]
\[ K_{31}^e = K_{31}^p \]
\[ K_{32}^e = K_{32}^p \]
\[ K_{33}^e = K_{33}^p \]

and

\[ p_1^e = -\frac{ql_{12}}{2} \]
\[ p_2^e = -\frac{ql_{12}}{2} \]
\[ p_3^e = 0 \] (2.45)

Regarding the homogenous Neumann boundary condition, from the equation (2.39), it can be seen that

\[ q = \gamma = 0 \] (2.46)

which means that even for this type of boundary condition, the \( p_i^e \) vanishes to zero.

Concluding, by this time we have computed analytically all the integrals of the matrix equation (2.26) for linear basis function and for triangular elements. The code for the case of our Laplace problem for this step can be found in the next section.

### 2.5 Formulation of the global matrix by assembling all the elements

In the previous steps, we formulated a matrix system of equation, 3x3 in the case of triangular meshing, with as unknowns the value of the potential in the 3 nodes of the triangle. Our target now is to map these 3x3 matrix systems, each one corresponding to a different element \( e \), to a global system matrix which describes the whole structure that we want to solve. For this, we could use the enumeration of `initmesh()`, which was described previously in this report, so as to map each local node, that is counted from 1 to 3, to a global enumeration of nodes, where each node in the mesh has a unique identification number starting from 1 to \( N \). In order to make the procedure more clear let us assume that we have a mesh consisting of 6 nodes as it is shown in Figure 2.6.

![Figure 2.6 Example of discretization of the domain.](image-url)
In this case, the matrix $t$ would be

$$
\begin{bmatrix}
K_{11}^{(1)} & K_{13}^{(1)} & 0 & 0 & 0 & K_{12}^{(1)} \\
K_{31}^{(1)} & K_{33}^{(1)} + K_{11}^{(2)} & K_{13}^{(2)} & 0 & 0 & K_{32}^{(1)} + K_{12}^{(2)} \\
0 & K_{31}^{(2)} & K_{33}^{(2)} & 0 & 0 & K_{32}^{(2)} \\
0 & 0 & 0 & 0 & 0 & 0 \\
K_{21}^{(1)} & K_{23}^{(1)} + K_{21}^{(2)} & K_{23}^{(2)} & 0 & 0 & (K_{22}^{(1)} + K_{22}^{(2)}) \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
u_5 \\
u_6
\end{bmatrix} =
\begin{bmatrix}
b_1^{(1)} \\
b_3^{(1)} + b_1^{(2)} \\
b_3^{(1)} + b_1^{(2)} - 10K_{12}^{(1)} \\
b_3^{(1)} + b_1^{(2)} - 10(K_{32}^{(1)} + K_{12}^{(2)}) \\
b_2^{(1)} \\
b_2^{(1)} + b_2^{(2)} - 10(K_{22}^{(1)} + K_{22}^{(2)})
\end{bmatrix}
$$

(2.47)

Table 2.4 Global enumeration of nodes.

<table>
<thead>
<tr>
<th>$e^{th}$ element</th>
<th>$1^{st}$ node</th>
<th>$2^{nd}$ node</th>
<th>$3^{rd}$ node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

and the global matrix for the first 2 elements would be filled in the following way:

However, not all of the values of the nodes $u_i$ are unknown. Actually, the nodes that obey the Dirichlet boundary condition have a known fixed value. For instance, let us assume that the node 6 has a fixed potential of 10 Volts, which means $u_6 = 10$. Thus, the vector of the unknowns $u$ will have now 5 elements and the above matrix should be formed in the following way:

$$
\begin{bmatrix}
K_{11}^{(1)} & K_{13}^{(1)} & 0 & 0 & 0 \\
K_{31}^{(1)} & K_{33}^{(1)} + K_{11}^{(2)} & K_{13}^{(2)} & 0 & 0 \\
0 & K_{31}^{(2)} & K_{33}^{(2)} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
K_{21}^{(1)} & K_{23}^{(1)} + K_{21}^{(2)} & K_{23}^{(2)} & 0 & (K_{22}^{(1)} + K_{22}^{(2)}) \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
u_5
\end{bmatrix} =
\begin{bmatrix}
b_1^{(1)} - 10K_{12}^{(1)} \\
b_3^{(1)} + b_1^{(2)} - 10(K_{32}^{(1)} + K_{12}^{(2)}) \\
b_3^{(1)} - 10K_{32}^{(2)} \\
b_2^{(1)} \\
b_2^{(1)} + b_2^{(2)} - 10(K_{22}^{(1)} + K_{22}^{(2)})
\end{bmatrix}
$$

(2.48)

Of course the matrix is not fully populated, as only the data of the first 2 elements have been processed. However, it becomes clear, that from a computational point of view, the nodes that have a known value $u_i$ should be at the end if the vector $u$, otherwise, except for the vector $b$, also the elements of the matrix $K$ should be rearranged.

