Finite Elements Project

Neil Abhra Chowdhury & Enrique Sanchez del Villar, Erik Pillon

March 19, 2018

Abstract

This is the final project of the course **Numerical Methods**, given at Phelma in the Academic Year 2017/2018, Bachelor Defree in Nuclear Engineering. 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 the *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 now 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^2R, 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:

$$q = -k \nabla T \tag{1}$$

where

q local heat flux density,

k material's conductivity,

∇T is the temperature gradient.

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

$$\nabla \cdot (-k\nabla T) = Q. \tag{2}$$

The final system of equation will be

$$\begin{cases} \nabla \cdot (-k\nabla T) = Q, & \\ -\kappa \nabla T \cdot \vec{n} = h(T - T_r), & \text{on the upper surface,} \\ -\kappa \nabla T \cdot \vec{n} = 0, & \text{on the non free surfaces.} \end{cases} \tag{3}$$

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`.

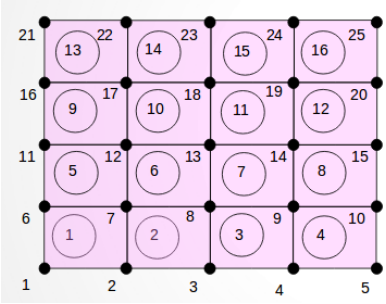

Figure 2: Discretization of the domain

Listing 1: Mesh

```
function [M,N] = points(lx,ly,nx,ny)
% ----   INPUT  ----
% lx, ly : lenghts
% nx, ny : number of points of the discretization
% ------   OUTPUT  ----------
% M: matrix of [x,y] components of the point
% N: number of points generated

x=0, y=0; %initialize all the variables to zero
i=1; %set counter
dy=ly/ny; dx=lx/nx;
% use the loops for both x and y
for y=0:dy:ly
    for x=0:dx:lx
        M(i,:)= [x,y]; %save the point
        i=i+1; %update the counter
    end
end
N = i-1;
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

```
function [E,n_Points] = mesh_domain(lx,ly,nx,ny,M)
% this is the algorithm for the discretization of the electrical problem
% part
% ------   INPUT   ----------
% lx, ly : characteristica lengths of the problem
% nx, ny : number of points in each direction
% M      : Matrix in which are stored the x and y coordinates of the points
% ------   OUTPUT   ---------
% E : Matrix in which are stored all the points of each element
% e : number of elements

