

Finite Elements Project

Neil Abhra Chowdhury & Enrique Sanchez del Villar, Erik Pillon

March 18, 2018

Abstract

This is the final project of the course **Numerical Methods**, given at Phelma in the Academic Year 2017/2018. In this project we will study a process of material elaboration involving heating by electrode. We'll study the steady state of this process. The objective is to model the physical phenomena which take place in this process with *finite element method*.

All the source code, matlab *m files* and figures can be found on the Github repository <https://github.com/ErikPillon/NumericalMethods>

1 Statement of the Problem

The study configuration is considered cylindrical. A scheme of the study geometry is given in Figure 1: The process is constituted by:

- a cylindrical crucible;
- the elaborated material; the geometry of the study domain occupied by the material is cylindrical;
- 2 electrodes.

The electrodes are in graphite. An electrical potential difference is applied between the top electrode and the bottom electrode included in the crucible: ΔU . This electrical potential difference is *continuous*. An electrical current pass through the material placed in the crucible. We suppose that the contact between the electrodes and elaborated material is perfect. **Joule effect heat the material**. The material of the crucible is an insulating material. The crucible is not model and will be replaced by an adapted boundary condition. Electrical problem has to be solved in the electrodes and in the elaborated material. The heat transfer has to be solved in the material only.

2 Equations of the physical phenomenon and Boundary Conditions

The objective of this part is to present the physical equations of the process in the steady state and boundary conditions. In this process two physical phenomena occur:

- electrical phenomenon;
- thermal phenomenon.

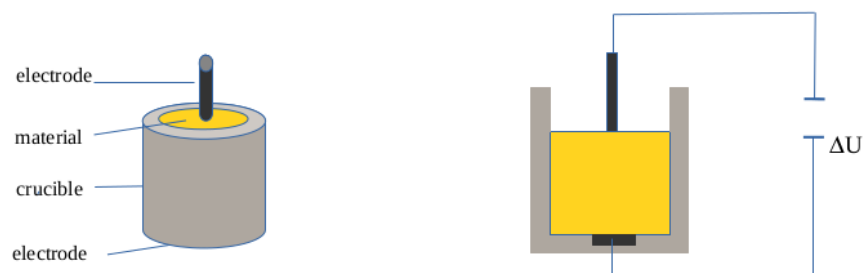


Figure 1: Problem Scheme

2.1 Presentation of the study domain

Describe the study domain. Precise where each phenomenon is solved.

The study domain is the one described in Figure 1, right part.

- We will solve the **thermal problem** only in the material only (yellow part), i.e. where $0.1 < r < 0.3$ and $0 < h < 0.3$.
- We will solve the **electrical problem** in the electrodes and in the elaborated material, i.e. for $0 \geq r < 0.3$ and $0 < h < 0.3$.

2.2 Electrical problem

Give the partial differential equation of the electrical problem. Give the boundary conditions of the electrical problem. Give the expression of the current density and of the Joule power density.

The Electrical problem can be modeled employing the fact that $E = -\nabla U$. We also know that the electrical flux is defined through $\vec{J} \cdot \vec{E}$. Using the fact that the divergence of the Electrical flux is 0 we obtain that:

$$\nabla \cdot (-\sigma \nabla U) = 0.$$

We focus now on the boundary conditions: we know that around the crucible there's insulator material, then we'll have that $\frac{\partial U}{\partial x} = 0$ on the boundary, i.e. $\nabla U \cdot \vec{n} = 0$. We can also take into consideration that ΔU is fixed and so we can write the final system as

$$\begin{cases} \nabla \cdot (-\sigma \nabla U) = 0, & 0 < x < 0.1, 0.02 < z < 0.42 \\ \nabla U \cdot \vec{n} = 0, & x \in \partial V \\ U = \Delta U, & 0 < x < 0.02, z = 0.4 \\ U = 0, & 0 < x < 0.04, z = 0. \end{cases}$$

2.3 Thermal problem

- Give the partial differential equation of the thermal problem.
- Give the boundary conditions of the thermal problem.

A medium through which electric current is conducted may involve the conversion of electrical or chemical energy into heat (or thermal energy), like in our case (i.e., Joule effect). In heat conduction analysis, such conversion processes are characterized as heat (or thermal energy) generation. For example, the temperature of a resistance wire rises rapidly when electric current passes through it as a result of the electrical energy being converted to heat at a rate of $I^2 R$, where I is the current and R is the electrical resistance of the wire. Note that heat generation is a volumetric phenomenon. That is, it occurs throughout the body of a medium. Therefore, the rate of heat generation in a medium is denoted per unit volume. The rate of heat generation in a medium may vary with time as well as position within the medium. When the variation of heat generation with position is known, the total rate of heat generation in a medium of volume V can be determined from

$$Q = \dot{q} = \int_V q_{gen} dV.$$

In our case, where there's no time dependence on the process of heat generation, as in the case of electric resistance heating throughout a homogeneous material, the relation reduces to

$$Q = \int_V q dV.$$

The Fourier Law tells us that:

$$q = -k \nabla T \quad (1)$$

where

q local heat flux density,

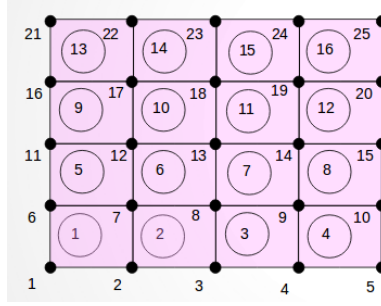


Figure 2: Discretization of the domain.

k material's conductivity,

∇T is the temperature gradient.

