U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Department of Mechanical Engineering
Integrated Masters in Mechanical Engineering

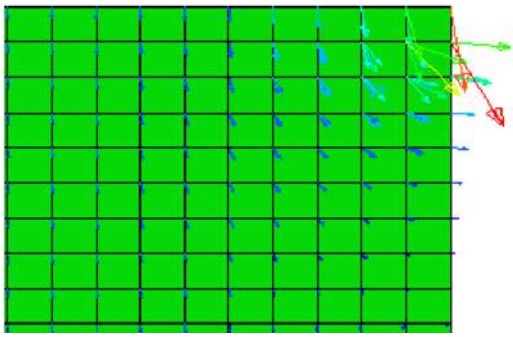
TECHNOLOGICAL PROCESSES SIMULATION

# Application of the Finite Difference Method and the Finite Element Method to Solve a Thermal Problem



| | |
|---|---|
| **Group 1** | Adam Cerovský |
| | Ana Dulce |
| | André Ferreira |





Teacher: Abel Dias dos Santos

March 2014

# Table of Contents


**1 Problem and objectives** ..... *6*

*1.1 Description* ..... *6*

*1.2 Objectives* ..... *6*

**2 Research** ..... *7*

*2.1 Introduction* ..... *7*

*2.2 Finite Difference Method* ..... *8*

*2.3 Finite Element Method* ..... *9*

**3. Procedure** ..... *12*

*3.1 Introduction* ..... *12*

*3.2 Determination of the temperature distribution* ..... *14*

3.2.1 $T_A = 175^oC$ ..... *14*

3.2.1.1 FDM ..... *14*

3.2.1.2 FEM ..... *19*

*3.2.2 $qA, in = 5\ W/m^2$* ..... *19*

3.2.2.1 FDM ..... *19*

3.2.2.2 FEM ..... *20*

*3.3 Determination of the heat flux distribution* ..... *21*

3.3.1 $T_A = 175^oC$ ..... *21*

3.3.1.1 FDM ..... *21*

3.3.1.2 FEM ..... *22*

3.3.2 $qA, in = 5\ W/m^2$ ..... 22

3.3.2.1 FDM ..... *22*

3.3.2.2 FEM ..... *22*

**4 Results and analysis** ..... *24*

*4.1 Results* ..... 24

*4.2 Results analysis* ..... *27*

4.2.1 Consistency and critical analysis ..... *27*

4.2.2 Disparities ..... 27

4.2.3 Other comments ..... *31*

**5 Conclusions** ..... *32*

**6 References** ..... *33*

**Appendixes** ..... *34*

*A1 - Matlab script for $T_A$=175 ºC, for the determination of the temperature distribution.* ..... *35*

*A2 - Matlab script for $qA, in = 5\ W/m^2$, for the determination of the temperature distribution.* ..... *37*

*A3 - Matlab script for $T_A$=175 ºC, for the determination of the heat flux distribution.* ..... *39*

*A4 - Matlab script for $qA, in = 5\ W/m^2$, for the determination of the heat flux distribution.* ..... *42*

*A5 - Abaqus script for* TA *= 175ºC, for the determination of the temperature and heat flux distribution with element size=0.5m.* ..... *45*

*A6 - Abaqus script for* qA, in *= 5 $W/m^2$, for the determination of the temperature and heat flux distribution with element size=0.5m.* ..... *47*

# List of figures

Figure 1 Schematic of the metal plate and the boundary conditions it is subject to.....6

Figure 2 Left: Object discretized with cubes for the FDM. Right: Same object discretized with pyramidal elements for FEM.[3] .....10

Figure 3 Whereas in FEM (right) the mesh can be more fine in certain areas to account for the smaller details, in FDM (left,) since all the elements must be squares, the mesh needs to be unnecessarily fine in low detail places so that it can be fine enough for the more detailed zones.[3].....11

Figure 4 Horizontal and vertical axis of a 12×36 mesh of the 2D plate, its center point and the distinction between two kinds of nodes.....12

Figure 5 Explanation of the meaning of "element size" in this context. .....13

Figure 6 Schematic representation of the metallic plate divided into a 2x6 mesh with 5 internal nodes .....16

Figure 7 Temperature distribution obtained by manually creating a matrix containing the node temperatures in Matlab.....17

Figure 8 Temperature along vertical and horizontal axes of the metal plate .....28

Figure 9 Zoom in of graphics a) and b) for a better viewing of the zones with larger error. .....29

Figure 10 Relationship between the rate of temperature change and the decrease in results precision. .....29

Figure 11 Comparisons of temperature and heat flux at the central node for both methods. .....30

# List of tables

Table 1 A sample of some possible element sizes to use in the discretization of the mesh ..... 14

Table 2 [A] matrix formed for different meshes for condition i) ..... 18

Table 3 [A] matrix formed when one of the boundaries is one of heat flux ..... 20

Table 4 Equations used in the determination of the distribution of heat flux with the FDM ..... 21

# 1 Problem and objectives

## 1.1 Description

A 6x18m stainless steel plate with thermal conductivity $\lambda = 16\ W/m℃$, is subject to two different known boundary conditions

i) $T_A = 175$ ℃;

ii) $\dot{q}_{A,in} = 5\ W/m^2$.

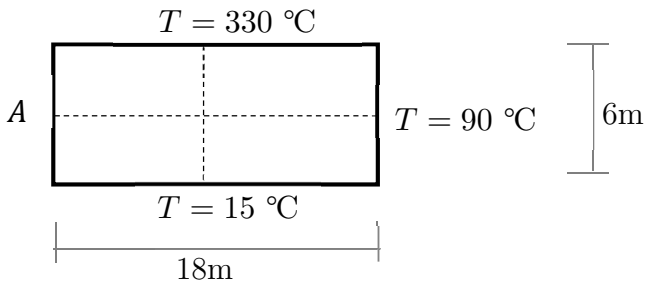



*Figure 1* Schematic of the metal plate and the boundary conditions it is subject to.

## 1.2 Objectives

The main objectives are:

- to determine the temperature and the heat flux distribution along the metal plate for the two boundary conditions by using the Finite Difference Method and the Finite Element Method;
- to compare the results between the two methods for a horizontal and a vertical central lines, and for a central point;
- analyze the influence of the mesh refining on the final results and on processing time;
- compare the usability of the two methods having in mind the software they were used with.

# 2 Research

## 2.1 Introduction

Solving a physical problem through computational modelling requires 4 critical steps:

- to define the problem in terms of a set number of variables for which we want to obtain values, as well as the boundary and initial conditions, if applicable. This step was already done in the previous chapter;
- to represent the physical reality by creating a model and writing the mathematical equations that describe it. For example in fluid mechanics the Navier-Stokes equations are considered to be an accurate representation of fluid motion;
- to solve the equations or simplified versions of them. The equations that generically describe physical problems are normally very complex to solve both analytically and numerically. Therefore it is many times necessary to introduce simplifying assumptions to reduce their complexity and make it possible obtain either an exact or approximate solution;
- To analyze the results critically, that is, to contemplate the results and recognize whether they make sense or not.

The problem described in the previous chapter is one of partial differential equations, whose numerical solutions can obtained by each of the three classical numerical methods finite difference method (FDM), the finite element method (FEM) and the finite volume method (FVM). The latter is out of the scope of this work and thus won't be discussed.

## 2.2 Finite Difference Method

The finite difference method is a mathematical method whose principle is based upon the application of a local Taylor expansion to approximate the partial differential equations (PDE). The FDM uses a very regular, fine and structured mesh formed by a square network of lines to construct the discretization of the PDE. This is a potential limitation of the method when handling complex geometries in multiple dimensions. This problem motivated the development of an integral form of the PDEs and the development of the finite element and the finite volume methods.

Since for this work it will be necessary to do some calculations by hand with the FDM method, it is important to have a basic understanding of how it works.

If, for a moment, we consider the definition of derivative

$$\frac{\partial u}{\partial x} = \lim_{\Delta x \to 0} \frac{u(x + \Delta x) - u(x)}{\Delta x}$$

We see that the closer to zero $\Delta x$ approaches, the smaller the difference between the real value of the derivative and the calculated value. We can get to a similar representation of the derivative by finite differences, by applying a Taylor expansion to the term $u(x + \Delta x)$ until the 1st order terms and doing some algebraic manipulation to result in

$$\frac{u(x + \Delta x) - u(x)}{\Delta x} = \frac{\partial u}{\partial x} + O(x)$$

Where O(x) is the truncation error, in this case of first order. This equation basically says that the slope of a curve at a certain point u(x), i.e., it's derivative, can be approximated by the slope calculated from two points of that curve a certain distance, $\Delta x$, apart from each other. We can now calculate 3 difference approximations for the slope of a curve at a

certain point, using 3 combinations of two points of the same curve. Let's use a simpler notation and call $u(x) = u_i$.

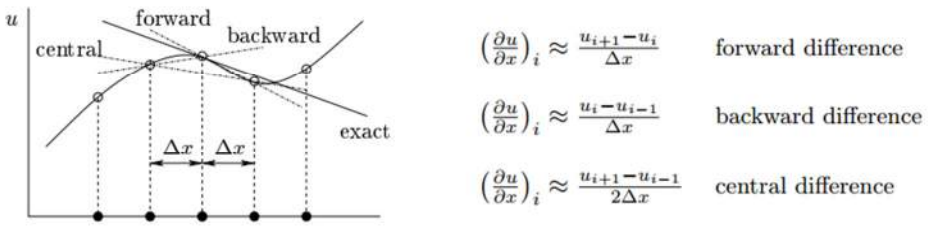



*Figure 2 The three possible ways for calculating a finite difference for a first order derivative according to the FDM[4]*

These differences are applied to each and every node of the discretized mesh, and it's how the method gets the values for each node. It is basically taking a kind of average of a node where it knows the value of the function and another nearby node where it doesn't.

This is a very ancient and general method that can be used in many kinds of problems (though it has found great use specially in fluid dynamics), but it needs a very regular, fine and structured mesh to conduct to the best results. Other disadvantages could be summarized in the difficulty of solving infinite problems and of defining the boundary conditions.

## 2.3 Finite Element Method

The Finite Element Method is a mathematical technique based on modeling the structure using small interconnected elements called finite elements. Usually helped by computer software, it turns a continuous model (real model, where each element has infinite freedom degrees) into a discrete model (analysis model, simpler, where each element has finite freedom degrees). It can be used to solve many kinds of problems such as thermal or structural problems.

The principle of this method is based on an equivalent integral equation to the PDE, which uses interpolations of the nodal solution, element to element, in other words, the unknown function is estimated inside the element by interpolating polynomials based on the function value at the element's nodes. This means that, through a variational integral formulation, the real problem can be introduced in a linear or non-linear equations system.