dx = lx/nx; dy = ly/ny;
e = 1;
x=0; y=0; % starting point
for iter=1:length(M)
    if M(iter,1)<lx & M(iter,2)<ly
        E(e,:) = [iter,iter+1,iter+1+nx+1,iter+nx+1];
        e = e+1;
    end
end
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

```
function [b_Ele,N] = b_elements_thermal(lx,ly,nx,ny)
% -------   OUTPUT     ----------
% b_Ele: matrix of the boundary elements
% N    : number of boundary elements generated

% for what concerns the thermal problem, we are putting Neumann boundary
% conditions on the top edge only, so we focus only on that edge
y = ly;
dx = lx/nx;

i = 1+(nx+1)*ny; % we want to start with the top-left point
f = 1;
for x=0:dx:lx-dx
    b_Ele(f,:) = [i,i+1];
    i = i+1;
    f=f+1;
end
% total number of boundary elements
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

$$\iiint_{\Omega} \beta_i \nabla \cdot (-\kappa \nabla T)\, d\Omega = \iiint_{\Omega} \beta_i Q\, d\Omega \tag{4}$$

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 (4) 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** (4)

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

Introducing the boundary conditions, see (3), 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

$$\forall i \iint_\Gamma \alpha_i h(T - T_r)\, d\Gamma + \iiint_\Omega \kappa \nabla \alpha_i \nabla T\, d\Omega = \iiint_\Omega \alpha_i Q\, d\Omega \tag{5}$$

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 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.,

$$\begin{aligned} &(x, y, z) \to (r, \theta, z) \\ &(x, y, z) \mapsto (rcos(\theta), rsin(\theta), z) \end{aligned} \tag{6}$$

The change of variable given by Eq.6 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

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

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

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

Then the final expression on the reference element will be:

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

Boundary conditions have to be added to this equation.

4.2.5 Detail the expression of the elementary matrix on an element e. Precise the size of each elementary matrix (sub matrix). Precise the expression of each integral and the principle of calculation. For each integral, precise the nature of the element e

The elementary matrix of an element e is given by the inner product $\nabla\alpha_i\nabla\alpha_j$. In particular we'll have a matrix A_{ij}^e in which we will have in position (i,j) the element $\nabla\alpha_i\nabla\alpha_j$. The matrix will be evaluated through a 2-dimensional quadrature formula; in 1-dimension quadrature formula allow us to pass from a continuous integral to a discrete sum of values. First of all we have to rescale the integral extrema to 1 and -1,

$$\int_a^b f(x)\,dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right) dx.$$

Then the discretization will become

$$\frac{b-a}{2} \sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right),$$

where suitable values x_i and weights w_i will be taken.

The matrix A_{ij}^e is a 4×4 matrix, for each inner element. For the outer element it will be a 2×2 matrix.

For the elements 22, 23, 24, 25 of Figure 3 we have also to develop the integral on the boundary element f

$$\sum_f \int_\Gamma \alpha_i h \left(\sum_{j=1}^{N} \alpha_j T_j\right) \rho(\xi,\eta)\,d\xi\,d\eta$$

The resulting local matrix will be a 2x2 matrix, given by the product $\alpha_i \cdot \alpha_j$.

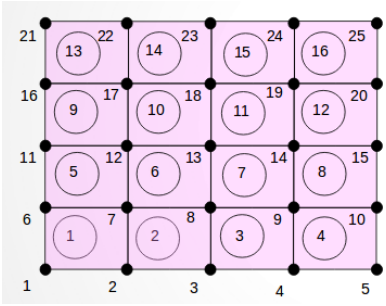

Figure 3: The boundary elements will have 3 boundaries or 2, depending on which one we are considering.

4.3 Algorithm of the Finite Element Formulation of Heat Transfer Equation

The algorithm goes as follows:

- Discretize the domain and build the mesh.
- Initialize the variables like heat conductivity, electrical conductivity, room Temperature, the Gauss nodes.
- 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 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.
  - Add the source term for the nearest neighbor.
  - Update the global matrix B with the function `Update` modified for the boundary elements
- Solve the linear problem $(A_{ij} + B_{ij})T_i = \text{rhs}$

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` and `mesh` as already stated in Section 4.1
- `heat_contribution`
- `integral_1D`
- The function `Update` for putting the values of the local matrix of each element e (resp. f) in the global matrix A (resp. B).
- `Update_rhs`
- `neumann_condition` for evaluating the contribution given only on the upper free surface.

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

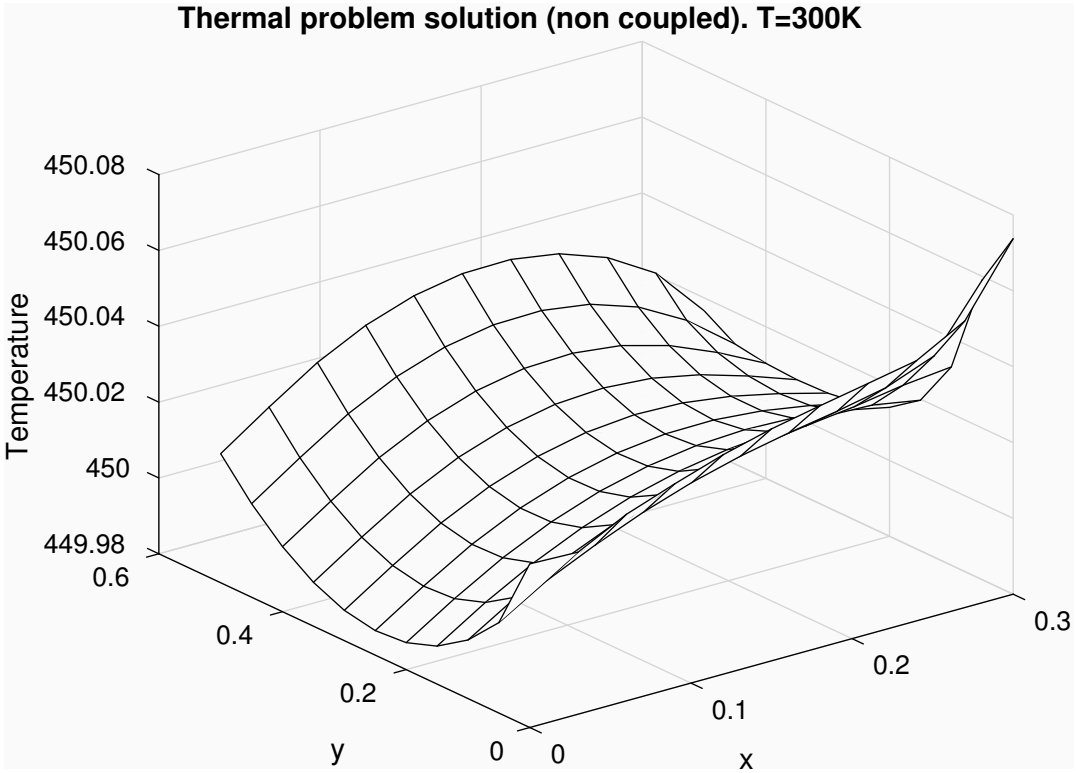

Figure 4: Result for the non-coupled thermal problem

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

