Finite Element Method Project

Louise Karsten
Anna Sjerling

May 25, 2020

Contents

1 Introduction **3**
- 1.1 Task 1 . . . 4
- 1.2 Task 2 . . . 4
- 1.3 Task 3 . . . 5

2 Procedure **6**
- 2.1 Task 1 . . . 6
  - 2.1.1 Theory . . . 6
  - 2.1.2 Matlab . . . 8
- 2.2 Task 2 . . . 10
  - 2.2.1 Theory . . . 10
  - 2.2.2 Matlab . . . 11
- 2.3 Task 3 . . . 12
  - 2.3.1 Theory . . . 12
  - 2.3.2 Matlab . . . 14

3 Results **15**
- 3.1 Task 1 . . . 15
- 3.2 Task 2 . . . 16
- 3.3 Task 3 . . . 18

4 Discussion **19**
- 4.1 Task 1 . . . 19
- 4.2 Task 2 . . . 19
- 4.3 Task 3 . . . 20

5 Appendix **22**
- 5.1 Task 1 . . . 22
- 5.2 Task 2 . . . 25
- 5.3 Task 3 . . . 26

1 Introduction

The aim of this report is to solve problems regarding temperature and stress distributions for a given integrated circuit, using a finite element analysis and Matlab. The heat is produced when electric current passes through the circuit. An application of this is servers belonging to Netflix where the generated heat, Q, is proportional to data consumption. (1)

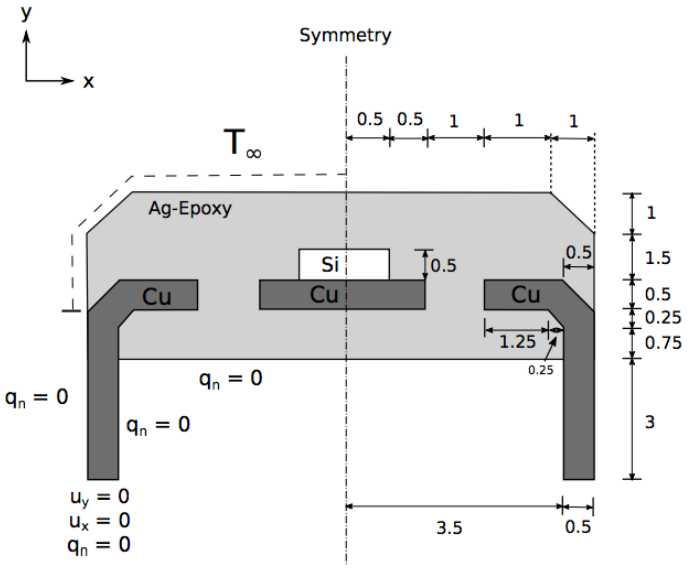


Figure 1: Shows the cross section of the circuit with given geometry. (1)

In figure 1 the component made of silicon (Si) is made of a semiconducting material where the heat is assumed to be generated, called a die. The conducting frame is made of copper (Cu) and silver (Ag) filled epoxy makes up the package. Because of symmetry only half of the circuit will be analysed and then extended. Table 1 shows given values and boundary conditions where Q is the generated heat, T_∞ is the constant temperature some distance away from the edge of convection, T_{start} is the initial temperature of the circuit before use and t is the thickness of the board. The parameters $u_x = u_y = 0$ reflects that the bottom boundaries are fixated. α_c is used along the boundary with T_∞ where convection is present according to $q_n = \alpha_c(T - T_\infty)$ and the remaining boundaries have $q_n = 0$ which means they are insulated. Table 2 shows material parameters for the materials in the circuit board. (1)

Table 1: Given values and boundary conditions. (1)

Label	Unit	Value
Q	W/m^3	$5 \cdot 10^7$
T_∞	°C	18
T_{start}	°C	30
t	mm	10
u_x	mm	0
u_y	mm	0
α_c	$W/(m^2 \cdot K)$	40