Compared to the Finite Difference Method, the Finite Element Method allows working with non-structured meshes, this means that contrary to the FDM elements that have to cubes, the FEM elements could have an arbitrary form though the most used ones are triangles and rectangles.

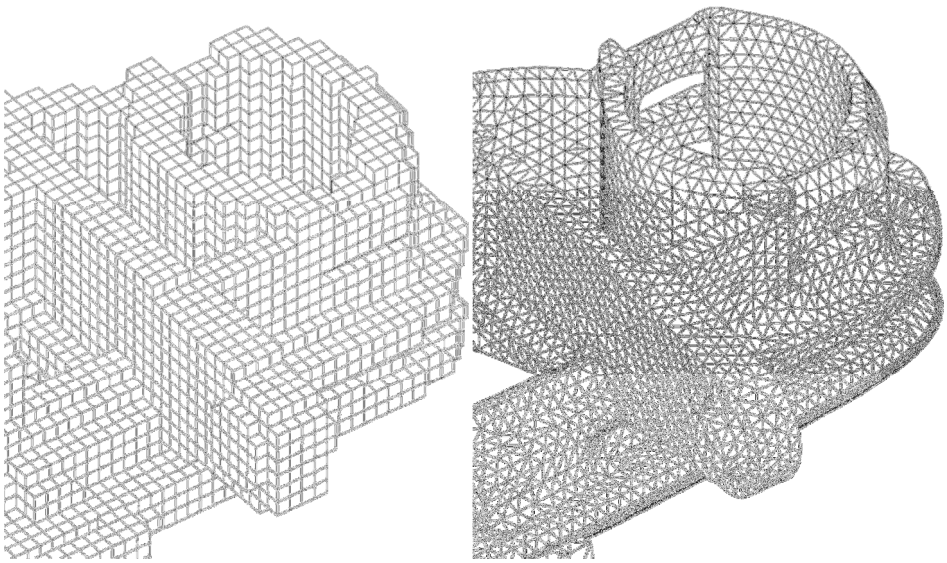


*Figure 3 Left: Object discretized with cubes for the FDM. Right: Same object discretized with pyramidal elements for FEM.*[3]

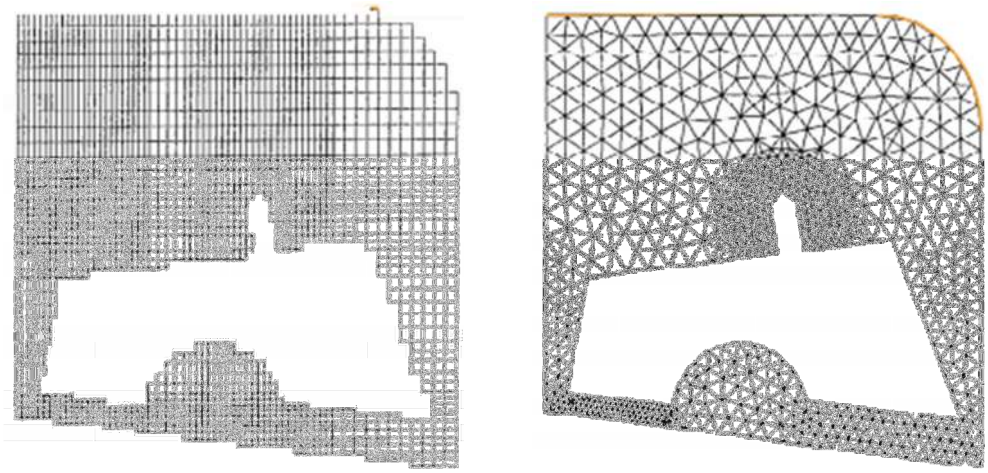


*Figure 4 Whereas in FEM (right) the mesh can be more fine in certain areas to account for the smaller details, in FDM (left,) since all the elements must be squares, the mesh needs to be unnecessarily fine in low detail places so that it can be fine enough for the more detailed zones.*[3]

# 3. Procedure

## 3.1 Introduction

In order to have precise comparisons, we chose to compare the temperatures along both a central horizontal and vertical lines. The only way to have that happen is to always divide the plate in an even number of elements.

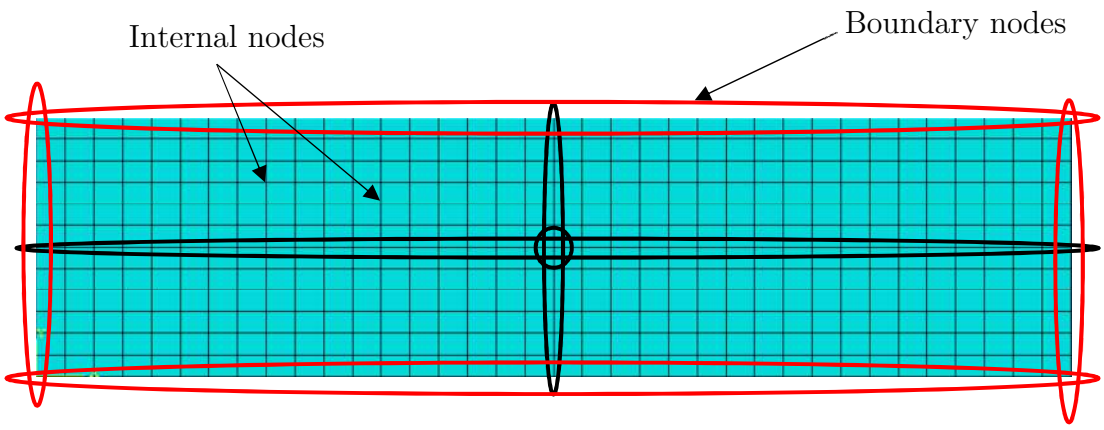



*Figure 5 Horizontal and vertical axis of a 12×36 mesh of the 2D plate, its center point and the distinction between two kinds of nodes.*

At this point, we knew we would be using Abaqus to apply the MEF, and all the elements would be equal squares, so some conclusions were drawn:

- Abaqus uses element size to define the discretization of the mesh, therefore it would make sense to use it in the MDF for the same purpose;

- the Abaqus version available for students is limited to 1000 nodes, so the element sizes that ought to be used could already be predicted.

Before proceeding any further a side note should be made about the terminology being used. It makes more sense to speak about "distance between nodes" or "grid interval" in FDM and "finite element dimensions" or "element size" in FEM, but since in this work the mesh used for both methods as well as the element shape, are always the same, it simplifies to just use one of those interchangeably, and so from here onward we will just refer to element size. as this concept.

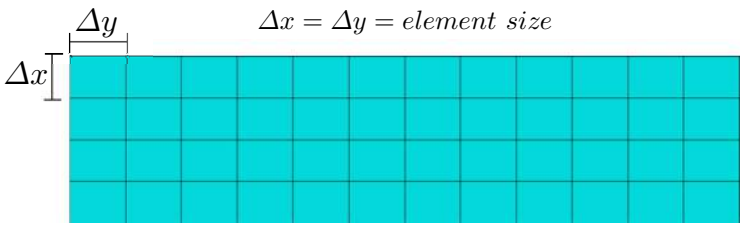



*Figure 6 Explanation of the meaning of "element size" in this context.*

The discretizations to implement had to satisfy the following restrictions:

- Number of column or line elements must be even;
- Number of column elements = 3 · number of line elements;
- Number of elements < 1000;
- Elements must be equal squares;
- Element size must be a rational number, so that it can be input correctly in the software.

We iterated for a couple of numbers, a sample of which are represented on table 1.

*Table 1 A sample of some possible element sizes to use in the discretization of the mesh*

| Element size [m] | No. of internal nodes | No. of total nodes | No. horizontal x vertical elements |
|---|---|---|---|
| 3 | 5 | 21 | 2x6 |
| 2,5 | 1,5 | 6,5 | 2,5x7,5 |
| 2 | 16 | 40 | 3x9 |
| 1,5 | 33 | 65 | 4x12 |
| 1 | 85 | 133 | 6x18 |
| 0,75 | 161 | 255 | 8x24 |
| 0,5 | 385 | 481 | 12x36 |
| 0,3 | 1121 | 1281 | 20x60 |
| 0,15 | 4641 | 4961 | 40x120 |

Not allowed

We picked the 3m as biggest element size. The next possible value, 1,5m, represents a 6.6 fold increase in internal nodes, which is already huge, so it was chosen, and finally 0,5 as the minimum size for the elements.

## 3.2 Determination of the temperature distribution

### 3.2.1 $T_A = 175^oC$

#### 3.2.1.1 FDM

As explained earlier, this method consists in routinely calculating the temperatures at nodes by using a kind of weighted average between other nodes' temperatures that comes from the simplification of the formula for the distribution of heat transfer in a 2D space, in steady state conditions without heat generation:

$$\nabla^2 T + \frac{\dot{q}_f}{\lambda} = \frac{1}{a}\frac{\partial T}{\partial t} \Longrightarrow \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

These partial derivatives can be expanded into a Taylor series reduced to the first order terms and since the nodes' distance between each other will be considered constant and equal in both directions x and y (square

elements) for simplicity reasons, we get a much more simplified formula that allows the calculation of the temperature of the nodes.

$$Node\ (i,j)\colon \underbrace{T_{i+1,j}}_{\rightarrow} + \underbrace{T_{i,j+1}}_{\uparrow} - 4T_{i,j} + \underbrace{T_{i-1,j}}_{\leftarrow} + \underbrace{T_{i,j-1}}_{\downarrow} = 0 \tag{1}$$

With the restriction

$$\forall_{i=\{1,2...C\}}, \forall_{j=\{1,2...L\}}$$

Where

C: Number of columns - 1

L: Number of lines -1

since for this case we're not interested in the temperatures at any of the boundary nodes, and

i: x coordinate of a node (see Figure 7)

j: y coordinate of a node (see Figure 7)

This equation means we are summing the temperature at the node in front, with the temperature with the node above minus 4 times the temperature of the current node, and the temperatures of the nodes below and backwards. This will give us as many equations as the number of internal nodes.

Here a problem arises, since it is very time consuming to do 385 equations by hand. We concluded it would be better to make a script to run in Matlab that would do all calculations for us, where we would just need to input the element size wanted. We will now briefly explain our reasoning that led us to the final script presented in appendix.

We started by analyzing the first two cases, 5 and 33 internal nodes. After seeing a pattern common to both cases, we would then code the program into Matlab and check whether it still worked for smaller elements.