The enumeration of the nodes that the function initmesh() provides does not take into account the nodes of which the value is known. Thus, we created during the preprocessing part of this report a new enumeration of the nodes such that the last nodes will be the one that we know their value. Of course in order to move from one enumeration to the other we need 2 mapping arrays, id_new=index(id_old) and id_old=index2(id_new).

An example of a code that performs the tasks described above is given here below:
Nf=max(index); %Number of unknown nodes
Mlocal=zeros(3,3);
K=sparse(zeros(inner_counter,inner_counter));
b=zeros(1,inner_counter);
Ne=size(t,2);

for e=1:Ne
    Ae=0.5*((p(1,t(2,e))-p(1,t(1,e))) * (p(2,t(3,e))-p(2,t(1,e)))) -
    (p(1,t(3,e))-p(1,t(1,e))) * (p(2,t(2,e))-p(2,t(1,e))))
    Mlocal(1,1)=-0.25/Ae*( (p(2,t(2,e))-p(2,t(3,e)))^2 +
    (p(1,t(2,e))-p(1,t(3,e))))^2 );
    Mlocal(2,2)=-0.25/Ae*( (p(2,t(3,e))-p(2,t(1,e)))^2 +
    (p(1,t(1,e))-p(1,t(3,e))))^2 );
    Mlocal(3,3)=-0.25/Ae*( (p(2,t(1,e))-p(2,t(2,e)))^2 +
    (p(1,t(2,e))-p(1,t(1,e))))^2 );
    Mlocal(1,2)=-0.25/Ae*( (p(2,t(2,e))-p(2,t(3,e)))*(p(2,t(3,e))-
    p(2,t(1,e)))*(p(1,t(3,e))-p(1,t(2,e)))*p(1,t(1,e))-
    p(1,t(2,e))));
    Mlocal(2,1)=Mlocal(1,2);
    Mlocal(1,3)=-0.25/Ae*( (p(2,t(2,e))-p(2,t(3,e)))*(p(2,t(1,e))-
    p(2,t(2,e)))*(p(1,t(3,e))-p(1,t(2,e)))*p(1,t(2,e))-
    p(1,t(1,e))));
    Mlocal(3,1)=Mlocal(1,3);
    Mlocal(2,3)=-0.25/Ae*( (p(2,t(3,e))-p(2,t(1,e)))*(p(2,t(1,e))-
    p(2,t(2,e)))*(p(1,t(1,e))-p(1,t(3,e)))*p(1,t(3,e))-
    p(1,t(1,e))));
    Mlocal(3,2)=Mlocal(2,3);
    for i=1:3
        for j=1:3
            if index(t(i,e))>inner_counter
                %Do nothing-action will be taken in the next elseif
            elseif index(t(j,e))>inner_counter
                %If this is Dirichlet node
                b(index(t(i,e)))=b(index(t(i,e)))-
                Mlocal(i,j)*boundary_val(t(j,e));
            else
                %If this is inner node
                K(index(t(i,e)),index(t(j,e)))=K(index(t(i,e)),index(t(j,e)))+Mlocal(i,j);
            end
        end
    end
end
The next task is to inverse the matrix and get the solution for our system of equations. When the dimensions of the matrix to be inversed are not very large, then direct solvers are used. An example of a direct solver is the LU decomposition that is utilized in our example. However, for extremely large matrices iterating methods are chosen.

\begin{verbatim}
b=b';
[L U]=lu(K);
Y=L\b;
X1=U\Y;
X=boundary_val;
for i=1:inner_counter
    X(index2(i))=X1(i);
end
\end{verbatim}

Now the array $X$ contains the values of the potential in all the nodes of the structure, including those that we computed with the FEM and those of which we knew the value due to the Dirichlet boundary condition.

If we want to plot the data we can use the following commands that utilize the plot tools of the Matlab Partial Differential Equations toolbox:

\begin{verbatim}
%using the pdeplot utility for plotting the results
figure(1)
pdeplot(p,e,t,'xydata',X,'mesh','on')
\end{verbatim}

2.6 Postprocessing of the results (Calculation of the capacity)

The problem of solving the Laplace equation in the given structure is solved and we have saved in the vector $X$ the values of the potentials in all the nodes of the mesh. However, many times, the result that is asked is not to compute the unknown variable that governs the PDE, but another quantity which is related to this variable. In our case we choose the capacity of the structure to demonstrate the procedure and in the end we will check our result with a commercial software that performs the same task.