```
clc; clear all; close all;
% this is the main file for the thermal part

%% discretization of the domain and building of the mesh for the thermal part
% initialization of the variables lx, ly, nx, ny
lx = 0.3; ly = 0.5; nx = 10; ny = 10;
dx = lx/nx; dy = ly/ny;

% creation of the points and of the mesh
[M, N_points] = points(lx,ly,nx,ny);
[Elem, N_elements] = mesh_domain(lx,ly,nx,ny,M);

%% initialization of the points for the gauss quadrature formula points
w = [0.347855 0.652145 0.652145 0.347855];
u = [-0.861136 -0.339981 0.339981 0.861136];

%% initialization of the variables
% Electrical conductivity constants
k1 = 9.93e6; k2 = 2.5e5;
% thermal conductivity
h = 80,2; % W/(m* K)
% temperature of the environment is kept fixed at 300K
Tr = 300;
% for the non-coupled problem we take Q=1 constant
Q = 1;
% initialize the rhs
```

```
% at the ened the system to solve will be (A+B)*T=rhs
rhs = zeros(N_points,1);

%% creation of the big matrices
% this matrix has to be filled with all the terms of interactions between
% the internal elements
Interaction = zeros(N_points);

%% evaluation of the integral on the variuos elements of the basis
for e = 1:N_elements
    % creation of the Jacobian matrix
    [Jac] = jacobian(M, Elem, e);
    J = det(Jac); %jacobian
    %% evaluation of the integral
    % evaluation of the 4x4 matrix gai_gaj
    % here we will need the gauss points
    gai_gaj = double_integral(M, Elem, e, dx, u, w);
    Contribution = h*gai_gaj*J;
    % update the global matrix values; it will recollect the global number
    % from the local number and the element e
    Interaction = Update(Interaction, Elem, e, Contribution);
    % we have to take into account the heat generated by the electric
    % problem
    A = heat_contribution(M, Elem, e, Q, dx, u);
    Contribution = A*J;
    rhs = Update_rhs(rhs, Elem, e, Contribution);
end