Table 2: Given material parameters. (1)

Parameter	Unit	Ag-epoxy	Si	Cu
Young's modulus, E	GPa	7	165	128
Poisson's ratio, ν	-	0.3	0.22	0.36
Expansion coefficient, α	1/K	$4 \cdot 10^{-5}$	$2.6 \cdot 10^{-6}$	$17.6 \cdot 10^{-6}$
Density, ρ	kg/m^3	2500	2530	8930
Specific heat, c_p	$J/(kg \cdot K)$	1000	703	386
Thermal conductivity, k	$W/(m \cdot K)$	5	149	385

1.1 Task 1

The purpose of this task was to find and present the stationary temperature distribution when assuming that the integrated circuit was in an environment with $T_\infty = 18$°C and the heat generated in the die was $Q = 5 \cdot 10^7$ W/m^3. How a 25% decrease in data consumption would affect the maximum temperature was to be investigated. (1)

1.2 Task 2

The purpose of this task was to first determine how the temperature distribution changed with respect to time during the first 20 minutes after startup and then find the stationary temperature distribution, assuming $T_\infty = 18$°C and $Q = 5 \cdot 10^7$ W/m^3. The temperature distribution at representative instants and at stationary conditions was to be presented together with the peak temperature for full quality and reduced quality. (1)

1.3 Task 3

The purpose of this task was to find the effective von Mises stress field and the displacement field due to thermal expansion from the stationary temperature distribution, assuming $T_\infty = 18°\text{C}$ and $Q = 5 \cdot 10^7$ W/m^3. The stress distributions and displacement field was to be presented for full and reduced quality along with their respective maximum values. (1)

2 Procedure

Below the work done in Matlab is described together with the theory needed to understand the code for every task.

2.1 Task 1

2.1.1 Theory

In order to get the differential equation for two-dimensional heat flow, a balance equation for an infinitely small part needs to be found. If the stationary problem is considered the following equation can be set up:

$$\int_A Q t dA = \oint_{\mathscr{L}} q_n t d\mathscr{L} \tag{1}$$

where Q is the heat supplied per unit volume and time to the body, q_n is the flux leaving the body and t is the thickness of the body. Using equation 1 together with Gauss divergence theorem and the assumption that the area A is arbitrary gives the balance equation 2. (2)

$$div(t\mathbf{q}) = tQ \tag{2}$$

This eventually leads to a differential equation known as Laplace equation and then the strong form:

$$div(t\mathbf{D}\nabla T) + tQ = 0 \tag{3a}$$

$$q_n = \mathbf{q}^{\mathbf{T}}\mathbf{n} = h \tag{3b}$$

$$T = g \tag{3c}$$

where $\mathbf{D}$ is the constitutive matrix and T is the temperature. Equation 3a is valid in region A, equation 3b is valid on a part of the boundary where the flux is given as h and equation 3c is valid on a part of the boundary where the temperature is given as g. For a more in depth description of how to get to the strong form, see *Introduction to the finite element method* pages 81-83. (2)

The corresponding weak form is given by equation 4.

$$\int_A (\nabla v)^T t\mathbf{D}\nabla T dA = -\int_{\mathscr{L}_h} vht d\mathscr{L} - \int_{\mathscr{L}_g} vq_n t d\mathscr{L} + \int_A vQt dA \tag{4a}$$

$$T = g \tag{4b}$$

where ν is an arbitrary weight function. To get the weak form from the balance equation, see equation 2, the balance equation is multiplied with an arbitrary weight function. Then it is integrated over the considered region, the Green-Gauss theorem is used and the boundary conditions are inserted. For a more in depth description of how to get to the weak form, see *Introduction to the finite element method* pages 85. This theory is the basis for a finite element method approximation of heat flow. (2)

The unknowns of the problem are called the problem's degrees of freedom. When solving finite element method-tasks it is often suitable to create two matrices, one called Dof and the other one called $Edof$. In the Dof-matrix, each element's degrees of freedom are lined up. The $Edof$-matrix adds a column first with the element number. (2)