Given the potential in an electrostatic problem, we can derive the vector of the electric field through the following equation:

$$E = -\nabla \varphi$$  \hspace{1cm} (2.49)

The energy stored through the electric field is given by

$$W_e = \frac{1}{2} \iiint_{\Omega} \varepsilon_0 |E|^2 \, dx \, dy$$  \hspace{1cm} (2.50)

Finally, the energy stored in a capacitor is given by:

$$W_e = \frac{1}{2} CV^2$$  \hspace{1cm} (2.51)

where $V$ is the fixed voltage of the one conductor when the second conductor is grounded. From (2.50) and (2.51), and if we divide the surface $\Omega$ into its meshed triangles, we conclude that:
\[ C = \frac{\varepsilon_0 \sum_{e=1}^{N} \iint_{\Omega_e} |E|^2 \, dx \, dy}{V^2} \quad (2.52) \]

The first step is to compute \( E \) and its magnitude inside each triangle. As it is going to be proven, with the specific choice of mesh elements and basis functions, the electric field is a constant vector. From (2.5) we get the potential \( \varphi_e \) inside a triangle given the potential \( \varphi_i \) on its nodes:

\[ \varphi_e = \sum_{i=1}^{3} \varphi_i^e N_i \]

Then,

\[ E^e = -\nabla \varphi^e = -\sum_{i=1}^{3} \varphi_i^e \nabla N_i = -\sum_{i=1}^{3} \varphi_i^e \left( \frac{\partial N_i}{\partial x} \hat{x} + \frac{\partial N_i}{\partial y} \hat{y} \right) \quad (2.53) \]

Taking into account (2.31):

\[
E^e = -\frac{1}{2A^e} \left\{ [(y_2^e - y_3^e)\varphi_1^e + (y_3^e - y_1^e)\varphi_2^e + (y_1^e - y_2^e)\varphi_3^e]\hat{x} \\
+ [(x_3^e - x_2^e)\varphi_1^e + (x_1^e - x_3^e)\varphi_2^e + (x_2^e - x_1^e)\varphi_3^e]\hat{y} \right\} \quad (2.54)
\]

it is obvious that the vector \( E^e \) is constant inside the area of the \( e^{th} \) triangle. Regarding the magnitude, we get easily

\[
|E^e|^2 = \frac{1}{4A^e} \left\{ [(y_2^e - y_3^e)\varphi_1^e + (y_3^e - y_1^e)\varphi_2^e + (y_1^e - y_2^e)\varphi_3^e]^2 \\
+ [(x_3^e - x_2^e)\varphi_1^e + (x_1^e - x_3^e)\varphi_2^e + (x_2^e - x_1^e)\varphi_3^e]^2 \right\} \quad (2.55)
\]

Finally, the expression for the capacity is the following:

\[ C = \frac{\varepsilon_0 \sum_{e=1}^{N} A^e |E^e|^2}{V^2} \quad (2.56) \]

The code that performs the above tasks is displayed below:
2.7 Display of the numerical results

It is common sense that the denser the mesh is, the more accurate the results will be. This is because, the elements are smaller, and thus the perturbation of the unknown variable inside their surface for 2D, or inside their volume for 3D problems can be better approximated with our basis functions. However, if the mesh is very dense, then the number of the nodes will increase a lot, resulting in a global matrix of greater dimension to be inverted. In Table 2.5 are provided some results from this Matlab code for comparison:

<table>
<thead>
<tr>
<th>Number of elements</th>
<th>Calculated capacity (pF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>159</td>
<td>78.243</td>
</tr>
<tr>
<td>636</td>
<td>75.743</td>
</tr>
<tr>
<td>2544</td>
<td>74.797</td>
</tr>
<tr>
<td>10176</td>
<td>74.433</td>
</tr>
<tr>
<td>40704</td>
<td>74.292</td>
</tr>
</tbody>
</table>

Table 2.5 FEM results of the described Matlab code.