% from the original discretization we extract the boundary elements
[b_Elem, N_b_elements] = b_elements_thermal(lx,ly,nx,ny);
% this matrix must bel filled with the terms given by the boundary effects
Bound = zeros(N_points);
for f = 1:N_b_elements
    [Jac] = jacobian(M, b_Elem, f);
    J = det(Jac);
    %% evaluation of the 1 dimensional integral
    % evaluation of the 2x2 matrix ai_aj
    % here we will need the gauss points
    ai_aj = integral_1D(M, b_Elem, f, dx, u, w);
    Contribution_boundary = h*ai_aj*J;
    % update the global matrix values; it will recollect the global number
    % from the local number and the element e
    Bound = Update_boundary(Bound, b_Elem, f, Contribution_boundary);
end

%% construction of the right hand side of the equation
% the for loop is for the Neumann condition
for f = 1:N_b_elements
    [Jac] = jacobian(M, b_Elem, f);
    J = det(Jac);
    %% evaluation of the 1 dimensional integral
    [ai] = neumann_condition(M, b_Elem, f, Tr, dx, u, w);
    Contribution_boundary = h*ai*J;
    % update the global matrix values; it will recollect the global number
    % from the local number and the element e
    rhs = Update_rhs(rhs, b_Elem, f, Contribution_boundary);
end

T = (Interaction+Bound)\rhs;
```

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

```
function [Jac] = jacobian(M, E, e)
% ———————— INPUT ————————————
% M: matrix of the points with x and y coordinates
% E: matrix of the mesh
% e: global number of the element considered
%
% ———————— OUTPUT ————————————
% Jac: 2x2 matrix with the values of the jacobian matrix

% we obtain the x and y coordinates of the points involved
x_i = M(E(e,:),1);
y_i = M(E(e,:),2);
% xi and eta points in which evaluate the functions
xi = [-1, 1, 1, -1];
eta = [-1, -1, 1, 1];
% n_elements = number of nodes considered
%              [it can change from a boundary element or an internal
%              element]
n_elements = length(x_i);

Jac = zeros(2);
for i = 1:n_elements
    [alfa, dalfa_xi, dalfa_eta] = alpha(xi(i),eta(i));
    Jac(1,1) = Jac(1,1) +dalfa_xi(i)*x_i(i);
    Jac(1,2) = Jac(1,1) +dalfa_xi(i)*y_i(i);
    Jac(2,1) = Jac(1,1) +dalfa_eta(i)*x_i(i);
    Jac(2,2) = Jac(1,1) +dalfa_eta(i)*y_i(i);
end
```

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

```
function [alfa, dalfa_xi, dalfa_eta] = alpha(xi,eta)
% ———————— INPUT ————————————
% xi, eta: nodes in which the function must be evaluated
%          - they can be also vector values
%
% ———————— OUTPUT ————————————
% alfa:      4x1 vector with the values of the four alfas evaluated at (xi,eta)
% dalfa_xi:  4x1 vector of the partial derivatives of alfas wrt xi evaluated
%            at (xi,eta)
% dalfa_eta: 4x1 vector of the partial derivatives of alfas wrt eta evaluated
%            at (xi,eta)

% alfa functions
alfa(1,:) = 1/4*(1-xi).*(1-eta);
alfa(2,:) = 1/4*(1+xi).*(1-eta);
alfa(3,:) = 1/4*(1+xi).*(1+eta);
alfa(4,:) = 1/4*(1-xi).*(1+eta);

% evaluation of the derivative respect to u
dalfa_xi(1,:) = -1/4.*(1-eta);
dalfa_xi(2,:) = 1/4.*(1-eta);
dalfa_xi(3,:) = 1/4.*(1+eta);
dalfa_xi(4,:) = -1/4.*(1+eta);