Convection is when a flux is present along the boundary, in other words not the boundaries where the flux (q_n) equals zero, see figure 1. This flux will affect the boundary vector $\mathbf{f}_b$ since $q_n = \alpha(T - T_\infty)$. Due to convection, the problem $\mathbf{K}\,\mathbf{a} = \mathbf{f}$ is therefore formulated as equation 5.

$$(\mathbf{K} + \mathbf{K}_c)\mathbf{a} = -\int_{\mathscr{L}_h} \mathbf{N}^T h t dL - \int_{\mathscr{L}_g} \mathbf{N}^T q_n t dL + T_\infty \int_{\mathscr{L}_c} \mathbf{N}^T \alpha t dL\mathscr{L} + \mathbf{f}_l \quad (5)$$

where $\mathscr{L}_h$, $\mathscr{L}_g$ and $\mathscr{L}_c$ are parts of the boundary where the flux q_n is prescribed, the temperature T and where there is convection respectively. In *Introduction to the finite element method* on page 220, there is a figure showing the boundaries along with their conditions for the interested reader. (2)

Given the circuit seen in figure 1, $h = 0$ along $\mathscr{L}_h$ and there is no boundary with prescribed temperature which results in equation 6.

$$(\mathbf{K} + \mathbf{K}_c)\mathbf{a} = T_\infty \int_{\mathscr{L}_c} \mathbf{N}^T \alpha t dL\mathscr{L} + \mathbf{f}_l \quad (6)$$

$\mathbf{K}_c$ presented in equation 6 is computed by equation 7. Adding $\mathbf{K}$ with $\mathbf{K}_c$ results in a symmetric matrix. (2)

$$\mathbf{K}_c = \int_{\mathscr{L}_c} \alpha \mathbf{N}^T \mathbf{N} t d\mathscr{L} \quad (7)$$

For a three-node element, equation 7 results in equation 8, where L is the length of the boundary where convection is present. This is because only the lower right corner presents the nodes where there is convection.

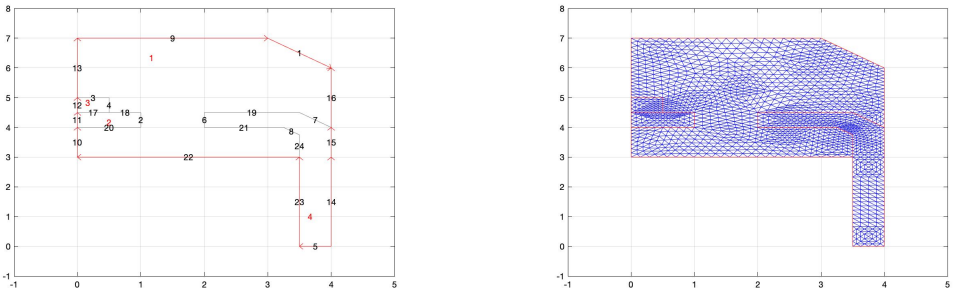
$$\mathbf{K}_c = \int_{\mathscr{L}_c} \alpha \mathbf{N}^T \mathbf{N} t d\mathscr{L} = \frac{\alpha L}{6} \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} \tag{8}$$

The integral term seen in equation 6 is for a three-node element calculated with equation 9, where L is the length of the boundary where convection is present. (3)

$$T_\infty \int_{\mathscr{L}_c} \mathbf{N}^T \alpha t d L \mathscr{L} = \alpha T_\infty L \cdot \begin{bmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} = \mathbf{f}_0 \tag{9}$$

2.1.2 Matlab

The first step of the project was to open the PDE-tool in Matlab. In this tool, the mesh was created to look like half of the integrated circuit given in the problem, see figure 1. Only half of the integrated circuit was needed because of symmetry. The created mesh is seen in figures 2(a) and 2(b), where figure 2(a) shows the boundaries and their numbers.