From the Gauss-Green theorem we know that

$$\iint_S q(x, y, z) dS = Q$$

where the first integral is all over the surface defined by $0.1 < r < 0.3$ and $0 < h < 0.3$ and Q is the heat generated by the Joule effect.

By the way, employing the divergence theorem, the left hand side of the equation can be rewritten as

$$\iiint \nabla \cdot q(x, y, z) dx dy dz = \iint_S q(x, y, z) dS$$

and then the following identity holds

$$\iiint \nabla \cdot q(x, y, z) dx dy dz = Q. \quad (2)$$

Using Fourier Law (1) and plugging it into the above equation we obtain

$$\nabla \cdot (-k \nabla T) = Q. \quad (3)$$

The final system of equation will be

$$\begin{cases} \nabla \cdot (-k \nabla T) = Q, \\ -k \nabla T \cdot \vec{n} = h(T - T_r) \end{cases} \quad (4)$$

3 Principle of the Modeling

In this project, in a first step, each equation will be developed and test. In a second step, the model with the two coupling equations will be developed and test. For numerical modeling the finite element method is used. In this project, describe the steps of the calculation and the variables used. Take time to define how you will present your numerical results.

4 Numerical Modeling of Heat Transfer Problem with Finite Elements Method

4.1 Study Domain and Mesh

We want to discretize our domain in the same way described in figure 2.

The variable that we are going to use will be the *length and the height of the domain*, as well as the *number of the point* we want in our discretization.

A simple algorithm that takes care of that could be the one in Listing 1, where we have put in input of our function `mesh` the number of points in which we want to discretize our domain `nx, ny`.

Listing 1: Mesh

```

1 function [M,N] = points(lx,ly,nx,ny)
2 % — INPUT —
3 % lx, ly : lenghts
4 % nx, ny : number of points of the discretization
5 % — OUTPUT —
6 % M: matrix of [x,y] components of the point
7 % N: number of points generated
8
9 x=0, y=0; %initialize all the variables to zero
10 i=1; %set counter
11 dy=ly/ny; dx=lx/nx;
12 % use the loops for both x and y
13 for y=0:dy:ly
14     for x=0:dx:lx
15         M(i,:)= [x,y]; %save the point
16         i=i+1; %update the counter
17     end
18 end
19 N = i-1;
20 end

```

We stress the fact that in `for` loop we take care also of the fact that if the point considered is a boundary point we have to come back and restart in *another line*.

The algorithm presented in Listing 2 takes care of numbering in a proper way the elements basis and the points we have generated with the alorithm `points`.

Listing 2: Build relationship for local point and overall point

```

1 function [E,n_Points] = mesh_domain(lx,ly,nx,ny,M)
2 % this is the algorithm for the discretization of the electrical problem
3 % part
4 % — INPUT —
5 % lx, ly : caractéristica lengths of the problem
6 % nx, ny : number of points in each direction
7 % M : Matrix in which are stored the x and y coordinates of the points
8 % — OUTPUT —
9 % E : Matrix in which are stored all the points of each element
10 % e : number of elements
11
12 dx = lx/nx; dy = ly/ny;
13 e = 1;
14 x=0; y=0; % starting point
15 for iter=1:length(M)
16     if M(iter,1)<lx & M(iter,2)<ly
17         E(e,:) = [iter,iter+1,iter+1+nx+1,iter+nx+1];
18         e = e+1;
19     end
20 end
21 n_Points = length(E);

```

In Listing 3 we give an algorithm for the mesh of the boundary elements. Since in the thermal part we have free boundary conditions in all the surface except on the upper surface, in which we have a Neumann boundary condition, we discretize only the upper surface.

Listing 3: Algorithm for boundary elements

```

1 function [b_Ele,N] = b_elements_thermal(lx,ly,nx,ny)
2 % — OUTPUT —
3 % b_Ele: matrix of the boundary elements
4 % N : number of boundary elements generated

```

```

5
6 % for what concerns the thermal problem, we are putting Neumann boundary
7 % conditions on the top edge only, so we focus only on that edge
8 y = ly;
9 dx = lx/nx;
10
11 i = 1+(nx+1)*ny; % we want to start with the top-left point
12 f = 1;
13 for x=0:dx:lx-dx
14     b_Ele(f,:) = [i,i+1];
15     i = i+1;
16     f=f+1;
17 end
18 % total number of boundary elements
19 N = length(b_Ele);

```

4.2 Galerkin's Formulation of Heat Transfer Equation

4.2.1 Projection of the partial differential equation on an element of the basis of the functions α_i

We want to evaluate at this stage the projection of our unknown function T into the element basis β_i , i.e., $\forall i$ we want

$$\iiint_{\Omega} \beta_i \nabla \cdot (-\kappa \nabla T) d\Omega = \iiint_{\Omega} \beta_i Q d\Omega \quad (5)$$