To obtain 5 nodes the metal plate should be divided as follow

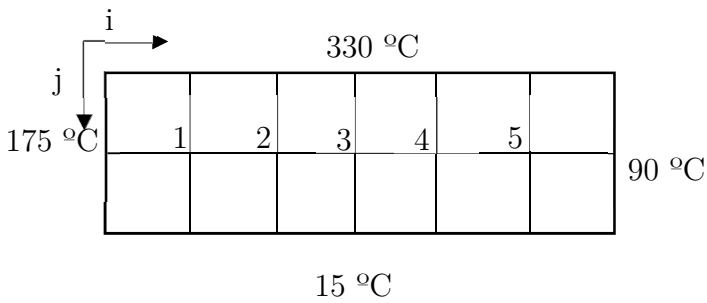



*Figure 7 Schematic representation of the metallic plate divided into a 2x6 mesh with 5 internal nodes*

We then got the 5 equations, one for each internal node, with 5 unknowns, the temperatures at each of those.

$$
\begin{aligned}
Node\ 1:&\ \ T_2 + 300 - 4T_2 + 175 + 15 = 0\\
Node\ 2:&\ \ T_3 + 300 - 4T_2 + T_1 + 15 = 0\\
Node\ 3:&\ \ T_4 + 300 - 4T_2 + T_2 + 15 = 0\\
Node\ 4:&\ \ T_5 + 300 - 4T_2 + T_3 + 15 = 0\\
Node\ 5:&\ \ 90 + 300 - 4T_2 + T_4 + 15 = 0
\end{aligned}
$$

This linear system of equations can be easily solved in Matlab by doing a basic matrix operation. We can form a matrix for the coefficients of the temperatures, let's call it A:

$$
A = \begin{bmatrix} -4 & 1 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 1 & -4 \end{bmatrix}
$$

And a vector for the independent terms, b:

$$
b = \begin{bmatrix} -300 - 175 - 15 \\ -300 - 15 \\ -300 - 15 \\ -300 - 15 \\ -90 - 300 - 15 \end{bmatrix}
$$

We can then say that $\mathbf{A}\times\mathbf{T}=\mathbf{b}$, where $\mathbf{T}=\{T_1,T_2,\ldots T_5\}$. Solving for $\mathbf{T}$ results in

$$\boldsymbol{T} = \begin{bmatrix} 173.0641 \\ 172.2564 \\ 170.9615 \\ 166.5897 \\ 150.3974 \end{bmatrix}$$

Which means

$$\begin{aligned} T_1 &= 173.0641\ ℃ \\ T_2 &= 172.2564\ ℃ \\ T_3 &= 170.9615\ ℃ \\ T_4 &= 166.5897\ ℃ \\ T_5 &= 150.3974\ ℃ \end{aligned}$$

Having the temperatures at each node, they now need to be plotted along with the temperatures at the boundaries of the plate. For that we created a matrix that includes all of the temperatures in their correct places

$$T = \begin{bmatrix} 330 & 330 & 330 & 330 & 330 & 330 & 330 \\ 175 & 173.06 & 172.25 & 170.96 & 166.58 & 150.39 & 90 \\ 15 & 15 & 15 & 15 & 15 & 15 & 15 \end{bmatrix}$$

By then creating the values for the axis where to plot these values, the graph for the distribution of temperatures can be obtained.

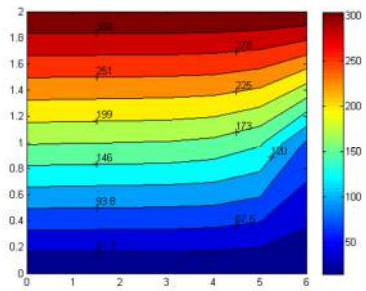
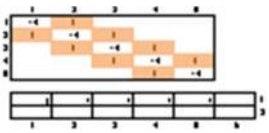
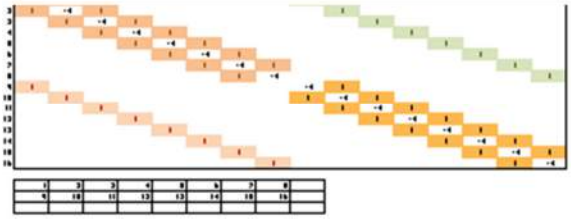
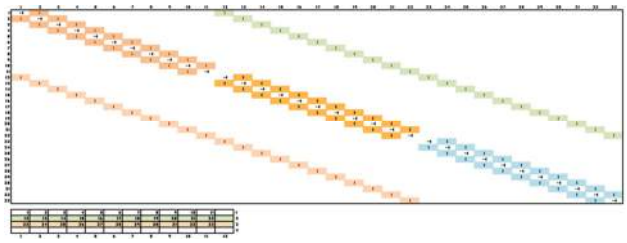
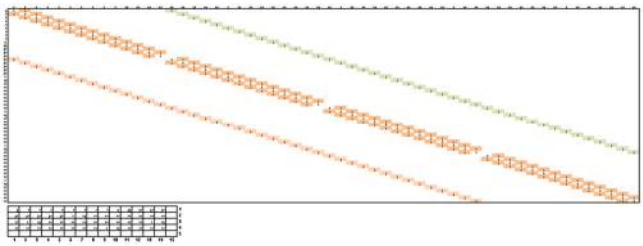



*Figure 8 Temperature distribution obtained by manually creating a matrix containing the node temperatures in Matlab.*

In order to code the script for the automatic creation of a matrix A, we needed to know how it changed as its size increased, that is, we needed to find a pattern common to any matrix A independent of the size.

By analyzing both the equations as well as the figures below we were able to find one and so it was coded.

*Table 2 [A] matrix formed for different meshes for condition i)*

| Mesh | [**A**] matrix formed |
|---|---|
| 2x6 |  |
| 3x9 |  |
| 4x12 |  |
| 5x15 |  |

For the coding of the "b" matrix, we knew it had to contain the independent terms. Which of those to include depends on the node. For example, for all the nodes in the top line, b(node)=b(node)-330. We didn't simply write b(node)=-330 because nodes both in the top line and first or last column also have their respective boundary temperature to sum to the 330ºC. For a node in first line and first column b(node)=-(330+175).

After having a vector with the values of the temperatures at the internal nodes, it was just a matter of putting them in a matrix along with the boundary nodes' temperatures, and plotting.

Further information the script, and the script itself can be found in the appendix A1.

#### 3.2.1.2 FEM

This approach was considerably easier since Abaqus only requires us to input the given variables, and to do the necessary steps to correctly model the plate inside the program: creating a part, assigning it a section with a material with a given thermal conductivity and so on.

Because of the need to compare the results obtained by the two different methods (FDM and FEM), the values for $\Delta_x$ and $\Delta_y$, which for this method represents the finite element dimensions, was the same as those used in the FDM: 3m, 1,5m and 0,5m.

The job was then submitted and the graphic and text results with the values for the temperatures - and heat fluxes - were gotten.

### 3.2.2 $\dot{q}_{A,in} = 5\ \mathbf{W/m^2}$

#### 3.2.2.1 FDM

Once again using equation (1)

$$\begin{aligned}
Node\ 1:&\ \ T_2 + 300 - 4T_2 + T_{A1} + 15 = 0\\
Node\ 2:&\ \ T_3 + 300 - 4T_2 + T_1 + 15 = 0\\
Node\ 3:&\ \ T_4 + 300 - 4T_2 + T_2 + 15 = 0\\
Node\ 4:&\ \ T_5 + 300 - 4T_2 + T_3 + 15 = 0\\
Node\ 5:&\ \ 90 + 300 - 4T_2 + T_4 + 15 = 0
\end{aligned}$$

Where $T_{Ai}$ represents the temperatures at the internal nodes at the left boundary. $T_{A1}$ can be found with

$$q_x = -\frac{\lambda}{2\Delta x}(-3T_{i,j} + 4T_{i+1,j} - T_{i+2,j})$$

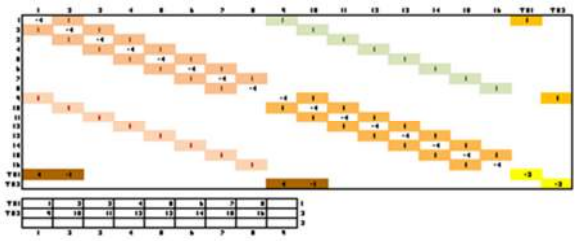
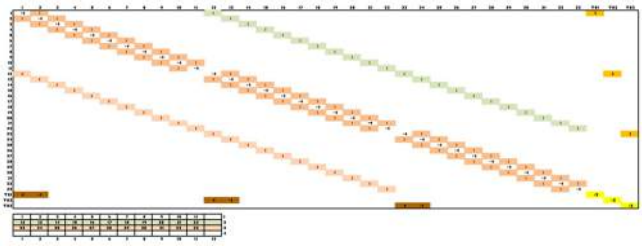
Which, for this case, takes the form

$$5 = -\frac{16}{2 \cdot 3}(-3T_1 + 4T_2 - T_3)$$

$T_{A1}$ was written instead of simply $T_A$ because the temperature at each of the left boundary internal nodes need not to be constant, since it isn't a boundary condition anymore. This also means there will be an extra heat flux equation for each extra internal node on the left boundary of the plate.

To code the script once again a pattern had to be found to create the A matrix. Below are the A matrixes generated for a 3x9 and 4x12 mesh.

*Table 3 [**A**] matrix formed when one of the boundaries is one of heat flux*

| Mesh | [**A**] matrix formed |
|---|---|
| 3x9 |  |
| 4x12 |  |

A similar logic was employed to the boundary condition i) to code the script for this boundary condition, which can be seen in the appendix A2.

#### 3.2.2.2 FEM

A similar procedure of what succeeded in i) was employed in this case. However, this time, the left boundary condition was a heat flux which enters in the plate ($\dot{q}_{A,in} = 5\ W/m^2$), instead of a temperature.

## 3.3 Determination of the heat flux distribution

### 3.3.1 $T_A = 175ºC$

#### 3.3.1.1 FDM

The objective for this part of the work was to determine the heat flux distribution along the plate. For that we used the Fourier equation

$$\dot{q} = -\lambda \cdot \nabla T$$

Which for bidimensional flow takes the form

$$\dot{q}_x = -\lambda \frac{dT}{dx}$$

$$\dot{q}_y = -\lambda \frac{dT}{dx}$$

We can now use the same logic as before, by expanding the derivatives into two Taylor series, but since this time we want 2nd order precision we only stop at the 2nd order terms.

As we will be using square elements, the above derivatives can take a simpler form. As was explained in the introduction the finite differences can be evaluated at the center, in front or at the back. We had to apply all the three cases in order to get results for all the nodes.