Furthermore, the graphical representation of the solution is shown in the Figure 2.7:

```matlab
%Calculation of the capacity
sum=0;
for e=1:Ne
    x1=p(1,t(1,e));
    x2=p(1,t(2,e));
    x3=p(1,t(3,e));
    y1=p(2,t(1,e));
    y2=p(2,t(2,e));
    y3=p(2,t(3,e));
    f1=X(t(1,e));
    f2=X(t(2,e));
    f3=X(t(3,e));
    Elec_field=((y2-y3)*f1+(y3-y1)*f2+(y1-y2)*f3)^2 +
                ((x3-x2)*f1+(x1-x3)*f2+(x2-x1)*f3)^2;
    %Ae is the surface of the eth element
    Ae=abs(0.5*( (p(1,t(2,e))-p(1,t(1,e))) * (p(2,t(3,e))-p(2,t(1,e)))
                - (p(1,t(3,e))-p(1,t(1,e))) * (p(2,t(2,e))-p(2,t(1,e)))))
    sum=sum+Elec_field/(4*Ae);
end
V=100;
eo=8.854*10^(-12);
C=eo*sum/V^2
```
The results above can be checked through a commercial software for electromagnetic applications. In our case we choose Comsol, which also uses the FEM so as to compute the requested results.

In the Comsol interface we choose the AC/DC module electrostatics in 2D and we design the geometry. We will calculate the potential over the surface of our structure and then we will calculate the capacity. The image of the solution of the potential is given in Figure 2.8 which has the same form as the one we computed in Figure 2.7:

In order to compare the solution provided by Comsol with our own solution, we will compare the values of capacity that the 2 programs calculate. Thus, in Table 2.6 the values of the capacity calculated by Comsol are provided:
### Table 2.6
Solution provided by Comsol.

Comparing the Tables 2.6 and 2.7 we can conclude that the calculated values of the capacity differ by around 0.3%. The above result is represented in Figure 2.9.

![Comparison between our Matlab code and Comsol.](image)

**Figure 2.9** Comparison between our Matlab code and Comsol.

#### 2.8 References


3. Method of Moments (MoM)

In the MoM, in comparison with FEM where we calculate the potential $\varphi$ in all the computational domain of our problem, the unknowns that should be computed are the sources of this potential. Obviously, these sources are the surface charges on the boundaries. This means that for a 2D electrostatic problem, the unknown sources are 1D, which reduces the calculation complexity. Moreover, in the calculation formulas used in the MoM, some properties of our space, which can include boundary conditions, are described by an analytical term that is called the Green’s function of the specific problem. In dimensionally small problems or in problems with very complicated geometry there is not great improvement. However, in open and radiation problems where with FEM the whole space should be discretized, in the MoM the reaction of this infinite space is taken into account through the Green’s function. On the other hand, the MoM requires much more analytical preprocessing of the problem, including the calculation of the Green’s function and the formulation of the integral equation, but this is needed only once for a certain problem and then the computation time is usually shorter compared to FEM.

Like FEM, the MoM is also a well-known technique for solving electromagnetic problems and thus there are plenty of references in the bibliography from where somebody can retrieve valuable knowledge and information. Some of the most important considerations for the certain computational method can be found in [1], [2], [3] and [4].

The basic steps for solving the given problem with the MoM are:

- Formulation of integral equation using the 2D free space Green’s function
- Discretization of the problem
- Formulation of the global matrix
- Set of the reference point for potentials
- Computation of the sources and of the capacitance

3.1 Formulation of integral equation using the 2D free space Green’s function

The potential $\varphi$ is given by the Poisson equation:

$$\nabla^2 \varphi(\vec{r}) = \frac{\partial^2 \varphi(\vec{r})}{\partial x^2} + \frac{\partial^2 \varphi(\vec{r})}{\partial x^2} = -\frac{S(\vec{r})}{\varepsilon_0} \quad (3.1)$$

where $S(\vec{r})$ refers to the sources-charges that are present in our problem. As soon as the boundary lines of the specific problem are Perfect Electric Conductors (PEC), there will be a charge distribution $\rho_l$ on the boundaries and only there. Thus $S(\vec{r})$ can be written as:

$$S(\vec{r}) = \rho_l(\vec{r}), \quad \vec{r} \in \partial \Omega \quad (3.2a)$$

$$S(\vec{r}) = 0, \quad \vec{r} \in \Omega \quad (3.2b)$$

(3.1) with (3.2b) is the Laplace equation that was also used in the FEM analysis.

The solution to (3.1), when the source is a delta distribution, provides the Green’s function of the problem under study:

$$\nabla^2 G(\vec{r}, \vec{r'}) = \delta(\vec{r} - \vec{r'}) \quad (3.3)$$

where $\vec{r'}$ is the position of the source and $\vec{r}$ is the position in that the potential is observed.

The solution of (3.3) is provided in many books including [2] and [5] and is:
\[ G(\vec{r}, \vec{r}') = -\frac{1}{2\pi} \ln \frac{|\vec{r} - \vec{r}'|}{|\vec{r}' - \vec{r}_0|} \quad (3.4) \]

In (3.4) \( \vec{r}_0 \) is the position where the potential \( \varphi \) is equal to zero. It has to be kept in mind, as it is going to be shown later, that for a system with more than one charge, \( \vec{r}_0 \) is not arbitrary, but constant for given geometry and charge distribution.