Now we can identify the element basis β_i with the element basis α_i and so we can write

$$\iiint_{\Omega} \alpha_i \nabla \cdot (-\kappa \nabla T) d\Omega = \iiint_{\Omega} \alpha_i Q d\Omega$$

4.2.2 Give the weak formulation of the Galerkin's method. Introduce the boundary conditions in the formulation.

Using the differential identity

$$\nabla \cdot (-\alpha_i \kappa \nabla T) = -\alpha_i \nabla \cdot (\kappa \nabla T) - \kappa \nabla \alpha_i \nabla T$$

we can plug the above equation into (5) obtaining

$$\iiint_{\Omega} \nabla \cdot (-\alpha_i \kappa \nabla T) d\Omega + \iiint_{\Omega} \kappa \nabla \alpha_i \nabla T d\Omega = \iiint_{\Omega} \alpha_i Q d\Omega$$

By the way, we have, thanks to the divergence theorem, that

$$\iiint_{\Omega} \nabla \cdot (-\alpha_i \kappa \nabla T) d\Omega = - \iint_{\Gamma} \alpha_i \kappa \nabla T \cdot \vec{n} d\Gamma.$$

Then finally we have the **weak formulation of the problem** (5)

$$\iiint_{\Omega} \kappa \nabla \alpha_i \nabla T d\Omega - \iint_{\Gamma} \alpha_i \kappa \nabla T \cdot \vec{n} d\Gamma = \iiint_{\Omega} \alpha_i Q d\Omega.$$

Introducing the boundary conditions, see (4), we have

$$\begin{cases} -\kappa \nabla T \cdot \vec{n} = h(T - T_r), & \text{on the free surface} \\ -\kappa \nabla T \cdot \vec{n} = 0, & \text{on all the non-free surface} \end{cases}$$

and then

$$\forall i \quad \iint_{\Gamma} \alpha_i h(T - T_r) d\Gamma + \iiint_{\Omega} \kappa \nabla \alpha_i \nabla T d\Omega = \iiint_{\Omega} \alpha_i Q d\Omega \quad (6)$$

4.2.3 Precise the expression of the elementary volume, the elementary surface for respectively the volume integral and surface integral.

The idea is to transform an integral all over the volume Ω in many sums all over the small elements e . The elements must be disjoint pairwise and the union of the small elements e must be give the original volume Ω . In symbols we have

$$\iiint_{\Omega} [\dots] d\Omega = \sum_e \iiint_{\omega_e} [\dots] d\Omega_e$$

Similarly we'll have that the integral all over the surface is made up by the sum by the sum of all the small surfaces:

$$\iint_{\Gamma} [\dots] d\Gamma = \sum_f \iint_{\gamma_f} [\dots] d\Gamma_f$$

We use the canonical change of variable, from cartesian to cylindrical, i.e.,

$$\begin{aligned} (x, y, z) &\rightarrow (r, \theta, z) \\ (x, y, z) &\mapsto (r \cos(\theta), r \sin(\theta), z) \end{aligned} \quad (7)$$

The change of variable given by Eq.7 gives as determinant of the Jacobian matrix the element r , in such a way that the integration must be performed changing $dx dy dz$ into $r dr d\theta dz$.

Then we have that

$$2\pi \iint_{\Omega} \kappa \nabla \alpha_i \nabla T r dr dz - 2\pi \int_{\Gamma} \alpha_i \kappa \nabla T \cdot \vec{n} dz = 2\pi \iint_{\Omega} \alpha_i Q r dr dz.$$

In particular we will perform this transformation

$$\iint_e f(x, y) dx dy = \int_{-1}^1 \int_{-1}^1 f(x(u, v), y(u, v)) \det(Jac) du dv$$

where

$$\begin{cases} x(u, v) = \sum_{i=1}^N \alpha_i(u, v) \cdot x_i \\ y(u, v) = \sum_{i=1}^N \alpha_i(u, v) \cdot y_i \end{cases} \quad \text{and} \quad \det(Jac) = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{vmatrix} = \begin{vmatrix} \sum_{i=1}^N \frac{\partial \alpha_i(u, v)}{\partial u} \cdot x_i & \sum_{i=1}^N \frac{\partial \alpha_i(u, v)}{\partial u} \cdot y_i \\ \sum_{i=1}^N \frac{\partial \alpha_i(u, v)}{\partial v} \cdot x_i & \sum_{i=1}^N \frac{\partial \alpha_i(u, v)}{\partial v} \cdot y_i \end{vmatrix}$$

and therefore the integral we have to evaluate become a summation with a suitable quadrature formula

$$\iint_e f(x, y) dx dy = \sum_{i \in \{nodes\}} f(x(u_i, y_i), y(u_i, y_i)) \det(J(u_i, y_i)) w_i$$

4.2.4 Give the expression of the integrals on the reference element

The expression for each element of the basis is then

$$\sum_j \iiint_e \nabla \alpha_i \nabla \alpha_j \xi d\xi d\eta.$$

Let's notice that we have

$$T = \sum_{j=1}^N \alpha_j(\xi, \eta, \zeta) \cdot T_j$$

and so the differential becomes

$$\nabla T = \sum_{j=1}^N \nabla \alpha_j(\xi, \eta, \zeta) \cdot T_j$$

where through elementary calculations we can see that

$$\nabla \alpha_j = [JacJ]^{-1} \begin{bmatrix} \frac{\partial \alpha}{\partial \xi} \\ \frac{\partial \alpha}{\partial \eta} \\ \frac{\partial \alpha}{\partial \zeta} \end{bmatrix}.$$