(a) Boundary mode in the PDE-tool.

(b) Mesh mode in the PDE-tool.

Figure 2: The figures show the mesh in the PDE-tool.

The next step of the project was to export the topology matrices p, e and t to the Matlab script. When this was done, the coordinates of the nodes were identified from the topology matrix p.

With the help of the topology matrix t, the Dof-matrix and the $Edof$-matrix were computed. These matrices were used to extract the element coordinates ex and ey using the function *coordextr* found in the CALFEM-package.

To define the boundary conditions, the edges with convection were first identified from the e-matrix. They are the edges labled 1, 9 and 16 in figure 2(a). After the edges with convection were identified, the nodes at the edges were combined with the given temperature and put together in a matrix with the initial temperature. This matrix could then be used as a matrix for all the boundary conditions.

Because of consisting of different materials, the subsections that the elements belonged to were found using the t-matrix and the value of thermal conductivity was varied. The function $flw2te$ found in the CALFEM-package was then used to calculate the element stiffness matrix as well as the element load vector. Using these, the global stiffness matrix and the global load vector were assembled. To simplify this computation, a new function called *assem_conv* was created. In this function, the length of the element was calculated. The length was then used to compute the stiffness matrix that came from conductivity as well as the boundary vector.

The function *solveq* in the CALFEM-package was used to solve the system and the result was plotted using the function *patch*. Here the element displacement matrix was found using the function *extract* from the CALFEM-package. To examine how a 25% decrease in data consumption would affect the maximum temperature, the value of Q was changed to 0.75 times the given value of Q. The temperature distributions were calculated.

2.2 Task 2

2.2.1 Theory

The finite element formulation for transient heat transfer is derived from the general form for the heat transfer equation:

$$c\rho\dot{T} + div(\mathbf{q}) - Q = 0 \tag{10}$$

where c is specific heat, ρ is density, T is temperature and $\mathbf{q}$ is the flux vector. Equation 10 is then multiplied by an arbitrary weight function, integrated over the body and the Green-Gauss theorem is used. Adopting the Galerkin approximation (choosing the weight function as $v(x) = \mathbf{N}(\mathbf{x})\mathbf{c}$), the Fourier law ($\mathbf{q} = -\mathbf{D}\nabla\ \mathbf{T}$) and choosing $T = \mathbf{Na}$ results in equation 11.

$$\mathbf{C\dot{a}} + \mathbf{Ka} = \mathbf{f}_b + \mathbf{f}_l \tag{11}$$

where $\mathbf{C}$ is computed with equation 12, $\mathbf{K}$ with equation 13, $\mathbf{f}_l$ with equation 14 and $\mathbf{f}_b$ with equation 15. In equations 12-15, V is volume, $\mathbf{N}$ is the shape functions, c is specific heat, ρ is density, $\mathbf{B}$ is the derivative of the shape functions, $\mathbf{D}$ is the constitutive matrix, Q is the heat and q_n is the flux. (4)

$$\mathbf{C} = \int_V \mathbf{N}^T c\rho\mathbf{N}dV \tag{12}$$

$$\mathbf{K} = \int_V \mathbf{B}^T\mathbf{DB}dV \tag{13}$$

$$\mathbf{f}_l = \int_V \mathbf{N}^T QdV \tag{14}$$

$$\mathbf{f}_b = -\int_S \mathbf{N}^T q_n dS \tag{15}$$

Equation 11 has a time derivative that can be approximated by equation 16. t_{n+1} and t_n are times at two instants, the time steps are close in time.

$$\dot{\mathbf{a}} = \frac{\mathbf{a}(t_{n+1}) - \mathbf{a}(t_n)}{t_{n+1} - t_n} \tag{16}$$