% evaluation of the derivative respect to v
dalfa_eta(1,:) = -1/4.*(1-xi);
dalfa_eta(2,:) = -1/4.*(1+xi);
dalfa_eta(3,:) = 1/4.*(1+xi);
dalfa_eta(4,:) = 1/4.*(1-xi);
```

```
end
```

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

```
function [M] = Update(M, Elements_matrix, e, A)
% ———————— INPUT ————————————
% Interaction    : matrix to be modified
% Elements_matrix: matrix of the mesh
% e              : number of the global element
% A              : matrix of the local contributions
%
% ———————— OUTPUT ————————————
% M: Matrix modified

if size(A) ~= [4,4]
    error('Local Matrix dimension are not 4x4')
    M = zeros(4);
    return
end

for i=1:4
    for j=1:4
        M(Elements_matrix(e,i),Elements_matrix(e,j)) = M(Elements_matrix(e,i),
            Elements_matrix(e,j))+A(i,j);
    end
end
```

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

```
function [A] = double_integral(M, Elem, e, dx, u, w)

A = zeros(4);
r = M(Elem(e,1),1);

for i = 1:4
    for j = 1:4
        sum = 0;
        for xi_coord = 1:4
            for eta_coord = 1:4
                [Alpha, dalpha_xi, dalpha_eta] = alpha(u(xi_coord),u(eta_coord));
                % the following is the summation of the inner product
                % grad(alpha_i).grad(alpha_j)
                w1 = w(xi_coord);
                w2 = w(eta_coord);
                sum=sum+(dalpha_xi(i)*dalpha_xi(j)+dalpha_eta(i)*dalpha_eta(j))*(r+dx/2+u
                    (xi_coord)*dx)*w1*w2;
            end
        end
        A(i,j) = sum;
    end
end

end
```

```
function [rhs] = Update_rhs(rhs, b_ele_m, f, B)
% algorithm for the updating of the right hand side of the equation
% ———————— INPUT ————————————
% rhs     : vector to be modified
% b_ele_m : matrix of the mesh
% e       : number of the global element
% B       : matrix of the local contributions
```

```
% ———————— OUTPUT ————————————
% M: Matrix modified

s = length(B(:,1));
if size(B) ~= [s,1] % check for matrix dimensions mismatch
    error('Local Matrix dimension are not correct');
    return
end

for i=1:s
    rhs(b_ele_m(f,i)) = rhs(b_ele_m(f,i))+B(i);
end
```

Listing 9: Function for evaluating the contribution given by the Joule effect. For the non coupled problem Q will be considered constant and equal to 1

```
function [A] = heat_contribution(M, Elem, e, Q, dx, u)
% function for evaluating the heat generated by the electric problem
% ——— INPUT  ———
% M    : x,y coordinates of the points
% Elem : global matrix of the boundary elements
% f    : global number of the boundary element considered
% dx   : x displacement
% u    : Gauss nodes
% ——— OUTPUT ———
% A : Term for the updating

A = zeros(4,1);
r = M(Elem(e,1),1);

for i = 1:4
    sum = 0;
    for xi_coord = 1:4
        [Alpha, dalpha_xi, dalpha_eta] = alpha(u(xi_coord),u(xi_coord));
        % summation over the gauss nodes
        w1 = u(xi_coord);
        sum = sum+(Alpha(i))*(r+dx/2+u(xi_coord)*dx)*w1;
    end
    A(i) = Q*sum;
end

end
```

```
function [M] = Update_boundary(M, b_ele_m, f, B)
% algorithm for the updating of the buondary elements
% ———————— INPUT ————————————
% M       : matrix to be modified
% b_ele_m : matrix of the mesh
% e       : number of the global element
% B       : matrix of the local contributions
% ———————— OUTPUT ————————————
% M: Matrix modified

if size(B) ~= [2,2]
    error('Local Matrix dimension are not 2x2');
    M = zeros(2);
    return
end
```