Then the final expression will be

$$\sum_e \int_{\omega_e} \kappa \nabla \alpha_i \cdot \left(\sum_{j=1}^N \nabla \alpha_j T_j \right) \rho(\xi, \eta) d\xi d\eta + \sum_f \int_{\Gamma} \alpha_i h \left(\sum_{j=1}^N \alpha_j T_j \right) \rho(\xi, \eta) d\xi d\eta = \sum_f \int_{\Gamma} \alpha_i h T_r \rho(\xi, \eta) d\eta + \sum_e \int_{\omega_e} \alpha_i Q \rho(\xi, \eta) d\Omega$$

- `jacobian(M, Elem, e)` where `M` is the coordinates matrix and `Elem` is the mesh matrix, while `e` is the global element taken into consideration.
- `alpha(xi,eta)` where `xi,eta` are the local coordinates.
- `update` for updating the global matrix with the 16 values obtained for each element e .
- The functions `points` and `mesh` as already stated in Section 4.1
- The function `Update` for putting the values of the local matrix of each element e (resp. f) in the global matrix A (resp. B).

In Fig. 4 is plotted the 3D mesh of the non coupled solution for the Thermal Problem.

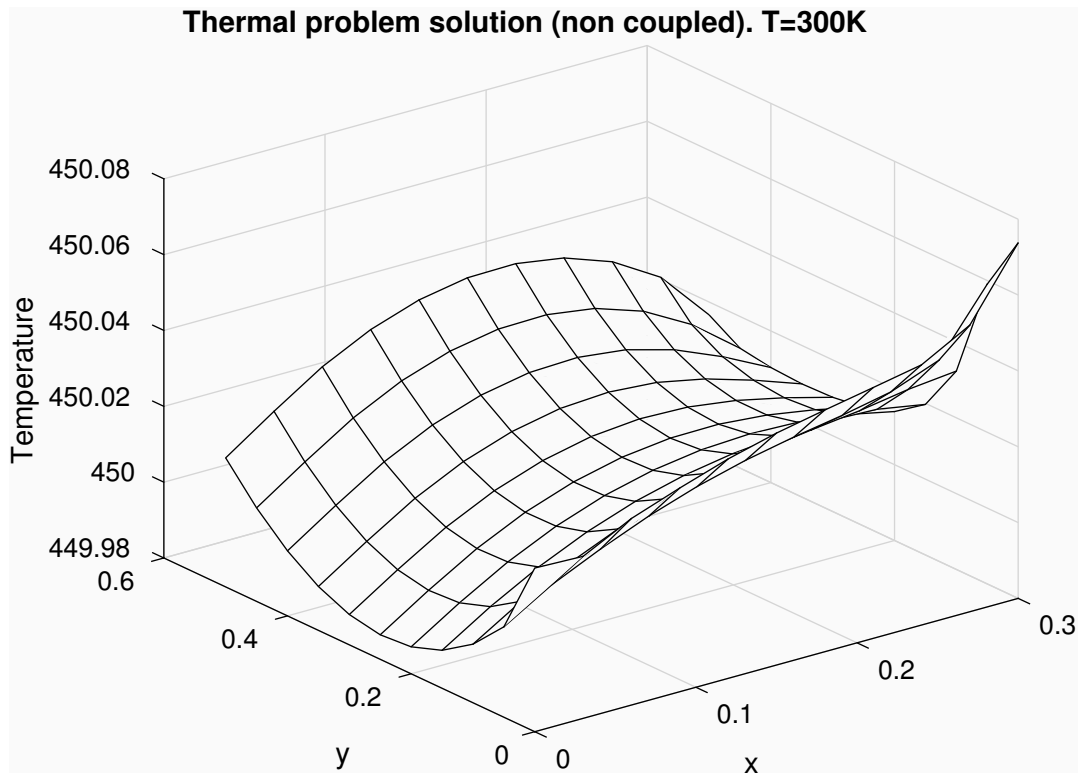


Figure 4: Result for the non-coupled thermal problem

4.4 Programming and calculation

4.4.1 Construct the program: main program and functions

In the following we paste the code used in the resolution of the thermal problem. Listing 4 is the main file used for the resolution. In the middle we've used several different functions, that are listed below.

Listing 4: main program

```

1  clc; clear all; close all;
2  % this is the main file for the thermal part
3
4  %% discretization of the domain and building of the mesh for the thermal part
5  % initialization of the variables lx, ly, nx, ny
6  lx = 0.3; ly = 0.5; nx = 10; ny = 10;
7  dx = lx/nx; dy = ly/ny;
8
9  % creation of the points and of the mesh
10 [M, N_points] = points(lx,ly,nx,ny);
11 [Elem, N_elements] = mesh_domain(lx,ly,nx,ny,M);
12
13 %% initialization of the points for the gauss quadrature formula points