Moreover, \( \vec{r}_0 \) cannot be considered to be of infinity, as then \( G(\vec{r}, \vec{r}') \to \infty \). This is not strange if we consider the following: The 2D problem is actually the cross section of a 3D structure with infinite length in the z direction. That means, that as soon as there is already a fixed boundary potential at infinity we cannot put reference potential also at infinity.

Knowing the Green’s function (3.4) and for a given charge distribution \( S(\vec{r}') \), we can get the imposed potential:

\[ \varphi(\vec{r}) = -\frac{1}{2\pi\varepsilon_0} \int_{\partial\Omega} S(\vec{r}') \ln \frac{|\vec{r} - \vec{r}'|}{|\vec{r}' - \vec{r}_0|} d\vec{r}' \quad (3.5) \]

By taking into account (3.1) and (3.2), (3.5) is written:

\[ \varphi(\vec{r}) = -\frac{1}{2\pi\varepsilon_0} \int_{\partial\Omega} \rho_l(\vec{r}') \ln |\vec{r} - \vec{r}'| d\vec{r}' + \frac{1}{2\pi\varepsilon_0} \int_{\partial\Omega} \rho_l(\vec{r}') \ln |\vec{r}' - \vec{r}_0| d\vec{r}' \quad (3.6) \]

As the last term of (3.6) does not depend on the point of observation \( \vec{r} \), from now on we will refer to it as:

\[ \varphi_0 = \varphi(\vec{r}_0) = \frac{1}{2\pi\varepsilon_0} \int_{\partial\Omega} \rho_l(\vec{r}') \ln |\vec{r}' - \vec{r}_0| d\vec{r}' \quad (3.7) \]

Thus, (3.6) can be written as:

\[ -\frac{1}{2\pi\varepsilon_0} \int_{\partial\Omega} \rho_l(\vec{r}') \ln |\vec{r} - \vec{r}'| d\vec{r}' = \varphi(\vec{r}) - \varphi(\vec{r}_0) \quad (3.8) \]

The unknown to be computed is the charge \( \rho_l(\vec{r}') \) while \( \vec{r}' \) goes over all the boundaries. In order to create a linear system of equations that will provide us with the solution of \( \vec{r}' \), we should discretize the source domain, which are the boundaries \( \partial\Omega \), and the sources \( \rho_l \) themselves.

### 3.2 Discretization of the problem

As there are charges both on the inner and the outer conductor, both conductors will be discretized in \( N \) segments and the \( i^{th} \) segment will have length \( l(i) \). The discretization process in Matlab follows:

```matlab
% MoM 2D electrostatic problem
% Defining the geometry
V1=100; V2=-500; L1=1; L2=2; L3=0.5; L4=0.5;
```
\[ p1 = [-1, 0]; \]
\[ p2 = p1 + [0, L1]; \]
\[ p3 = p1 + [L2, L1]; \]
\[ p4 = p1 + [L2, 0]; \]
\[ p5 = [-0.5, 0.25]; \]
\[ p6 = p5 + [0, L3]; \]
\[ p7 = p5 + [L4, L3]; \]
\[ p8 = p5 + [L4, 0]; \]

\% Discretization
\]
\[ max_length = 0.002; \]
\[ N1 = \text{ceil}(L1 / max_length); \]
\[ l1 = L1 / N1; \]
\[ N1 = N1 + 1; \]
\[ N2 = \text{ceil}(L2 / max_length); \]
\[ l2 = L2 / N2; \]
\[ N2 = N2 + 1; \]
\[ N3 = \text{ceil}(L3 / max_length); \]
\[ l3 = L3 / N3; \]
\[ N3 = N3 + 1; \]
\[ N4 = \text{ceil}(L4 / max_length); \]
\[ l4 = L4 / N4; \]
\[ N4 = N4 + 1; \]