*Table 4 Equations used in the determination of the distribution of heat flux with the FDM*

| Equation used | Valid for |
|---|---|
| $\dot{q}_{i,j_{\|x}} = -\frac{\lambda}{2\Delta x}(-3T_{i,j} + 4T_{i+1,j} - T_{i+2,j})$ | First column nodes |
| $\dot{q}_{i,j_{\|y}} = -\frac{\lambda}{2\Delta y}(-3T_{i,j} + 4T_{i,j+1} - T_{i,j+2})$ | First line nodes |
| $\dot{q}_{i,j_{\|x}} = -\frac{\lambda}{2\Delta x}(-T_{i-1,j} + T_{i+1,j})$ <br> $\dot{q}_{i,j_{\|y}} = -\frac{\lambda}{2\Delta y}(-T_{i,j-1} + T_{i,j+1})$ | Other nodes |
| $\dot{q}_{i,j_{\|x}} = -\frac{\lambda}{2\Delta x}(T_{i-2,j} - 4T_{i-1,j} + 3T_{i,j})$ | Last column nodes |

| | |
|---|---|
| $\dot{q}_{i,j_{\|y}} = -\frac{\lambda}{2\Delta y}(T_{i,j-2} - 4T_{i,j-1} + 3T_{i,j})$ | Last line nodes |

From the above equations it can be seen that in order to obtain values for the heat fluxes the temperature distribution must be evaluated first. The temperature distribution to use will then be the ones obtained from the previous chapters.

After having the heat flux in both directions for each node, we calculate the norm of the resultant vector

$$\dot{q}_{i,j} = (\dot{q}_{i,j_{|x}}{}^{2} + \dot{q}_{i,j_{|y}}{}^{2})^{0.5}$$

And plot in the same way the temperature distribution was plotted.

#### 3.3.1.2 FEM

For each element size that the temperature distribution was saved, the heat flux was gotten at the same time. Once all the problem characteristics were explained (for the same boundary conditions), it was possible to get, in the same attempt, temperatures and heat flux results.

### 3.3.2 $\dot{q}_{A,in}$ = 5 W/m²

#### 3.3.2.1 FDM

The process is equal to the employed in the chapter for the determination of the heat flux with $T_A$=175ºC with FDM, with the difference that the distribution of temperatures is determined by the same process as it was previously done in chapter 3.2.2.1.

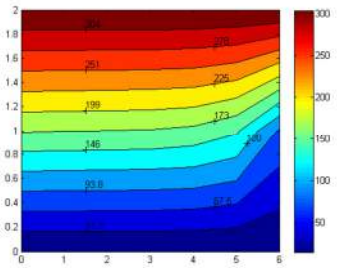
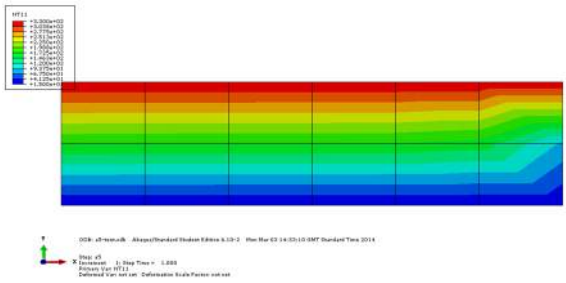
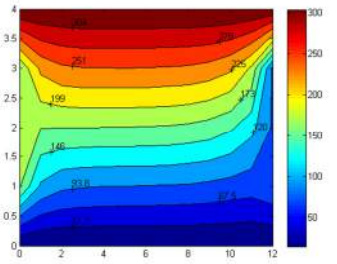
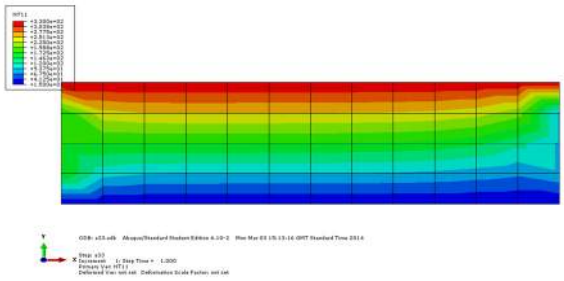
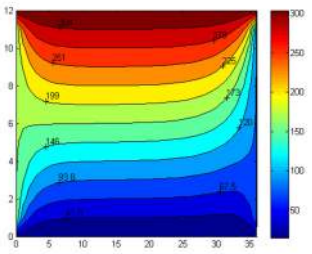
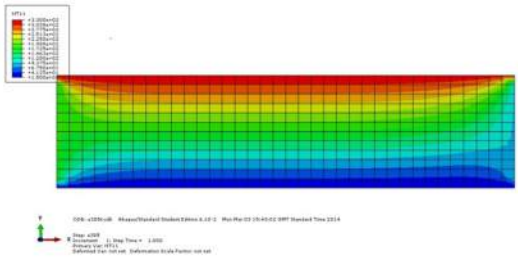
#### 3.3.2.2 FEM

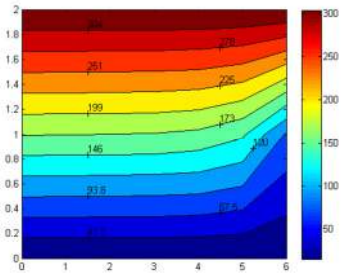
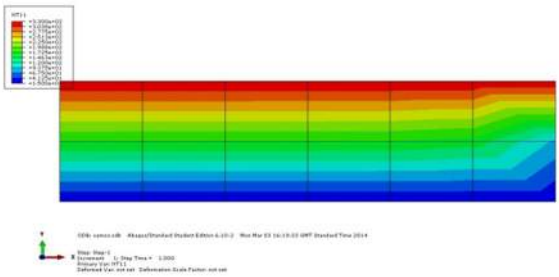
The procedure adopted in this situation was similar to the procedure executed in 3.2.2.2. with the difference that in this situation, it was in

sequence with getting the temperature distribution with the heat flux boundary condition (ii).

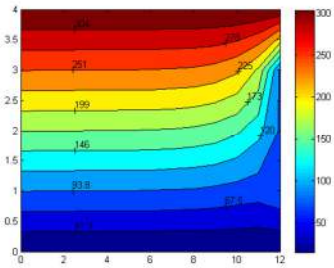
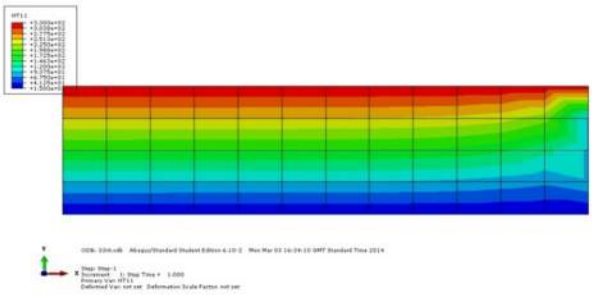
# 4 Results and analysis

## 4.1 Results

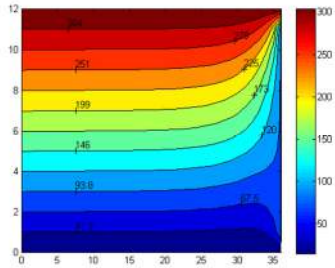
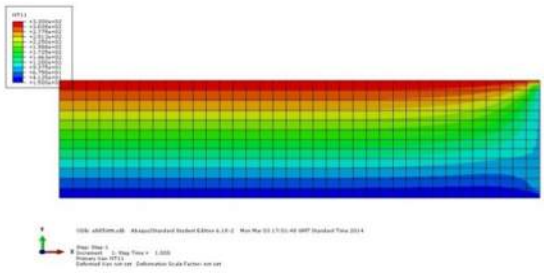
| Matlab | Abaqus | Description |
|---|---|---|
|  |  | 5 nodes T dist i) |
|  |  | 33 nodes T dist i) |
|  |  | 385 nodes T dist i) |

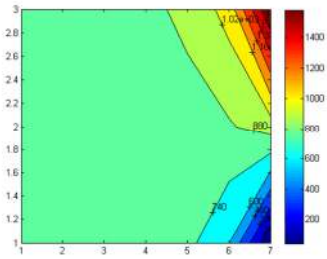
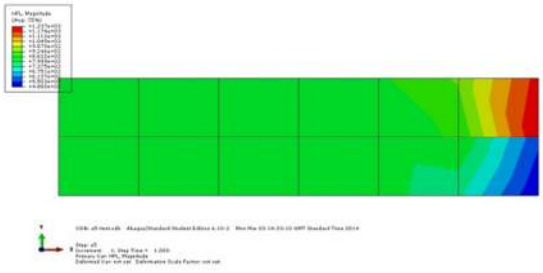
5 nodes
T dist
ii)

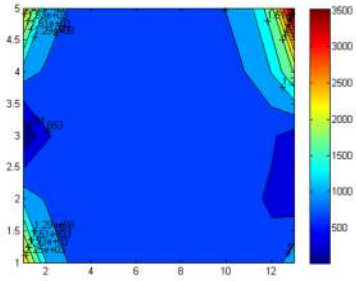
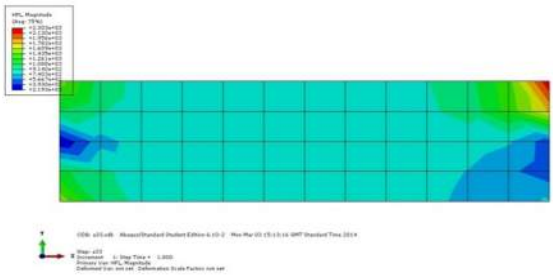
33
nodes
T dist
ii)

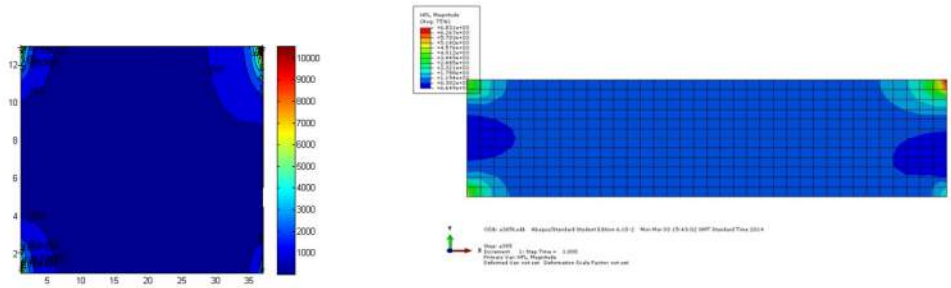
385
nodes
T dist
ii)

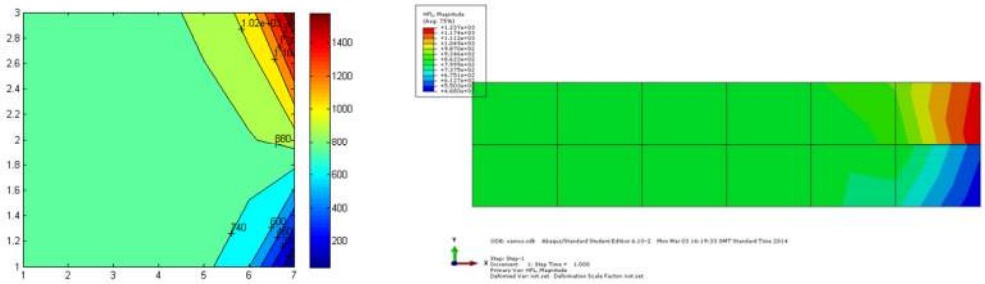
5 nodes
$\dot{q}$ dist
i)