```



```

14 w = [0.347855 0.652145 0.652145 0.347855];
15 u = [-0.861136 -0.339981 0.339981 0.861136];
16
17 %% initialization of the variables
18 % Electrical conductivity constants
19 k1 = 9.93e6; k2 = 2.5e5;
20 % thermal conductivity
21 h = 80,2; % W/(m* K)
22 % temperature of the environment is kept fixed at 300K
23 Tr = 300;
24 % for the non-coupled problem we take Q=1 constant
25 Q = 1;
26 % initialize the rhs
27 % at the end the system to solve will be (A+B)*T=rhs
28 rhs = zeros(N_points,1);
29
30 %% creation of the big matrices
31 % this matrix has to be filled with all the terms of interactions between
32 % the internal elements
33 Interaction = zeros(N_points);
34
35 %% evaluation of the integral on the various elements of the basis
36 for e = 1:N_elements
37     % creation of the Jacobian matrix
38     [Jac] = jacobian(M, Elem, e);
39     J = det(Jac); %jacobian
40     %% evaluation of the integral
41     % evaluation of the 4x4 matrix gai_gaj
42     % here we will need the gauss points
43     gai_gaj = double_integral(M, Elem, e, dx, u, w);
44     Contribution = h*gai_gaj*J;
45     % update the global matrix values; it will recollect the global number
46     % from the local number and the element e
47     Interaction = Update(Interaction, Elem, e, Contribution);
48     % we have to take into account the heat generated by the electric
49     % problem
50     A = heat_contribution(M, Elem, e, Q, dx, u);
51     Contribution = A*J;
52     rhs = Update_rhs(rhs, Elem, e, Contribution);
53 end
54
55 % from the original discretization we extract the boundary elements
56 [b_Elem, N_b_elements] = b_elements_thermal(lx,ly,nx,ny);
57 % this matrix must be filled with the terms given by the boundary effects
58 Bound = zeros(N_points);
59 for f = 1:N_b_elements
60     [Jac] = jacobian(M, b_Elem, f);
61     J = det(Jac);
62     %% evaluation of the 1 dimensional integral
63     % evaluation of the 2x2 matrix ai_aj
64     % here we will need the gauss points
65     ai_aj = integral_1D(M, b_Elem, f, dx, u, w);
66     Contribution_boundary = h*ai_aj*J;
67     % update the global matrix values; it will recollect the global number
68     % from the local number and the element e
69     Bound = Update_boundary(Bound, b_Elem, f, Contribution_boundary);
70 end
71
72 %% construction of the right hand side of the equation
73 % the for loop is for the Neumann condition
74 for f = 1:N_b_elements

```

```

75     [Jac] = jacobian(M, b_Elem, f);
76     J = det(Jac);
77     %% evaluation of the 1 dimensional integral
78     [ai] = neumann_condition(M, b_Elem, f, Tr, dx, u, w);
79     Contribution_boundary = h*ai*J;
80     % update the global matrix values; it will recollect the global number
81     % from the local number and the element e
82     rhs = Update_rhs(rhs, b_Elem, f, Contribution_boundary);
83 end
84
85 T = (Interaction+Bound)\rhs;
86
87 figure
88 [x,y] = meshgrid(0:dx:lx,0:dy:ly)
89 mesh(x,y,reshape(T,11,11))
90 title('Thermal problem solution (non coupled). T=300K')
91 xlabel('x')
92 ylabel('y')
93 zlabel('Temperature')
94 filename = 'Temperature_result';
95 print(filename, '-deps')

```

Where we used the function jacobian, alpha, update and dIntegral listed below.

Listing 5: function for the creation of the jacobian matrix for each element

```

1 function [Jac] = jacobian(M, E, e)
2 % ----- INPUT -----
3 % M: matrix of the points with x and y coordinates
4 % E: matrix of the mesh
5 % e: global number of the element considered
6 %
7 % ----- OUTPUT -----
8 % Jac: 2x2 matrix with the values of the jacobian matrix
9
10 % we obtain the x and y coordinates of the points involved
11 x_i = M(E(e,:),1);
12 y_i = M(E(e,:),2);
13 % xi and eta points in which evaluate the functions
14 xi = [-1, 1, 1, -1];
15 eta = [-1, -1, 1, 1];
16 % n_elements = number of nodes considered
17 % [it can change from a boundary element or an internal
18 % element]
19 n_elements = length(x_i);
20
21 Jac = zeros(2);
22 for i = 1:n_elements
23     [alfa, dalfa_xi, dalfa_eta] = alpha(xi(i),eta(i));
24     Jac(1,1) = Jac(1,1) +dalfa_xi(i)*x_i(i);
25     Jac(1,2) = Jac(1,1) +dalfa_xi(i)*y_i(i);
26     Jac(2,1) = Jac(1,1) +dalfa_eta(i)*x_i(i);
27     Jac(2,2) = Jac(1,1) +dalfa_eta(i)*y_i(i);
28 end

```

Listing 6: Function for the creation of the alpha functions and the derivatives of the latter

```

1 function [alfa, dalfa_xi, dalfa_eta] = alpha(xi,eta)
2 % ----- INPUT -----
3 % xi, eta: nodes in which the function must be evaluated
4 % - they can be also vector values
5 %