\[ p12 = [1 \times \text{ones}(1, N1); \text{linspace}(p1(1), p2(1), N1); \text{linspace}(p1(2), p2(2), N1)]; \]
\[ p23 = [2 \times \text{ones}(1, N2); \text{linspace}(p2(1), p3(1), N2); \text{linspace}(p2(2), p3(2), N2)]; \]
\[ p34 = [3 \times \text{ones}(1, N1); \text{linspace}(p3(1), p4(1), N1); \text{linspace}(p3(2), p4(2), N1)]; \]
\[ p41 = [4 \times \text{ones}(1, N2); \text{linspace}(p4(1), p1(1), N2); \text{linspace}(p4(2), p1(2), N2)]; \]
\[ p56 = [5 \times \text{ones}(1, N3); \text{linspace}(p5(1), p6(1), N3); \text{linspace}(p5(2), p6(2), N3)]; \]
\[ p67 = [6 \times \text{ones}(1, N4); \text{linspace}(p6(1), p7(1), N4); \text{linspace}(p6(2), p7(2), N4)]; \]
\[ p78 = [7 \times \text{ones}(1, N3); \text{linspace}(p7(1), p8(1), N3); \text{linspace}(p7(2), p8(2), N3)]; \]
\[ p85 = [8 \times \text{ones}(1, N4); \text{linspace}(p8(1), p5(1), N4); \text{linspace}(p8(2), p5(2), N4)]; \]

\[ s12 = [p12(1, 1:N1-1); p12(2, 1:N1-1); p12(2, 2:N1); p12(3, 1:N1-1); p12(3, 2:N1)]; \]
\[ s23 = [p23(1, 1:N2-1); p23(2, 1:N2-1); p23(2, 2:N2); p23(3, 1:N2-1); p23(3, 2:N2)]; \]
\[ s34 = [p34(1, 1:N1-1); p34(2, 1:N1-1); p34(2, 2:N1); p34(3, 1:N1-1); p34(3, 2:N1)]; \]
\[ s41 = [p41(1, 1:N2-1); p41(2, 1:N2-1); p41(2, 2:N2); p41(3, 1:N2-1); p41(3, 2:N2)]; \]
\[ s56 = [p56(1, 1:N3-1); p56(2, 1:N3-1); p56(2, 2:N3); p56(3, 1:N3-1); p56(3, 2:N3)]; \]
\[ s67 = [p67(1, 1:N4-1); p67(2, 1:N4-1); p67(2, 2:N4); p67(3, 1:N4-1); p67(3, 2:N4)]; \]
\[ s78 = [p78(1, 1:N3-1); p78(2, 1:N3-1); p78(2, 2:N3); p78(3, 1:N3-1); p78(3, 2:N3)]; \]
\[ s85 = [p85(1, 1:N4-1); p85(2, 1:N4-1); p85(2, 2:N4); p85(3, 1:N4-1); p85(3, 2:N4)]; \]

\% segments(i, 1) -> segment_id
\% segments(i, 2) -> x1(i)
\% segments(i, 3) -> x2(i)
\% segments(i, 4) -> y1(i)
\% segments(i, 5) -> y2(i)
\]
\[ \text{segments} = \{\text{horzcat}(s12, s23, s34, s41, s56, s67, s78, s85)\}'; \]
In each segment, we will approximate for simplicity the charge density \( \rho_l \) as constant. Then, the total charge distribution \( \rho_l \) will be given by:

\[
\rho_l(\vec{r}) = \sum_{j=1}^{N} q_j f_j(\vec{r}) \tag{3.9}
\]

where

\[
f_j(\vec{r}) = \begin{cases} 1, & \vec{r} \in j\text{th segment} \\ 0, & \text{otherwise} \end{cases} \tag{3.10}
\]

Substituting (3.9) in (3.8) and changing the sequence of summation and integration we get:

\[
-\frac{1}{2\pi \varepsilon_0} \sum_{j=1}^{N} q_j \int_{\vec{r}} f_j(\vec{r}') \ln|\vec{r} - \vec{r}'| \, d\vec{r}' = \varphi(\vec{r}) - \varphi(\vec{r}_0) \tag{3.11}
\]

In order to compute the coefficients \( q_j \), we will use the Galerkin projection that was also used in FEM, which means that as test functions will be chosen the same basis functions \( f_i \), \( i = 1: N \). So, for \( i = 1: N \), we get:

\[
-\frac{1}{2\pi \varepsilon_0} \sum_{j=1}^{N} q_j \int_{\vec{r}_i} \int_{\vec{r}' \in \ell_i} f_i(\vec{r}) f_j(\vec{r}') \ln|\vec{r} - \vec{r}'| \, d\vec{r}' \, d\vec{r} = \int_{\ell_i} f_i(\vec{r}) (\varphi(\vec{r}) - \varphi(\vec{r}_0)) \, d\vec{r} \tag{3.12}
\]