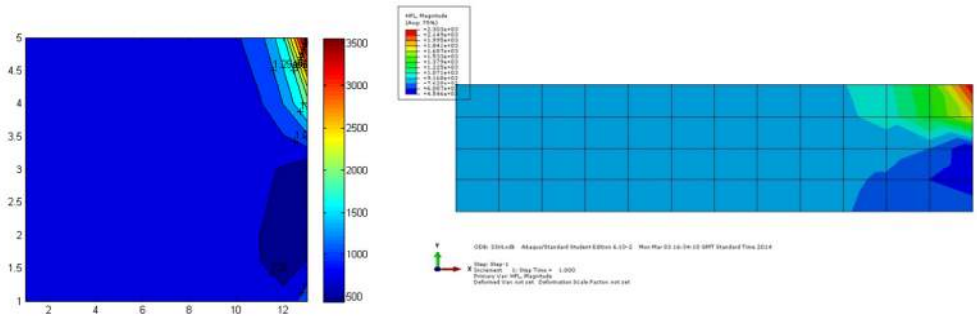
33
nodes
$\dot{q}$ dist
i)



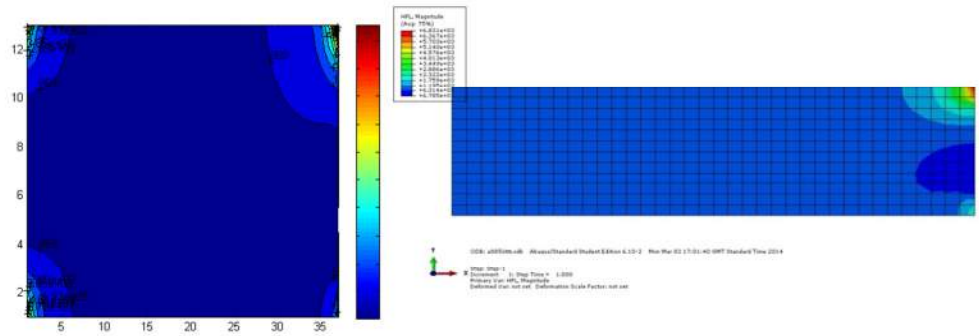
385 nodes $\dot{q}$ dist i)



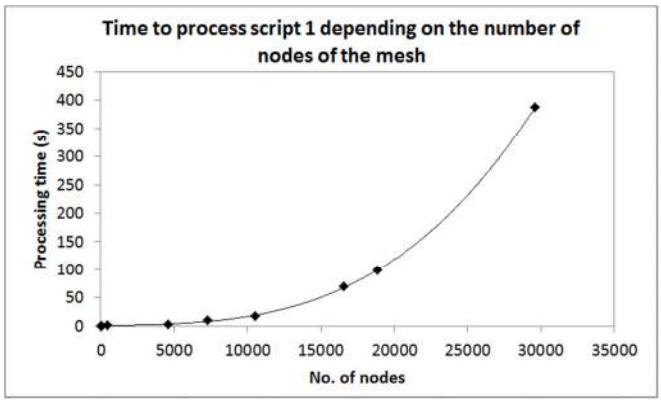
5 nodes $\dot{q}$ dist ii)



33 nodes $\dot{q}$ dist ii)



385 nodes $\dot{q}$ dist ii)



## 4.2 Results analysis

### 4.2.1 Consistency and critical analysis

A primary basic approach is to observe if the images match what one would expect from common sense and if they are reasonably consistent between the two methods.

From the pictures it's possible to see that each boundary is at the temperature imposed by the boundary conditions and in between there is an interpolation between those. The heat flux is also the greatest near the edges, which makes sense given that this is where the difference of temperatures are the biggest.

We notice, however, a big (>20%) difference in the results of the values for the heat fluxes between the two methods on the edges and nearby. This difference is greater the closer to the edges we are and doesn't seem to depend on the discretization of the mesh. On one hand, if the script implemented for the FDM as an error, it could be causing this disparity. On the other hand, it could be due to the different ways that both methods calculate the heat flux.

As for the effect of the increase in number of nodes / elements, for now we can only speculate that the results should be getting closer to the physical reality. The results, specially the temperature distribution ones, do seem to converge.

### 4.2.2 Disparities

In order to better inspect the results a further analysis was made. The temperatures along the vertical and horizontal axes were plotted for the two methods.

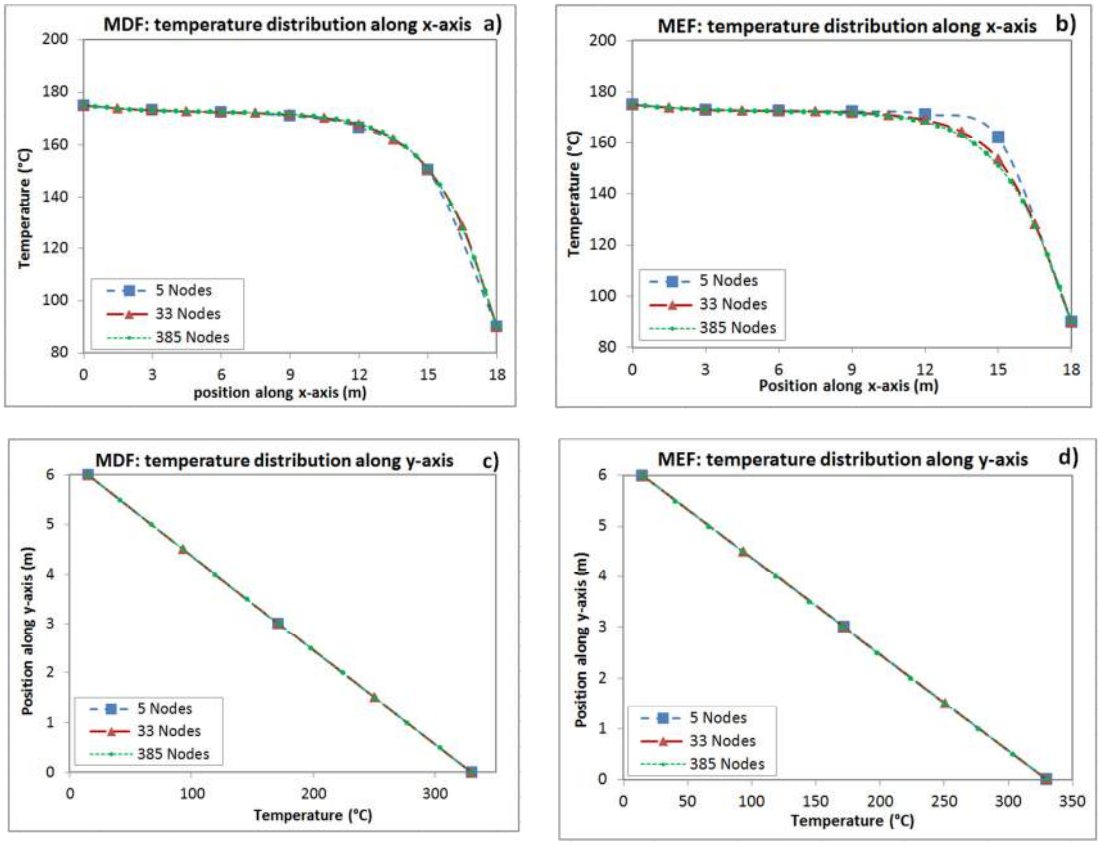



*Figure 9 Temperature along vertical and horizontal axes of the metal plate*

Some slight differences can be observed in both methods, specially when there is a great temperature variation. Nonetheless, the results are still almost coincident. Another observation can be made, which is that that the lower the number of nodes, by using the MDF the temperature tends to be more under estimated, while with the MEF, the temperature tends to be overestimated.

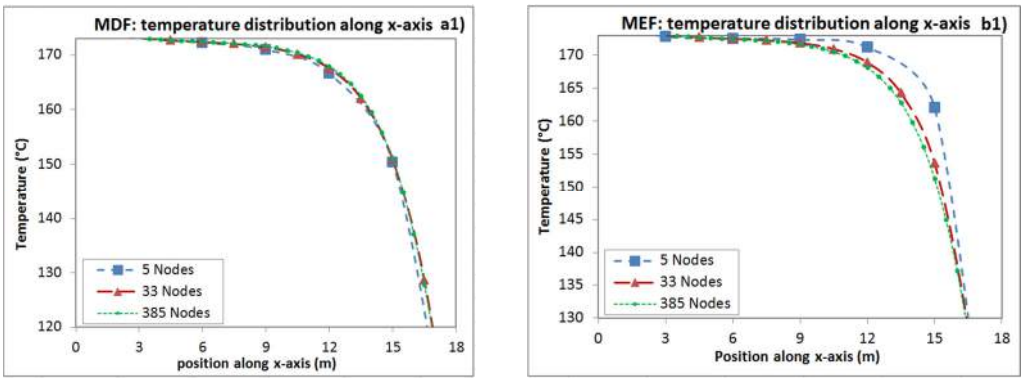



*Figure 10 A zoom in of graphics a) and b) for a better viewing of the zones with larger error.*

However this is the case when the temperature is decreasing. For a general case what this could mean is that MDF tends to overreact to changes while MEF tends to underreact when low number of nodes/elements are employed.

As for the distribution along the y-axis, the results were seemingly identical. This may be due to the fact that the variation of temperature is constant, that is, $\frac{\partial T}{\partial y} = cte$, or $\frac{\partial^2 T}{\partial y^2} = 0$.