```

```

6 % ----- OUTPUT -----
7 % alfa:      4x1 vector with the values of the four alfas evaluated at (xi,eta)
8 % dalfa_xi:  4x1 vector of the partial derivatives of alfas wrt xi evaluated
9 %            at (xi,eta)
10 % dalfa_eta: 4x1 vector of the partial derivatives of alfas wrt eta evaluated
11 %            at (xi,eta)
12
13 % alfa functions
14 alfa(1,:) = 1/4*(1-xi).*(1-eta);
15 alfa(2,:) = 1/4*(1+xi).*(1-eta);
16 alfa(3,:) = 1/4*(1+xi).*(1+eta);
17 alfa(4,:) = 1/4*(1-xi).*(1+eta);
18
19 % evaluation of the derivative respect to u
20 dalfa_xi(1,:) = -1/4.*(1-eta);
21 dalfa_xi(2,:) = 1/4.*(1-eta);
22 dalfa_xi(3,:) = 1/4.*(1+eta);
23 dalfa_xi(4,:) = -1/4.*(1+eta);
24
25 % evaluation of the derivative respect to v
26 dalfa_eta(1,:) = -1/4.*(1-xi);
27 dalfa_eta(2,:) = -1/4.*(1+xi);
28 dalfa_eta(3,:) = 1/4.*(1+xi);
29 dalfa_eta(4,:) = 1/4.*(1-xi);
30 end

```

Listing 7: Function for update the global matrix with the values obtained for the local element e

```

1 function [M] = Update(M, Elements_matrix, e, A)
2 % ----- INPUT -----
3 % Interaction      : matrix to be modified
4 % Elements_matrix: matrix of the mesh
5 % e                : number of the global element
6 % A                : matrix of the local contributions
7 %
8 % ----- OUTPUT -----
9 % M: Matrix modified
10
11 if size(A) ~= [4,4]
12     error('Local Matrix dimension are not 4x4')
13     M = zeros(4);
14     return
15 end
16
17 for i=1:4
18     for j=1:4
19         M(Elements_matrix(e,i),Elements_matrix(e,j)) = M(Elements_matrix(e,i),
20             Elements_matrix(e,j))+A(i,j);
21     end
22 end

```

Listing 8: Function for evaluating the double integral $\iint \nabla \alpha_i \nabla \alpha_j$

```

1 function [A] = double_integral(M, Elem, e, dx, u, w)
2
3 A = zeros(4);
4 r = M(Elem(e,1),1);
5
6 for i = 1:4
7     for j = 1:4
8         sum = 0;

```

```

9      for xi_coord = 1:4
10         for eta_coord = 1:4
11             [Alpha, dalpha_xi, dalpha_eta] = alpha(u(xi_coord),u(eta_coord));
12             % the following is the summation of the inner product
13             % grad(alpha_i).grad(alpha_j)
14             w1 = w(xi_coord);
15             w2 = w(eta_coord);
16             sum=sum+(dalpha_xi(i)*dalpha_xi(j)+dalpha_eta(i)*dalpha_eta(j))*(r+dx/2+u
                (xi_coord)*dx)*w1*w2;
17         end
18     end
19     A(i,j) = sum;
20 end
21 end
22
23 end

```

5 Model the Electrical Problem with the Finite Elements Method

5.1 Study Domain and Mesh

Since the domain of interest of the electric problem is larger than the thermal problem, we have to slightly modify our algorithm. We then have the algorithm in Listing 9, where we have put in input of our function `mesh_electric` the number of points in which we want to discretize our domain.

Listing 9: Points

```

1  function [M,N] = points_electric()
2  % ----- OUTPUT -----
3  % M: matrix of [x,y] components of the point
4  % N: number of points generated
5
6  % discretization for the electric part
7  dx=0.01; dy=0.01;
8  N = 1; % counter
9
10 % bottom electrode
11 y = 0;
12 while y <= 0.02
13     x = 0;
14     while x <= 0.04
15         M(N,:) = [x,y];
16         x = x+dx;
17         N = N+1;
18     end
19     y = y+dy;
20 end
21
22 % material
23 while y <= 0.4
24     x = 0;
25     while x <= 0.1
26         M(N,:) = [x,y];
27         x = x+dx;
28         N = N+1;
29     end
30     y = y+dy;
31 end
32
33 % top electrode
34 while y <= 0.1

```

```

35     x = 0;
36     while x <= 0.02
37         M(N,:) = [x,y];
38         x = x+dx;
39         N = N+1;
40     end
41     y = y+dy;
42 end

```

Listing 10: Build relationship for local point and overall point

```

1 function [E,e] = mesh_electric(M)
2 % ----- INPUT -----
3 % M : Matrix in which are stored the x and y coordinates of the points
4 % ----- OUTPUT -----
5 % E : Matrix in which are stored all the points of each element
6 % e : number of elements
7
8 dx = 0.01; dy = 0.01;
9 e = 1;
10 y=0;
11 while y <= 0.02
12     x = 0;
13     while x <= 0.04
14         nx = 0.04/dx;
15         E(e,:) = [e,e+1,e+1+nx,e+nx];
16         x = x+dx;
17         e = e+1;
18     end
19     y = y+dy;
20 end
21
22 % material
23 while y <= 0.4
24     x = 0;
25     while x <= 0.1
26         nx = 0.1/dx;
27         E(e,:) = [e,e+1,e+1+nx,e+nx];
28         x = x+dx;
29         e = e+1;
30     end
31     y = y+dy;
32 end
33
34 % top electrode
35 while y <= 0.1
36     x = 0;
37     while x <= 0.02
38         nx = 0.02/dx;
39         E(e,:) = [e,e+1,e+1+nx,e+nx];
40         x = x+dx;
41         e = e+1;
42     end
43     y = y+dy;
44 end
45
46 end