3.3 Formulation of the global matrix

The system of equations (3.12) can be written in matrix formulation as:

\[
-\frac{1}{2\pi \varepsilon_0} [A] \vec{q} = \vec{b} \tag{3.13}
\]

where

\[
A_{ij} = \int_{\ell_i} \int_{\ell_j} f_i(\vec{r}) f_j(\vec{r}') \ln|\vec{r} - \vec{r}'| \, d\vec{r}' \, d\vec{r} \tag{3.14a}
\]

\[
b_i = \int_{\ell_i} f_i(\vec{r}) (\varphi(\vec{r}) - \varphi(\vec{r}_0)) \, d\vec{r} \tag{3.14b}
\]

and

\[
\vec{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_N \end{bmatrix} \tag{3.14c}
\]

As all the segments lie on straight lines, for simplicity, we will assume that each segment belongs to an arbitrary x axis, thus:
\[ A_{ij} = \int_{i}^{j} f_i(x)f_j(x') \ln|x - x'| \, dx' \, dx \]  
\[ (3.15a) \]

and

\[ b_i = \int_{i}^{j} f_i(x)(\varphi(x) - \varphi_0) \, dx \]  
\[ (3.15b) \]

Due to the fact that \( f_i \) is constant in the \( i^{th} \) segment and 0 elsewhere, we can write:

\[ A_{ij} = \int_{i}^{j} \ln|x - x'| \, dx' \, dx \]  
\[ (3.16a) \]

\[ b_i = \int_{i}^{j} (\varphi(x) - \varphi_0) \, dx \]  
\[ (3.16b) \]

Let us assume that the \( i^{th} \) segment is \([x_0, x_1]\). Then, for computing the \( A_{ij} \) elements we will consider 2 different cases, when \( i = j \) and \( i \neq j \).

If \( i = j \), then there is a singularity when \( x = x' \) resulting in the following computations:

\[ A_{ii} = \int_{x=x_0}^{x_1} \int_{x'=x_0}^{x_1} \ln|x - x'| \, dx' \, dx = \int_{x=x_0}^{x_1} \int_{x'=x_0}^{x} \ln(x - x') \, dx' \, dx + \int_{x=x_0}^{x_1} \int_{x'=x}^{x_1} \ln(x' - x) \, dx' \, dx \]

\[ = \int_{x=x_0}^{x_1} [(x - x_0)\ln(x - x_0) + (x_1 - x)\ln(x_1 - x) - x_1 + x_0] \, dx \]  
\[ (3.17) \]

Considering that \( \lim_{\varepsilon \to 0^+} \varepsilon \ln \varepsilon = 0 \) and \( l_i = x_1 - x_0 \) is the length of the \( i^{th} \) segment:

\[ A_{ii} = l_i^2 \ln l_i - \frac{3}{2} \]  
\[ (3.18) \]

If \( i \neq j \), we will consider \( |x - x'| \equiv R_{ij} = \text{const} \) the distance between the centers of the \( i^{th} \) and \( j^{th} \) segments. It is obvious that the farther the distance is between the \( i^{th} \) and the \( j^{th} \) segment, the better the approximation is. Then:

\[ A_{ij} = l_i l_j \ln R_{ij} \]  
\[ (3.19) \]

Finally, the value of the potential \( \varphi_i \) on the \( i^{th} \) segment is known due to the boundary conditions and it is constant, (3.16b) is written as:

\[ b_i = (\varphi_i - \varphi_0)l_i = \begin{cases} (V_1 - \varphi_0)l_i, & \text{if } i^{th} \text{ segment } \in \text{ Conductor 1} \\ (V_2 - \varphi_0)l_i, & \text{if } i^{th} \text{ segment } \in \text{ Conductor 2} \end{cases} \]  
\[ (3.20) \]

So, for \( i = 1: N \), (3.13) is written:

\[ -\frac{1}{2\pi \varepsilon_0} (A_{i1}q_1 + A_{i2}q_2 + \cdots + A_{iN}q_N) = b_i \]  
\[ (3.21) \]
or using (3.18), (3.19) and (3.20):

\[- \frac{1}{2\pi \varepsilon_0} (l_i l_i \ln R_{i1} + \cdots + l_i l_i (\ln l_i - 3/2) + \cdots + l_i l_i \ln R_{iN}) q_i = l_i (\varphi_i - \varphi_0) \]  \hspace{1cm} (3.22)

This means that we can rewrite our matrices as follows:

\[ A_{ii} = l_i (\ln l_i - 3/2) \hspace{1cm} (3.23a) \]
\[ A_{ij} = l_i \ln R_{ij} \hspace{1cm} (3.23b) \]
\[ b_i = \varphi_i - \varphi_0 \hspace{1cm} (3.23c) \]