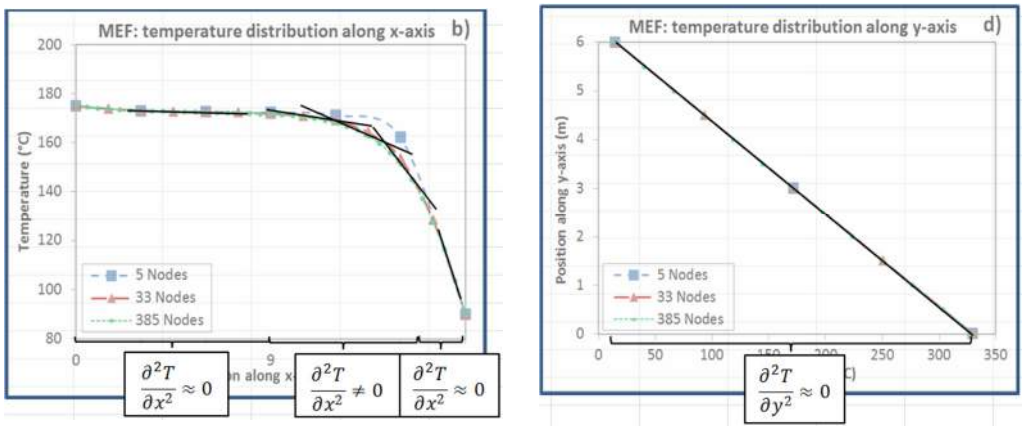



*Figure 11 Relationship between the rate of temperature change and the decrease in results precision.*

It seems, in fact, that the greater the second derivative of T in order to x or y is, i.e., the greater the rate at which the temperature changes, the greater the error[1] is, and this error increases the lower the number of nodes or elements are employed.

[1] "Error" in this context means relative to the best available approximation to the physical reality, which is assumed to be the simulation with the most number of nodes, in this case, the one with 385.

At the moment of choosing the finite elements dimensions, it was important to remember that there had to be a point whose position did not change, in this case, the central point was chosen, so that further analysis could be made. It will be very useful later to compare the accuracy and quality of results. Therefore, below, are the graphics that show the temperature and heat flux distribution in the central point, by both methods analysis.

In the graphic e) it's possible to realize that the MDF method calculates the temperatures with a slightly lesser error than the MEF method. Nevertheless, the main illation should be taken, is that as long as the mesh is more discretized, the results obtained by MDF and MEF methods are getting closer, and, theoretically, with an infinite number of nodes, they'll be equal (in this case, equal to 171,6 °C) and equal to the real[2] temperature at the same point.

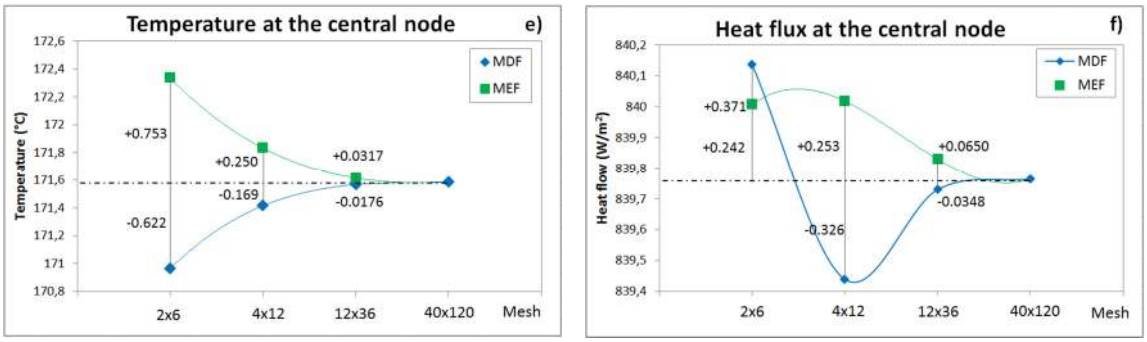



*Figure 12 Comparisons of temperature and heat flux at the central node for both methods.*

Concerning the heat flux two possible explanations can be put forward for what is an apparently strange behavior of the MDF curve:

- If the script has an error, some of the heat flux values obtained with FDM might be wrong;
- The first point was taken with a mesh consisting of only 12 elements, which means the margin of error is very large.

[2] Since the student version of Abaqus is limited to 1000 nodes, and the next smallest element size that satisfied all the restrictions (0.15m=>40x120 mesh) required more than those, the results for the temperature and heat flux for the central node gotten from Matlab were the ones used as the standard, and assumed to be the closest to the physical reality.

This could have caused the first point to have an apparently strange value.

### 4.2.3 Other comments

Even though decreasing element size improves the solution by decreasing the error, as it can be observed on the last result, it also increases processing time, for what it seems, exponentially. This is, of course, a drawback of finding solutions with numerical methods, one that has to be weighted every time one is to submit the problem for analysis.

Finally it should be noted the difference in ease of obtaining a solution with both FDM using Matlab, and FEM using Abaqus. Even though Abaqus performs equally time consuming and complicated calculations, the user interface makes it much simpler to get a solution by providing a graphical user interface that is able, behind the scenes, to do the coding of scripts for the user automatically. This is, of course, a big advantage of using FEM with Abaqus rather than the FDM with Matlab. In fact, around or more than 80% of the total time to obtain results in both programs, was spent in Matlab coding the scripts.

# 5 Conclusions

This work indeed showed how essential it is to know numerical methods to find solutions for partial derivative equations. At the same time, it showed us how powerful and indispensable software like Matlab® and Abaqus® are. These kinds of tools truly simplify the solving process of complex equations resultants of the application of the Finite Difference Method and Finite Element Method and allow obtaining reliable results in a relatively quick and efficient way.

We also saw how the discretizing process of the mesh can influence the results, and how a mesh with fewer nodes will tend to have worse results than a mesh with more elements/nodes and that the reason for this was that if the number of elements/nodes is superior, the analysis has more detail. At best, if we have an infinite number of nodes, all the variables tested by FDM and FEM and the physical reality will be coincident. However, if we have this supposed infinite number of nodes, the running time of the software will also be infinite, which is one of the limitations of the numerical approach for solving these problems.

In conclusion, this work served as proof of how science and technology can contribute for development of knowledge in many realms and showed us that it is always essential to know several mathematical methods as well as software to apply them, to give us an improved capacity to evaluate, compare, and discuss the physical problems that we may have at hand.

# 6 References

[1] Santos, A.D. "Simulaçao de Processos Tecnológicos: Apontamentos"

[2] Peiró, J., Sherwin, S. (2005) "Handbook of Materials Modeling. Volume I: Methods and Models". Department of Aeronautics, Imperial College, London, UK. pp. 1-32.

[3] ESI-Group: The Virtual Space Try Out Company. "FEM-FDM". [powerpoint slides] Available at: http://www.delcam-ural.ru/files/R007%20FEM-FDM.pdf

[4] Kuzmin, D. "Finite Difference Method." Institute of Applied Mathematics University of Dortmund. [Lecture Slides from Introduction to Computational Fluid Dynamics. Available at http://www.mathematik.uni-dortmund.de/~kuzmin/cfdintro/lecture4.pdf]

# Appendixes

# A1 - Matlab script for $T_A$=175 ºC, for the determination of the temperature distribution.

```
clc
clear all
%Asks for the size of the nodes in meters that we want
prompt={'Enter Size of Nodes [meters]. Suggestions: 3, 2, 1.5,
1, 0.75, 0.6, 0.5 ,0.3, 0.15','How do you want to display the
result? Mesh(1), Surface(2) or Contour(3)?','BC left?','BC
up?','BC right?','BC down?'};
%Use these values for the size of the nodes
3,2,1.5,1,0.75,0.6,0.5,0.3,0.15 to have central lines and for
the mesh to be possible
name='Input';
numlines=1;
defaultanswer={'1.5','3','175','330','90','15'};
answer=inputdlg(prompt,name,numlines,defaultanswer);
S = str2num(answer{1});
tm = str2num(answer{2});
bcl = str2num(answer{3});
bcu = str2num(answer{4});
bcr = str2num(answer{5});
bcd = str2num(answer{6});
L=(6/S)-1;
C=(18/S)-1;
N=L*C;
%--------------------------- A MATRIX --------------------------
%Makes a matrix with the size N x N, and puts -4 in the diagonal
A=eye(N,N)*(-4);
%Puts the 1s far from the -4
    for i=1:(L-1)*C
        A(i,i+C)=1;
    end
    for i=C+1:L*C
        A(i,i-C)=1;
    end
%Puts the 1's near the -4
for Li=1:L
    if Li==1
        for i=1:C-1
         A(i,i+1)=1;
         A(i+1,i)=1;
        end
    elseif Li==L
        for i=(Li-1)*C+2:Li*C
                A(i,i-1)=1;
                A(i-1,i)=1;
        end
    else
        for i=(Li-1)*C+1:Li*C-1
               A(i,i+1)=1;
               A(i+1,i)=1;
        end
    end
end
%--------------------------- B VECTOR --------------------------
%Creates vector b full of zeros with dimension = to number of
nodes
b=zeros(N,1);
%Puts the values of BC on the vector B in the correct places
for i=1:L
```

```
    for j=1:C
        a=(i-1)*C+j;
        if i==1
            b(a)=b(a)-bcu;
        end
        if i==L
            b(a)=b(a)-bcd;
        end
        if j==1
            b(a)=b(a)-bcl;
        end
        if j==C
            b(a)=b(a)-bcr;
        end
    end
end
matt=[A\b];
%---------------------- TEMPERATURE MATRIX ---------------------
%Puts the temperatures in the correct places of T matrix
Lt=L+2;
Ct=C+2;
T=eye(Lt,Ct);
for i=1:Lt
    for j=1:Ct
        if i==1
            T(i,j)=bcu;
        elseif i==Lt
            T(i,j)=bcd;
        elseif j==1
           T(i,j)=bcl;
        elseif j==Ct
            T(i,j)=bcr;
        else
            T(i,j)=matt((i-2)*C+j-1);
        end
    end
end
%---------------------------- PLOTTING -------------------------
x=0:1:C+1;
y=L+1:-1:0;
y=y';
if tm==2
surface(x,y,T);
shading interp
elseif tm==3
clabel(contourf(x,y,T,11))
else
mesh(x,y,T)
end
```