```

5.2 Galerkin's Formulation of the Electrical Equation

The projection of the partial differential equation on a basis element is

$$\iiint_{\omega} \alpha_i \nabla \cdot (-\sigma \nabla U) d\Omega = 0, \quad \forall i$$

and then the **weak formulation for the electric part** becomes

$$\iiint_{\omega} \sigma \alpha_i \cdot \nabla U d\Omega = 0, \quad \forall i. \quad (8)$$

The boundary conditions given can be modeled in the following fashion

$$\begin{cases} U = 0, & \text{on the top of the crucible,} \\ U = U_0, & \text{on the bottom of the crucible,} \\ J \cdot n = 0, & \text{on all the other surfaces.} \end{cases}$$

As before, the elementary element of surface and volume are given by the canonical change of variables, from cartesian to cylindrical, i.e.,

$$\begin{aligned} (x, y, z) &\rightarrow (r, \theta, z) \\ (x, y, z) &\mapsto (r \cos(\theta), r \sin(\theta), z) \end{aligned} \quad (9)$$

The change of variable given by Eq.9 gives as determinant of the Jacobian matrix the element r , in such a way that the integration must be performed changing $dx dy dz$ into $r dr d\theta dz$.

The second change of variables will be

$$\begin{aligned} (r, \theta, z) &\rightarrow (\xi, \eta) \\ (r, \theta, z) &\mapsto (\xi(r, z), \eta(r, z)) \end{aligned} \quad (10)$$

and we will have $dx dy dz = \det(Jac) d\xi d\eta$.

5.3 Algorithm of the Finite Element Formulation of the Electrical Equation

The algorithm goes as follows:

- Discretize the domain and build the mesh.
- For each element e of the domain:
 - Construct the Jacobian matrix Jac with the functions α_i
 - Evaluate the Jacobian $\det(Jac)$
 - Evaluate all the 16 terms $\nabla \alpha_i \nabla \alpha_j$ with the help of the Gauss quadrature formula.
 - Update the global matrix A with the function `Update`
- For each element f of the upper boundary:
 - Construct the Jacobian Jac (that is already a scalar value) with the functions α_1 and α_2
 - Evaluate all the 4 terms $\alpha_i \alpha_j$ with the help of the 1D Gauss quadrature formula.
 - Update the global matrix B with the function `Update` modified for the boundary elements
- Solve the linear problem $(A_{ij} + B_{ij})T_i = Q_i$

In the program we make use of the following functions:

- `jacobian(M, Elem, e)` where M is the coordinates matrix and $Elem$ is the mesh matrix, while e is the global element taken into consideration.
- `alpha(xi, eta)` where xi, eta are the local coordinates.
- `update` for updating the global matrix with the 16 values obtained for each element e .
- The functions `points_electric` and `mesh` as already stated in Section 4.1
- The function `Update_electric` for putting the values of the local matrix of each element e (resp. f) in the global matrix A (resp. B).

5.4 Programming and calculation

5.4.1 Construct the program: main program and functions

Listing 11: main program

```
1 clc; clear all; close all;
2 % this is the main file for the electric part
3
4 %% discretization of the domain and building of the mesh for the thermal part
5 % initialization of the variables lx, ly, nx, ny
6 dx = 0.01; dy = 0.01;
7
8 % creation of the points and of the mesh
9 [M, N_points] = points_electric();
10 [Elem, N_elements] = mesh_electric(M);
11 [b_Elem, N_b_elements] = b_elements_electric(lx,ly,nx,ny);
12
13 %% initialization of the points for the gauss quadrature formula
14 w=[0.347855 0.652145 0.652145 0.347855];
15 u=[-0.861136 -0.339981 0.339981 0.861136];
16
17 %% initialization of the variables
18 k = 1;
19 sigma = 1;
20
21 %% creation of the big matrices
22 % this matrix has to be filled with all the terms of interactions between
23 % the internal elements
24 Interaction = zeros(N_points);
25
26 %% evaluation of the integral on the variuos elements of the basis
27 for e = 1:N_elements
28     % creation of the Jacobian matrix
29     [Jac] = jacobian(M, Elem, e);
30     J = det(Jac); %jacobian
31
32     %% evaluation of the integral
33     % evaluation of the 4x4 matrix gai_gaj
34     % here we will need the gauss points
35     gai_gaj = double_integral(M, Elem, e, dx, u, w);
36     Contribution = sigma*gai_gaj*J;
37     % update the global matrix values; it will recollect the global number
38     % from the local number and the element e
39     Interaction = Update(Interaction, Elem, e, Contribution);
40 end
41
42 % this matrix must bel filled with the terms given by the boundary effects
43 Bound = zeros(N_points);
44 for f = 1:N_b_elements
45     [Jac] = jacobian(M, Elem, e);
46     J = det(Jac);
47
48     %% evaluation of the 1 dimensional integral
49     % evaluation of the 2x2 matrix ai_aj
50     % here we will need the gauss points
51     ai_aj = integral_1D(M, b_Elem, f, dx, u, w);
52     Contribution_boundary = k*ai_aj*J;
53     % update the global matrix values; it will recollect the global number
54     % from the local number and the element e
55     Bound = Update_boundary(Bound, b_Elem, f, Contribution_boundary);
56 end
57 % for testing we use rhs random
```

```

57 rhs = 1000*rand(N_points,1);
58 T = (Interaction+Bound)\rhs;

```

Where we used the function jacobian and alpha listed below.