Approximating $\mathbf{a}$ with equation 17 and $\mathbf{f}$ with equation 18 makes it possible to reformulate equation 11 as equation 19. θ is unknown but often chosen as 1 due to that $\theta = 1$ gives an implicit scheme that is stable. (5)

$$\mathbf{a} = \theta \mathbf{a}_{n+1} + (1-\theta)\mathbf{a}_n \tag{17}$$

$$\mathbf{f} = \theta \mathbf{f}_{n+1} + (1-\theta)\mathbf{f}_n \tag{18}$$

$$\mathbf{C}\frac{\mathbf{a}(t_{n+1}) - \mathbf{a}(t_n)}{t_{n+1} - t_n} + \mathbf{K}(\theta \mathbf{a}_{n+1} + (1-\theta)\mathbf{a}_n) = \theta \mathbf{f}_{n+1} + (1-\theta)\mathbf{f}_n \tag{19}$$

2.2.2 Matlab

Task 2 was based on the script written for Task 1. Therefore, the first step of solving this task was to copy the script from Task 1. Since neither the stiffness matrix nor the force vector were time dependent, only the C-matrix had to be computed before the time-stepping method could be used.

The function *plantml* was used to compute the element C^e-matrix. This was done taking into account that the input variable x changed depending on what subdomain the element was in. Using the element C^e-matrix, the C-matrix was assembled using a similar method to the one presented in the function *assem_conv* in Task 1 (see Appendix, Task 1).

A for-loop was created to solve the implicit time-stepping problem presented in equation 19. The resulting temperature distributions were plotted using the function *patch* and calculated. In the end, the quality was reduced by changing the value of Q to 0.75 times the given value of Q.

2.3 Task 3

2.3.1 Theory

When a body is under some kind of force this usually produces stresses and strains which leads to displacements. The stresses can be defined by a stress tensor, $\mathbf{S}$, according to:

$$\mathbf{S} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} \tag{20}$$

In equation 20, σ_{xy} refers to a stress in the y-direction for a surface with a normal vector in the x-direction. It can be proven that $\mathbf{S}$ is a symmetric matrix, which means only 6 elements in the matrix in equation 20 are unique. This leads to an equilibrium condition for stresses in a body, see *Introduction to the finite element method* pages 237-241 for a more in depth description. The equilibrium condition is presented in equation 21.

$$\widetilde{\boldsymbol{\nabla}}^T \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0} \tag{21}$$

where $\widetilde{\boldsymbol{\nabla}}^T$ is a 3x6 matrix differential operator, $\boldsymbol{\sigma}$ is a 6x1 matrix with the unique stress components and $\mathbf{b}$ is a 3x1 matrix with the body forces. (2)

If the displacements in a body are defined as $\mathbf{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T$ the strains, $\boldsymbol{\epsilon}$, are related to the displacements according to $\boldsymbol{\epsilon} = \widetilde{\boldsymbol{\nabla}}\mathbf{u}$. For the special case of plane strain the displacements can be described according to $u_x = u_x(x, y)$, $u_y = u_y(x, y)$ and $u_z = 0$. This gives the same relation between $\boldsymbol{\epsilon}$ and $\mathbf{u}$ but with some changes in definition, see *Introduction to the finite element method* page 247. This also results in only two degrees of freedom instead of three. (2)

When assuming plain strain as well as isotropy, equation 22 is obtained, where the stress $\boldsymbol{\sigma}$, the constitutive matrix $\mathbf{D}$, the strain $\boldsymbol{\epsilon}$ and the thermal strain $\boldsymbol{\epsilon_0}$ are defined in equations 23-26.