## A2 - Matlab script for $\dot{q}_{A,in} = 5\ W/m^2$, for the determination of the temperature distribution.

```
clc
clear all
%Asks for the size of the nodes in meters that we want
prompt={'Enter Size of Nodes [meters]. Suggestions: 3, 2, 1.5,
1, 0.75, 0.6, 0.5 ,0.3, 0.15','How do you want to display the
result? Mesh(1), Surface(2) or Contour(3)?','Heat flux at left
boundary [W/m2] ? ','Thermal conductivity [W/mK] ?','BC up?','BC
right?','BC down?'};
%Nota: valores para  o tamanho dos nós a
%colocar: 3,2,1.5,1,0.75,0.6,0.5,0.3,0.15
name='Input';
numlines=1;
defaultanswer={'1.5','3','5','16','330','90','15'};
answer=inputdlg(prompt,name,numlines,defaultanswer);
S = str2num(answer{1});
tm = str2num(answer{2});
q = str2num(answer{3});
lambda = str2num(answer{4});
bcu = str2num(answer{5});
bcr = str2num(answer{6});
bcd = str2num(answer{7});
L=(6/S)-1;
C=(18/S)-1;
N=L*C;
%--------------------------- A MATRIX --------------------------
%Makes a matrix with the size (N+L) x (N+L), and puts -4 in the
diagonal
A=eye(N+L,N+L)*(-4);
%Puts the 1s far from the -4
    for i=1:(L-1)*C
        A(i,i+C)=1;
    end
    for i=C+1:L*C
        A(i,i-C)=1;
    end
%Puts the 1's near the -4
for Li=1:L
A(Li+L*C,(Li-1)*C+1)=4;
A(Li+L*C,(Li-1)*C+2)=-1;
    if Li==1
        for i=1:C-1
         A(i,i+1)=1;
         A(i+1,i)=1;
        end
    elseif Li==L
        for i=(Li-1)*C+2:Li*C
                A(i,i-1)=1;
                A(i-1,i)=1;
        end
    else
        for i=(Li-1)*C+1:Li*C-1
               A(i,i+1)=1;
               A(i+1,i)=1;
        end
    end
end
for i=C*L+1:C*L+L
A(i,i)=-3;
```

```
end
for i=1:L
A((i-1)*C+1,L*C+i)=1;
end
%--------------------------- B VECTOR --------------------------
%Creates vector b full of zeros with dimension = to number of
nodes
b=zeros(N,1);
%Makes vector B depending on BC
for i=1:L
    for j=1:C
        a=(i-1)*C+j;
        if i==1
            b(a)=b(a)-bcu;
        end
        if i==L
            b(a)=b(a)-bcd;
        end
        if j==1
            b(a)=b(a);
        end
        if j==C
            b(a)=b(a)-bcr;
        end
    end
end
for i=C*L+1:C*L+L
    b(i)=-q*S*2/lambda;
end
matt=[A\b];
%----------------------- TEMPERATURE MATRIX --------------------
%Puts the temperatures in the correct places of T matrix
Lt=L+2;
Ct=C+2;
T=eye(Lt,Ct);
for i=1:Lt
    for j=1:Ct
        if i==1
            T(i,j)=bcu;
        elseif i==Lt
            T(i,j)=bcd;
        elseif j==1
                  T(i,j)=matt(L*C+i-1);
            elseif j==Ct
            T(i,j)=bcr;
        else
            T(i,j)=matt((i-2)*C+j-1);
        end
    end
end
%--------------------------- PLOTTING --------------------------
x=0:1:C+1;
y=L+1:-1:0;
y=y';
if tm==2
surface(x,y,T);
shading interp
elseif tm==3
clabel(contourf(x,y,T,11))
else
mesh(x,y,T)
end
```

# A3 - Matlab script for $T_A$=175 ºC, for the determination of the heat flux distribution.

```
clc
clear all
%Asks for the size of the nodes in meters that we want
prompt={'Enter Size of Nodes [meters]. Suggestions: 3, 2, 1.5,
1, 0.75, 0.6, 0.5 ,0.3, 0.15','How do you want to display the
result?    Mesh(1),    Surface(2)    or    Contour(3)?','Thermal
conductivity  [W/mK]  ?','BC  left?','BC  up?','BC  right?','BC
down?'};
%Nota: valores para  o tamanho dos nós a
%colocar: 3,2,1.5,1,0.75,0.6,0.5,0.3,0.15
name='Input';
numlines=1;
defaultanswer={'1.5','3','16','175','330','90','15'};
answer=inputdlg(prompt,name,numlines,defaultanswer);
S = str2num(answer{1});
tm = str2num(answer{2});
lambda = str2num(answer{3});
bcl = str2num(answer{4});
bcu = str2num(answer{5});
bcr = str2num(answer{6});
bcd = str2num(answer{7});
L=(6/S)-1;
C=(18/S)-1;
N=L*C;
%---------------------------- A MATRIX -------------------------
%Makes a matrix with the size N x N, and puts -4 in the diagonal
A=eye(N,N)*(-4);
%Puts the 1s far from the -4
    for i=1:(L-1)*C
        A(i,i+C)=1;
    end
    for i=C+1:L*C
        A(i,i-C)=1;
    end
%Puts the 1's near the -4
for Li=1:L
    if Li==1
        for i=1:C-1
         A(i,i+1)=1;
         A(i+1,i)=1;
        end
    elseif Li==L
        for i=(Li-1)*C+2:Li*C
                A(i,i-1)=1;
                A(i-1,i)=1;
        end
    else
        for i=(Li-1)*C+1:Li*C-1
               A(i,i+1)=1;
               A(i+1,i)=1;
        end
    end
end
%---------------------------- B VECTOR -------------------------
%Creates vector b full of zeros with dimension = to number of
nodes
b=zeros(N,1);
for i=1:L
```

```
    for j=1:C
        a=(i-1)*C+j;
        if i==1
            b(a)=b(a)-bcu;
        end
        if i==L
            b(a)=b(a)-bcd;
        end
        if j==1
            b(a)=b(a)-bcl;
        end
        if j==C
            b(a)=b(a)-bcr;
        end
    end
end
matt=[A\b];
%Puts the temperatures in the correct places of T matrix
Ct=C+2;
Lt=L+2;
T=eye(Lt,Ct);
for i=1:Lt
    for j=1:Ct
        if i==1
            T(i,j)=bcu;
        elseif i==Lt
            T(i,j)=bcd;
        elseif j==1
           T(i,j)=bcl;
        elseif j==Ct
            T(i,j)=bcr;
        else
            T(i,j)=matt((i-2)*C+j-1);
        end
    end
end
%--------------------------- HEAT FLOW -------------------------
%Vector for x heat flow
qx=zeros((Lt)*(Ct),1);
for i=1:Lt
      for j=1:Ct
            a=(i-1)*Ct+j;
            if j==1
                  qx(a)=abs((-3*T(i,j)+4*T(i,j+1)-
T(i,j+2))*(lambda)/(2*S));
            elseif j==Ct
                  qx(a)=abs((T(i,j-2)-4*T(i,j-
1)+3*T(i,j))*(lambda)/(2*S));
            else
                  qx(a)=abs((-T(i,j-1)+T(i,j+1))*(lambda)/(2*S));
            end
      end
end
%Vector for y heat flow
qy=zeros((Lt)*(Ct),1);
for i=1:Lt
      for j=1:Ct
            a=(i-1)*Ct+j;
            if i==1
                  qy(a)=abs((-3*T(i,j)+4*T(i+1,j)-
T(i+2,j))*(lambda)/(2*S));
            elseif i==Lt
```

```
                  qy(a)=abs((T(i-2,j)-4*T(i-
1,j)+3*T(i,j))*(lambda)/(2*S));
            else
                  qy(a)=abs((-T(i-1,j)+T(i+1,j))*(lambda)/(2*S));
            end
      end
end
%Vector for heat flow
q=zeros((Lt)*(Ct),1);
for a=1:(Lt)*(Ct)
      q(a)=sqrt((qx(a))^2+(qy(a))^2);
end
%Now i need to put vector q in matrix form
Qm=zeros(Lt,Ct);
for i=1:Lt
      for j=1:Ct
            a=(i-1)*Ct+j;
            Qm(i,j)=q(a);
      end
end
%---------------------------- PLOTTING -------------------------
x=1:1:Ct;
y=Lt:-1:1;
y=y';
if tm==2
surface(x,y,Qm);
shading interp
elseif tm==3
clabel(contourf(x,y,Qm,11))
else
mesh(x,y,Qm)
end
```

## A4 - Matlab script for $\dot{q}_{A,in} = 5\ W/m^2$, for the determination of the heat flux distribution.

```
clc
clear all
%Asks for the size of the nodes in meters that we want, how to
display results, thermal conduct. and BC's
prompt={'Enter Size of Nodes [meters]. Suggestions: 3, 2, 1.5,
1, 0.75, 0.6, 0.5 ,0.3, 0.15','How do you want to display the
result? Mesh(1), Surface(2) or Contour(3)?','Heat flux at left
boundary [W/m2] ? ','Thermal conductivity [W/mK] ?','BC up?','BC
right?','BC down?'};
%Nota: valores para  o tamanho dos nós a
%colocar: 3,2,1.5,1,0.75,0.6,0.5,0.3,0.15
name='Input';
numlines=1;
defaultanswer={'1.5','3','5','16','330','90','15'};
answer=inputdlg(prompt,name,numlines,defaultanswer);
S = str2num(answer{1});
tm = str2num(answer{2});
q = str2num(answer{3});
lambda = str2num(answer{4});
bcu = str2num(answer{5});
bcr = str2num(answer{6});
bcd = str2num(answer{7});
L=(6/S)-1;
C=(18/S)-1;
N=L*C;
%--------------------------- A MATRIX --------------------------
%Makes a matrix with the size (N+L) x (N+L), and puts -4 in the
diagonal
A=eye(N+L,N+L)*(-4);
%Puts the 1s far from the -4
    for i=1:(L-1)*C
        A(i,i+C)=1;
    end
    for i=C+1:L*C
        A(i,i-C)=1;
    end
%Puts the 1's near the -4
for Li=1:L
A(Li+L*C,(Li-1)*C+1)=4;
A(Li+L*C,(Li-1)*C+2)=-1;
    if Li==1
        for i=1:C-1
         A(i,i+1)=1;
         A(i+1,i)=1;
        end
    elseif Li==L
        for i=(Li-1)*C+2:Li*C
                A(i,i-1)=1;
                A(i-1,i)=1;
        end
    else
        for i=(Li-1)*C+1:Li*C-1
               A(i,i+1)=1;
               A(i+1,i)=1;
        end
    end
end
for i=C*L+1:C*L+L
```

```
A(i,i)=-3;
end
for i=1:L
A((i-1)*C+1,L*C+i)=1;
end
%-------------------------- B VECTOR -------------------------
%Creates vector b full of zeros with dimension = to number of
nodes
b=zeros(N,1);
for i=1:L
    for j=1:C
        a=(i-1)*C+j;
        if i==1
            b(a)=b(a)-bcu;
        end
        if i==L
            b(a)=b(a)-bcd;
        end
        if j==1
            b(a)=b(a);
        end
        if j==C
            b(a)=b(a)-bcr;
        end
    end
end
for i=C*L+1:C*L+L
    b(i)=-q*S*2/lambda;
end
matt=[A\b];
%--------------------- TEMPERATURE MATRIX --------------------
%Puts the temperatures in the correct places of T matrix
Lt=L+2;
Ct=C+2;
T=eye(Lt,Ct);
for i=1:Lt
    for j=1:Ct
        if i==1
            T(i,j)=bcu;
        elseif i==Lt
            T(i,j)=bcd;
        elseif j==1
                  T(i,j)=matt(L*C+i-1);
            elseif j==Ct
            T(i,j)=bcr;
        else
            T(i,j)=matt((i-2)*C+j-1);
        end
    end
end
%-------------------------- HEAT FLOW ------------------------
%Vector for x heat flow
qx=zeros((Lt)*(Ct),1);
for i=1:Lt
      for j=1:Ct
            a=(i-1)*Ct+j;
            if j==1
                  if i==1
                  qx(a)=abs((-3*T(i,j)+4*T(i,j+1)-
T(i,j+2))*(lambda)/(2*S));
                  elseif i==Lt
```

```
                    qx(a)=abs((-3*T(i,j)+4*T(i,j+1)-
T(i,j+2))*(lambda)/(2*S));
                    else
                    qx(a)=5;
                    end
              elseif j==Ct
                    qx(a)=abs((T(i,j-2)-4*T(i,j-
1)+3*T(i,j))*(lambda)/(2*S));
              else
                    qx(a)=abs((-T(i,j-1)+T(i,j+1))*(lambda)/(2*S));
              end
        end
end
%Vector for y heat flow
qy=zeros((Lt)*(Ct),1);
for i=1:Lt
        for j=1:Ct
              a=(i-1)*Ct+j;
              if i==1
                    qy(a)=abs((-3*T(i,j)+4*T(i+1,j)-
T(i+2,j))*(lambda)/(2*S));
              elseif i==Lt
                    qy(a)=abs((T(i-2,j)-4*T(i-
1,j)+3*T(i,j))*(lambda)/(2*S));
              else
                    qy(a)=abs((-T(i-1,j)+T(i+1,j))*(lambda)/(2*S));
              end
        end
end
%Vector for heat flow
q=zeros((Lt)*(Ct),1);
for a=1:(Lt)*(Ct)
        q(a)=sqrt((qx(a))^2+(qy(a))^2);
end
%Now i need to put vector q in matrix form
Qm=zeros(Lt,Ct);
for i=1:Lt
        for j=1:Ct
              a=(i-1)*Ct+j;
              Qm(i,j)=q(a);
        end
end
%---------------------------- PLOTTING -------------------------
x=1:1:Ct;
y=Lt:-1:1;
y=y';
if tm==2
surface(x,y,Qm);
shading interp
elseif tm==3
clabel(contourf(x,y,Qm,11))
else
mesh(x,y,Qm)
end
```