```
for i=1:2
    for j=1:2
        M(b_ele_m(f,i),b_ele_m(f,j)) = M(b_ele_m(f,i),b_ele_m(f,j))+B(i,j);
    end
end
```

```
function [B] = neumann_condition(M, b_Elem, f, Tr, dx, u, w)
% function for evaluating a 1D integral
% ---- INPUT  ----
% M      : x,y coordinates of the points
% b_Elem : global matrix of the boundary elements
% f      : global number of the boundary element considered
% Tr     : Temperature of the room
% dx     : x displacement
% u,w    : Gauss nodes
% ---- OUTPUT ----
% B : Term for the updating

B = zeros(2,1);
r = M(b_Elem(f,1),1);

for i = 1:2
    sum = 0;
    for xi_coord = 1:4
        for eta_coord = 1:4
            [Alpha, dalpha_xi, dalpha_eta] = alpha(u(xi_coord),u(eta_coord));
            % the following is the summation of the inner product
            % alpha_i.alpha_j
            w1 = w(xi_coord);
            sum=sum+(Alpha(i))*(r+dx/2+u(xi_coord)*dx)*w1;
        end
    end
    B(i) = sum*Tr; % update the vector of the contributions
end

end
```

5 Model the Electrical Problem with the Finite Elements Method

5.1 Study Domain and Mesh

Since the domian 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 10, where we have put in input of our function `mesh electric` the number of points in which we want to discretize our domain.

Listing 10: Points

```
function [M,N] = points_electric()
% ------   OUTPUT  ----------
% M: matrix of [x,y] components of the point
% N: number of points generated

% discretization for the electric part
dx=0.01;  dy=0.01;
N = 1; % counter

% bottom electrod
y = 0;
while y <= 0.02
    x = 0;
```

```
    while x <= 0.04
        M(N,:) = [x,y];
        x = x+dx;
        N = N+1;
    end
    y = y+dy;
end

% material
while y <= 0.4
    x = 0;
    while x <= 0.1
        M(N,:) = [x,y];
        x = x+dx;
        N = N+1;
    end
    y = y+dy;
end

% top electrod
while y <= 0.1
    x = 0;
    while x <= 0.02
        M(N,:) = [x,y];
        x = x+dx;
        N = N+1;
    end
    y = y+dy;
end
```

Listing 11: Build relationship for local point and overall point

```
function [E,e] = mesh_electric(M)
% ------   INPUT   ----------
% M : Matrix in which are stored the x and y coordinates of the points
% ------   OUTPUT   ---------
% E : Matrix in which are stored all the points of each element
% e : number of elements

dx = 0.01; dy = 0.01;
e = 1;
y=0;
while y <= 0.02
    x = 0;
    while x <= 0.04
        nx = 0.04/dx;
        E(e,:) = [e,e+1,e+1+nx,e+nx];
        x = x+dx;
        e = e+1;
    end
    y = y+dy;
end

% material
while y <= 0.4
    x = 0;
    while x <= 0.1
        nx = 0.1/dx;
        E(e,:) = [e,e+1,e+1+nx,e+nx];
        x = x+dx;
        e = e+1;
```

```
    end
    y = y+dy;
end

% top electrod
while y <= 0.1
    x = 0;
    while x <= 0.02
        nx = 0.02/dx;
        E(e,:) = [e,e+1,e+1+nx,e+nx];
        x = x+dx;
        e = e+1;
    end
    y = y+dy;
end

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

$$\iiint_{\omega} \sigma \alpha_i \cdot \nabla U\, d\Omega = 0, \quad \forall i. \tag{7}$$

The boundary conditions given can be modelized 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.,

$$\begin{aligned} &(x, y, z) \rightarrow (r, \theta, z) \\ &(x, y, z) \mapsto (r cos(\theta), r sin(\theta), z) \end{aligned} \tag{8}$$

The change of variable given by Eq.8 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

$$\begin{aligned} &(r, \theta, z) \rightarrow (\xi, \eta) \\ &(r, \theta, z) \mapsto (\xi(r, z), \eta(r, z)) \end{aligned} \tag{9}$$

and we will have $dxdydz = det(Jac)d\xi d\eta$.