Matrix A and vector b are computed for the given problem with the following code, ignoring for the time being the reference potential \( \varphi_0 \).

```matlab
%centers(i,1)->x(i)
%centers(i,2)->y(i)
centers=0.5*[(segments(:,2)+segments(:,3)),(segments(:,4)+segments(:,5))];
N=size(segments,1);
%length(i)->l(i)
l=sqrt((segments(:,2)-segments(:,3)).^2+(segments(:,4)-segments(:,5)).^2);
A=zeros(size(segments,1),size(segments,1));
R=zeros(size(segments,1),size(segments,1));
b=zeros(size(segments,1),1);
counter=0;
for i=1:size(segments,1)
    for j=1:size(segments,1)
        if i==j
            %A(j,j)=(l(j))*(l(j))/2;
            A(i,i)=l(i)*(log(l(i))-1.5);
        else
            R(i,j)=sqrt((centers(i,1)-centers(j,1))^2+(centers(i,2)-centers(j,2))^2);
            A(i,j)=l(j)*log(R(i,j));
        end
    end
    if (segments(i,1)==5||segments(i,1)==6||segments(i,1)==7||segments(i,1)==8)
        b(i)=V1;
    else
        counter=counter+1;
        b(i)=V2;
    end
end
```
3.4 Set of the reference point for potentials

If we look back again to (3.13), the matrix $A$ includes information only about the geometry of the structure and it will be always the same regardless the value of $\varphi_0$. Moreover, if we write again (3.13) separating the reference potential $\varphi_0$ and omitting for the time being the term $-\frac{1}{2\pi \varepsilon_0}$:

$$[A]\hat{q} = \begin{bmatrix} V_1 - \varphi_0 \\ V_2 - \varphi_0 \\ \vdots \end{bmatrix}$$  (3.14)

the solution of the charges $\hat{q}$ depends on the reference potential $\varphi_0$, thus it cannot be chosen arbitrary and it has to be regarded as another unknown. This requires another additional equation, which in the case of a two-conductors capacitance is that the total charge is zero:

$$\sum_{j=1}^{N} q_j = 0$$  (3.15)

If we subtract the last equation (the $N^{th}$ row) of the system (3.14) from all the others, we get a new system of $N - 1$ equations and $N$ unknowns, as the unknown $\varphi_0$ is not any more included. By including as $N^{th}$ equation (3.15), the system of equation can be written as:

$$\begin{bmatrix} A_{11} - A_{N1} & A_{12} - A_{N2} & \cdots & A_{1N} - A_{NN} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N-11} - A_{N1} & A_{N-12} - A_{N2} & \cdots & A_{N-1N} - A_{NN} \end{bmatrix} \hat{q} = \begin{bmatrix} V_1 - V_2 \\ \vdots \\ 0 \end{bmatrix}$$  (3.16)

The new matrix system (3.16) is treated in Matlab as following:

```matlab
AsubstractionN=A(N,:);
bsubstractionN=b(N);
Asub=meshgrid(AsubstractionN);
Anew=A-Asub;
bnew=b-bsubstractionN*ones(N,1);
Anew(N,:)=ones(1,N);
```

3.5 Computation of the sources and of the capacitance

The solution of the system (3.16) will provide the charge density over all the metallic boundaries. This can be achieved using many numerical methods, but we should always keep in mind that in comparison to the global matrix in FEM which was sparse, matrix $A$ in MoM is dense.

In order to compute the capacitance of the system, we should calculate the total charge in one conductor and divide with the voltage difference between the 2 conductors:

$$C = \frac{Q|\text{one conductor}}{\Delta V} = \sum_{\forall i|\text{one conductor}} \frac{q_i l_i}{\Delta V}$$  (3.17)

which is calculated with the following part of the Matlab code:
In Table 3.1 there are provided the values of the capacitance for different number of segments.

<table>
<thead>
<tr>
<th>Number of segments</th>
<th>Calculated capacity (pF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>77.265</td>
</tr>
<tr>
<td>160</td>
<td>75.203</td>
</tr>
<tr>
<td>400</td>
<td>74.660</td>
</tr>
<tr>
<td>800</td>
<td>74.448</td>
</tr>
<tr>
<td>1600</td>
<td>74.331</td>
</tr>
</tbody>
</table>

*Table 3.1* MoM results of the described Matlab code.

It is already obvious that a system with fewer unknowns, but with a denser matrix to be inverted is needed to be solved in the case of MoM in order to reach the same accuracy with FEM. In need to be kept into account that in the present demonstration of MoM 0\(^{th}\) order basis functions but in FEM 1\(^{st}\) order basis functions were used. Moreover, the terms (3.19) can be calculated analytically with more accuracy for segments that are not too far away each other.

### 3.6 References