## A5 - Abaqus script for $T_A$ = 175ºC, for the determination of the temperature and heat flux distribution with element size=0.5m.

```
# -*- coding: mbcs -*-
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
mdb.models.changeKey(fromName='Model-1', toName='Modelo2')
mdb.models['Modelo2'].ConstrainedSketch(name='__profile__',
sheetSize=40.0)
mdb.models['Modelo2'].sketches['__profile__'].rectangle(point1=(
0.0, 0.0), point2=(
    18.0, 6.0))
mdb.models['Modelo2'].Part(dimensionality=TWO_D_PLANAR,
name='Modelo2', type=
    DEFORMABLE_BODY)
mdb.models['Modelo2'].parts['Modelo2'].BaseShell(sketch=
    mdb.models['Modelo2'].sketches['__profile__'])
del mdb.models['Modelo2'].sketches['__profile__']
mdb.models['Modelo2'].Material(name='Material-1')
mdb.models['Modelo2'].materials['Material-
1'].Conductivity(table=((16.0, ), ))
mdb.models['Modelo2'].HomogeneousSolidSection(material='Material
-1', name=
    'Section-1', thickness=None)
mdb.models['Modelo2'].parts['Modelo2'].SectionAssignment(offset=
0.0, offsetField='',
    offsetType=MIDDLE_SURFACE, region=Region(

faces=mdb.models['Modelo2'].parts['Modelo2'].faces.getSequenceFr
omMask(mask=(
    '[#1      ]',       ),       )),       sectionName='Section-1',
thicknessAssignment=FROM_SECTION)
mdb.models['Modelo2'].rootAssembly.DatumCsysByDefault(CARTESIAN)
mdb.models['Modelo2'].rootAssembly.Instance(dependent=ON,
name='Modelo2-1', part=
    mdb.models['Modelo2'].parts['Modelo2'])
mdb.models['Modelo2'].HeatTransferStep(amplitude=RAMP,
name='Step-1', previous=
    'Initial', response=STEADY_STATE)
mdb.models['Modelo2'].TemperatureBC(amplitude=UNSET,
createStepName='Step-1',
    distributionType=UNIFORM,       fieldName='',       fixed=OFF,
magnitude=175.0, name=
    'BC-1', region=Region(
    edges=mdb.models['Modelo2'].rootAssembly.instances['Modelo2-
1'].edges.getSequenceFromMask(
    mask=('[#8 ]', ), )))
mdb.models['Modelo2'].TemperatureBC(amplitude=UNSET,
createStepName='Step-1',
```

```
    distributionType=UNIFORM,       fieldName='',       fixed=OFF,
magnitude=90.0, name=
    'BC-2', region=Region(
    edges=mdb.models['Modelo2'].rootAssembly.instances['Modelo2-
1'].edges.getSequenceFromMask(
    mask=('[#2 ]', ), )))
mdb.models['Modelo2'].TemperatureBC(amplitude=UNSET,
createStepName='Step-1',
    distributionType=UNIFORM,       fieldName='',       fixed=OFF,
magnitude=330.0, name=
    'BC-3', region=Region(
    edges=mdb.models['Modelo2'].rootAssembly.instances['Modelo2-
1'].edges.getSequenceFromMask(
    mask=('[#4 ]', ), )))
mdb.models['Modelo2'].TemperatureBC(amplitude=UNSET,
createStepName='Step-1',
    distributionType=UNIFORM,       fieldName='',       fixed=OFF,
magnitude=15.0, name=
    'BC-4', region=Region(
    edges=mdb.models['Modelo2'].rootAssembly.instances['Modelo2-
1'].edges.getSequenceFromMask(
    mask=('[#1 ]', ), )))
mdb.models['Modelo2'].parts['Modelo2'].setMeshControls(elemShape
=QUAD, regions=

mdb.models['Modelo2'].parts['Modelo2'].faces.getSequenceFromMask
(('[#1 ]', ), ),
    technique=STRUCTURED)
mdb.models['Modelo2'].parts['Modelo2'].seedPart(deviationFactor=
0.1, size=0.5)
mdb.models['Modelo2'].parts['Modelo2'].setElementType(elemTypes=
(ElemType(
    elemCode=DC2D4,                       elemLibrary=STANDARD),
ElemType(elemCode=DC2D3,
    elemLibrary=STANDARD)), regions=(

mdb.models['Modelo2'].parts['Modelo2'].faces.getSequenceFromMask
(('[#1 ]', ), ), ))
mdb.models['Modelo2'].parts['Modelo2'].generateMesh()
```

## A6 - Abaqus script for $\dot{q}_{A,in} = 5$ W/m², for the determination of the temperature and heat flux distribution with element size=0.5m.

```
# -*- coding: mbcs -*-
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
sheetSize=40.0)
mdb.models['Model-
1'].sketches['__profile__'].rectangle(point1=(0.0, 0.0),
    point2=(18.0, 6.0))
mdb.models['Model-1'].Part(dimensionality=TWO_D_PLANAR,
name='Part-1', type=
    DEFORMABLE_BODY)
mdb.models['Model-1'].parts['Part-1'].BaseShell(sketch=
    mdb.models['Model-1'].sketches['__profile__'])
del mdb.models['Model-1'].sketches['__profile__']
mdb.models['Model-1'].Material(name='Material-1')
mdb.models['Model-1'].materials['Material-
1'].Conductivity(table=((16.0, ), ))
mdb.models['Model-
1'].HomogeneousSolidSection(material='Material-1', name=
    'Section-1', thickness=None)
mdb.models['Model-1'].parts['Part-
1'].SectionAssignment(offset=0.0,
    offsetField='', offsetType=MIDDLE_SURFACE, region=Region(
    faces=mdb.models['Model-1'].parts['Part-
1'].faces.getSequenceFromMask(
    mask=('[#1     ]',     ),     )),     sectionName='Section-1',
thicknessAssignment=
    FROM_SECTION)
mdb.models['Model-1'].rootAssembly.DatumCsysByDefault(CARTESIAN)
mdb.models['Model-1'].rootAssembly.Instance(dependent=ON,
name='Part-1-1',
    part=mdb.models['Model-1'].parts['Part-1'])
mdb.models['Model-1'].HeatTransferStep(amplitude=RAMP,
name='Step-1', previous=
    'Initial', response=STEADY_STATE)
mdb.models['Model-1'].SurfaceHeatFlux(createStepName='Step-1',
magnitude=5.0,
    name='Load-1', region=Region(
    side1Edges=mdb.models['Model-
1'].rootAssembly.instances['Part-1-
1'].edges.getSequenceFromMask(
    mask=('[#8 ]', ), )))
mdb.models['Model-1'].TemperatureBC(amplitude=UNSET,
createStepName='Step-1',
    distributionType=UNIFORM,       fieldName='',       fixed=OFF,
magnitude=90.0, name=
    'BC-1', region=Region(
```

```
    edges=mdb.models['Model-1'].rootAssembly.instances['Part-1-
1'].edges.getSequenceFromMask(
    mask=('[#2 ]', ), )))
mdb.models['Model-1'].TemperatureBC(amplitude=UNSET,
createStepName='Step-1',
    distributionType=UNIFORM,       fieldName='',       fixed=OFF,
magnitude=330.0, name=
    'BC-2', region=Region(
    edges=mdb.models['Model-1'].rootAssembly.instances['Part-1-
1'].edges.getSequenceFromMask(
    mask=('[#4 ]', ), )))
mdb.models['Model-1'].TemperatureBC(amplitude=UNSET,
createStepName='Step-1',
    distributionType=UNIFORM,       fieldName='',       fixed=OFF,
magnitude=15.0, name=
    'BC-3', region=Region(
    edges=mdb.models['Model-1'].rootAssembly.instances['Part-1-
1'].edges.getSequenceFromMask(
    mask=('[#1 ]', ), )))
mdb.models['Model-1'].parts['Part-
1'].setMeshControls(elemShape=QUAD, regions=
    mdb.models['Model-1'].parts['Part-
1'].faces.getSequenceFromMask(('[#1 ]',
    ), ), technique=STRUCTURED)
mdb.models['Model-1'].parts['Part-
1'].seedPart(deviationFactor=0.1, size=0.5)
mdb.models['Model-1'].parts['Part-
1'].setElementType(elemTypes=(ElemType(
    elemCode=DC2D4,                         elemLibrary=STANDARD),
ElemType(elemCode=DC2D3,
    elemLibrary=STANDARD)), regions=(
    mdb.models['Model-1'].parts['Part-
1'].faces.getSequenceFromMask(('[#1 ]',
    ), ), ))
mdb.models['Model-1'].parts['Part-1'].generateMesh()
```