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 $(A_{ij} + B_{ij})U_i = rhs$

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 12: main program

```
clc; clear all; close all;
% this is the main file for the electric part

%% discretization of the domain and building of the mesh for the thermal part
% initialization of the variables lx, ly, nx, ny
dx = 0.01; dy = 0.01;

% creation of the points and of the mesh
[M, N_points] = points_electric();
[Elem, N_elements] = mesh_electric(M);
[b_Elem, N_b_elements] = b_elements_electric(lx,ly,nx,ny);

%% initialization of the points for the gauss quadrature formula
w=[0.347855 0.652145 0.652145 0.347855];
u=[-0.861136 -0.339981 0.339981 0.861136];

%% initialization of the variables
k = 1;
sigma = 1;

%% creation of the big matrices
% this matrix has to be filled with all the terms of interactions between
% the internal elements
Interaction = zeros(N_points);

%% evaluation of the integral on the variuos elements of the basis
for e = 1:N_elements
    % creation of the Jacobian matrix
    [Jac] = jacobian(M, Elem, e);
    J = det(Jac); %jacobian

    %% evaluation of the integral
    % evaluation of the 4x4 matrix gai_gaj
    % here we will need the gauss points
    gai_gaj = double_integral(M, Elem, e, dx, u, w);
```

```
    Contribution = sigma*gai_gaj*J;
    % update the global matrix values; it will recollect the global number
    % from the local number and the element e
    Interaction = Update(Interaction, Elem, e, Contribution);
end

% this matrix must bel filled with the terms given by the boundary effects
Bound = zeros(N_points);
for f = 1:N_b_elements
    [Jac] = jacobian(M, Elem, e);
    J = det(Jac);
    %% evaluation of the 1 dimensional integral
    % evaluation of the 2x2 matrix ai_aj
    % here we will need the gauss points
    ai_aj = integral_1D(M, b_Elem, f, dx, u, w);
    Contribution_boundary = k*ai_aj*J;
    % update the global matrix values; it will recollect the global number
    % from the local number and the element e
    Bound = Update_boundary(Bound, b_Elem, f, Contribution_boundary);
end
% for testing we use rhs random
rhs = 1000*rand(N_points,1);
T = (Interaction+Bound)\rhs;
```

Where we used the function `jacobian` and `alpha` listed below.

Listing 13: Function for the creation of the jacobian matrix for each element

```
function [Jac] = jacobian(M, E, e)
% ---------- INPUT ------------
% M: matrix of the points with x and y coordinates
% E: matrix of the mesh
% e: global number of the element considered
%
% ---------- OUTPUT -----------
% Jac: 2x2 matrix with the values of the jacobian matrix

% we obtain the x and y coordinates of the points involved
x_i = M(E(e,:),1);
y_i = M(E(e,:),2);
% xi and eta points in which evaluate the functions
xi = [-1, 1, 1, -1];
eta = [-1, -1, 1, 1];
% n_elements = number of nodes considered
%              [it can change from a boundary element or an internal
%              element]
n_elements = length(x_i);

Jac = zeros(2);
for i = 1:n_elements
    [alfa, dalfa_xi, dalfa_eta] = alpha(xi(i),eta(i));
    Jac(1,1) = Jac(1,1) +dalfa_xi(i)*x_i(i);
    Jac(1,2) = Jac(1,1) +dalfa_xi(i)*y_i(i);
    Jac(2,1) = Jac(1,1) +dalfa_eta(i)*x_i(i);
    Jac(2,2) = Jac(1,1) +dalfa_eta(i)*y_i(i);
end
```