Listing 12: Function for the creation of the jacobian matrix for each element

```

1 function [Jac] = jacobian(M, E, e)
2 % ----- INPUT -----
3 % M: matrix of the points with x and y coordinates
4 % E: matrix of the mesh
5 % e: global number of the element considered
6 %
7 % ----- OUTPUT -----
8 % Jac: 2x2 matrix with the values of the jacobian matrix
9
10 % we obtain the x and y coordinates of the points involved
11 x_i = M(E(e,:),1);
12 y_i = M(E(e,:),2);
13 % xi and eta points in which evaluate the functions
14 xi = [-1, 1, 1, -1];
15 eta = [-1, -1, 1, 1];
16 % n_elements = number of nodes considered
17 % [it can change from a boundary element or an internal
18 % element]
19 n_elements = length(x_i);
20
21 Jac = zeros(2);
22 for i = 1:n_elements
23     [alfa, dalfa_xi, dalfa_eta] = alpha(xi(i),eta(i));
24     Jac(1,1) = Jac(1,1) +dalfa_xi(i)*x_i(i);
25     Jac(1,2) = Jac(1,1) +dalfa_xi(i)*y_i(i);
26     Jac(2,1) = Jac(1,1) +dalfa_eta(i)*x_i(i);
27     Jac(2,2) = Jac(1,1) +dalfa_eta(i)*y_i(i);
28 end

```

Listing 13: Function for the creation of the alpha functions and the derivatives of the latter

```

1 function [alfa, dalfa_xi, dalfa_eta] = alpha(xi,eta)
2 % ----- INPUT -----
3 % xi, eta: nodes in which the function must be evaluated
4 % - they can be also vector values
5 %
6 % ----- OUTPUT -----
7 % alfa: 4x1 vector with the values of the four alfas evaluated at (xi,eta)
8 % dalfa_xi: 4x1 vector of the partial derivatives of alfas wrt xi evaluated
9 % at (xi,eta)
10 % dalfa_eta: 4x1 vector of the partial derivatives of alfas wrt eta evaluated
11 % at (xi,eta)
12
13 % alfa functions
14 alfa(1,:) = 1/4*(1-xi).*(1-eta);
15 alfa(2,:) = 1/4*(1+xi).*(1-eta);
16 alfa(3,:) = 1/4*(1+xi).*(1+eta);
17 alfa(4,:) = 1/4*(1-xi).*(1+eta);
18
19 % evaluation of the derivative respect to u
20 dalfa_xi(1,:) = -1/4.*(1-eta);
21 dalfa_xi(2,:) = 1/4.*(1-eta);
22 dalfa_xi(3,:) = 1/4.*(1+eta);
23 dalfa_xi(4,:) = -1/4.*(1+eta);
24

```



```

25 % evaluation of the derivative respect to v
26 dalfa_eta(1,:) = -1/4.*(1-xi);
27 dalfa_eta(2,:) = -1/4.*(1+xi);
28 dalfa_eta(3,:) = 1/4.*(1+xi);
29 dalfa_eta(4,:) = 1/4.*(1-xi);
30 end

```

Listing 14: Function for update the global matrix with the values obtained for the local element e

```

1 function [M] = Update(M, Elements_matrix, e, A)
2 % ----- INPUT -----
3 % Interaction      : matrix to be modified
4 % Elements_matrix: matrix of the mesh
5 % e                : number of the global element
6 % A                : matrix of the local contributions
7 %
8 % ----- OUTPUT -----
9 % M: Matrix modified
10
11 if size(A) ~= [4,4]
12     error('Local Matrix dimension are not 4x4')
13     M = zeros(4);
14     return
15 end
16
17 for i=1:4
18     for j=1:4
19         M(Elements_matrix(e,i),Elements_matrix(e,j)) = M(Elements_matrix(e,i),
20             Elements_matrix(e,j))+A(i,j);
21     end
22 end
23

```

Listing 15: Function for evaluating the double integral $\iint \nabla \alpha_i \nabla \alpha_j$

```

1 function [A] = double_integral(M, Elem, e, dx, u, w)
2
3 A = zeros(4);
4 r = M(Elem(e,1),1);
5
6 for i = 1:4
7     for j = 1:4
8         sum = 0;
9         for xi_coord = 1:4
10             for eta_coord = 1:4
11                 [Alpha, dalpha_xi, dalpha_eta] = alpha(u(xi_coord),u(eta_coord));
12                 % the following is the summation of the inner product
13                 % grad(alpha_i).grad(alpha_j)
14                 w1 = w(xi_coord);
15                 w2 = w(eta_coord);
16                 sum=sum+(dalpha_xi(i)*dalpha_xi(j)+dalpha_eta(i)*dalpha_eta(j))*(r+dx/2+u
17                     (xi_coord)*dx)*w1*w2;
18             end
19         end
20         A(i,j) = sum;
21     end
22 end
23

```