Listing 14: Function for the creation of the alpha functions and the derivatives of the latter

```
function [alfa, dalfa_xi, dalfa_eta] = alpha(xi,eta)
% ---------- INPUT ------------
% xi, eta: nodes in which the function must be evaluated
```

```
%          - they can be also vector values
%
% -------- OUTPUT ----------
% alfa:      4x1 vector with the values of the four alfas evaluated at (xi,eta)
% dalfa_xi:  4x1 vector of the partial derivatives of alfas wrt xi evaluated
%            at (xi,eta)
% dalfa_eta: 4x1 vector of the partial derivatives of alfas wrt eta evaluated
%            at (xi,eta)

% alfa functions
alfa(1,:) = 1/4*(1-xi).*(1-eta);
alfa(2,:) = 1/4*(1+xi).*(1-eta);
alfa(3,:) = 1/4*(1+xi).*(1+eta);
alfa(4,:) = 1/4*(1-xi).*(1+eta);

% evaluation of the derivative respect to u
dalfa_xi(1,:) = -1/4.*(1-eta);
dalfa_xi(2,:) = 1/4.*(1-eta);
dalfa_xi(3,:) = 1/4.*(1+eta);
dalfa_xi(4,:) = -1/4.*(1+eta);

% evaluation of the derivative respect to v
dalfa_eta(1,:) = -1/4.*(1-xi);
dalfa_eta(2,:) = -1/4.*(1+xi);
dalfa_eta(3,:) = 1/4.*(1+xi);
dalfa_eta(4,:) = 1/4.*(1-xi);
end
```

Listing 15: Function for update the global matrix with the values obtained for the local element e

```
function [M] = Update(M, Elements_matrix, e, A)
% -------- INPUT ----------
% Interaction    : matrix to be modified
% Elements_matrix: matrix of the mesh
% e              : number of the global element
% A              : matrix of the local contributions
%
% -------- OUTPUT ----------
% M: Matrix modified

if size(A) ~= [4,4]
    error('Local Matrix dimension are not 4x4')
    M = zeros(4);
    return
end

for i=1:4
    for j=1:4
        M(Elements_matrix(e,i),Elements_matrix(e,j)) = M(Elements_matrix(e,i),
            Elements_matrix(e,j))+A(i,j);
    end
end
```

Listing 16: Function for evaluating the double integral $\iint \nabla\alpha_i \nabla\alpha_j$

```
function [A] = double_integral(M, Elem, e, dx, u, w)

A = zeros(4);
r = M(Elem(e,1),1);

for i = 1:4
```

```
    for j = 1:4
        sum = 0;
        for xi_coord = 1:4
            for eta_coord = 1:4
                [Alpha, dalpha_xi, dalpha_eta] = alpha(u(xi_coord),u(eta_coord));
                % the following is the summation of the inner product
                % grad(alpha_i).grad(alpha_j)
                w1 = w(xi_coord);
                w2 = w(eta_coord);
                sum=sum+(dalpha_xi(i)*dalpha_xi(j)+dalpha_eta(i)*dalpha_eta(j))*(r+dx/2+u
                    (xi_coord)*dx)*w1*w2;
            end
        end
        A(i,j) = sum;
    end
end

end
```

6 Model the coupled problem with the finite element method

6.1 Finite Element Formulation

The problem essentially is solved coupling the two equations: first must be solved the electric problem and then the thermal problem; the link between the two equations is the term $Q = J \cdot E = \sigma(\nabla U)^2$.

6.2 Algorithm of the Finite Element Formulation for the Coupled Problem

The algorithm goes as follows:

- Discretize the domain and build the mesh.
- For each element e of the domain solve the electrical part as in the program `main2.m`
- With the value obtained from the electric problem extract for each element the heat Q.
- Solve the thermal problem using the non constant value Q for each element.