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\epsilon} - \mathbf{D}\boldsymbol{\epsilon}_0 \tag{22}$$

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix} \tag{23}$$

$$\mathbf{D} = \frac{E}{(1+v)(1-2v)} \cdot \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix} \tag{24}$$

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{xy} \end{bmatrix} \tag{25}$$

$$\boldsymbol{\epsilon}_0 = (1+\nu)\alpha\Delta T \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \tag{26}$$

When only considering the thermal stress of the element, $\boldsymbol{\epsilon}_0$ alone will contribute to the stress. This means that the stress that will be present initially is defined from equation 22 as $\boldsymbol{\sigma}_0 = \mathbf{D}\boldsymbol{\epsilon}_0$. Equation 22 also leads to a definition of σ_{zz}, presented in equation 27. (2)

$$\sigma_{zz} = \nu(\sigma_{xx} - \sigma_{yy}) - \alpha E \Delta T \tag{27}$$

From the equilibrium equation, see equation 21 the weak form for a two-dimensional problem can be obtained. This result can be used to make a finite element formulation of two-dimensional elasticity according to equation 28.

$$\Big(\int_{\mathbf{A}} \mathbf{B}^T\mathbf{D}\mathbf{B}t dA\Big)\mathbf{a} = \int_{\mathscr{L}_h} \mathbf{N}^T\mathbf{h}t d\mathscr{L} + \int_{\mathscr{L}_g} \mathbf{N}^T\mathbf{t}t d\mathscr{L} + \int_A \mathbf{N}^T\mathbf{b}t dA + \int_A \mathbf{B}^T\mathbf{D}\boldsymbol{\epsilon}_0 t dA \tag{28}$$

In equation 28 t is the thickness and $\mathbf{t}$ is the quantity $\mathbf{Sn}$ where $\mathbf{n}$ is the normal vector. A known value of $\mathbf{t}$ is $\mathbf{h}$. The equation can also be written as $\mathbf{K}\mathbf{a} = \mathbf{f}_b + \mathbf{f}_l + \mathbf{f}_0$ where $\mathbf{f}_b$ is made up of the first two integrals. (2)

There are multiple ways of calculating the stresses that an element is exposed to and one of these is the effective von Mises stress. The effective von Mises stress is derived from Hooke's law ($\sigma = E\epsilon$) together with strain energy. To calculate the effective von Mises stress per element, equation 29 is used.

$$\sigma_{eff} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2 - \sigma_{xx}\sigma_{yy} - \sigma_{xx} + \sigma_{zz} - \sigma_{yy}\sigma_{zz} + 3\tau_{xy}^2 + 3\tau_{xz}^2 + 3\tau_{yz}^2} \tag{29}$$

In equation 29 σ stands for the normal stress and τ stands for the shear stress.

When working with 2D-problems, $\tau_{xz} = \tau_{yz} = 0$. To get the von Mises stress per element instead of per node, the approximation that the stress is the mean value of the stresses in the elements connected to a node can be made. (1)

2.3.2 Matlab

The first thing that was done was to write down all given values of importance to this task. In the same way as in the first two tasks, the topology matrices p, e and t were exported to the Matlab script from the PDE-tool and p was used to get the coordinates of the nodes. The temperature distributions from task 1 were also saved.

The next step was to compute the Dof-matrix and the $Edof$-matrix, this time taking into account that each node had two degrees of freedom instead of one (because no temperature distribution was investigated in this task). With the help of these matrices, the element coordinates ex and ey could be extracted.

Given the information that the circuit had the conditions $u_x = u_y = 0$ at the bottom as well as $u_x = 0$ at the symmetry line, the boundary conditions were defined. Similarly to task 1, the stiffness matrix $\mathbf{K}$ as well as $\mathbf{f}_0$ were then assembled, the latter using the function $plantf$ from the CALFEM-package. Solving equation 28 and extracting the element displacements was done before calculating what the displaced element coordinates would look like.

With the function $plants$ from the CALFEM-package, the effective von Mises stress per element (see equation 29) was calculated. The effective von Mises stress per node was then approximated. To be able to visualize the resulting stresses, the function $extract$ from the CALFEM-package was used. The resulting stress distributions and the displacement fields were then presented in plots. After this, the peak stresses were identified, both in sizes and positions. The last thing that was done was reducing the quality by 25%.

3 Results

The results for each task are presented below.

3.1 Task 1

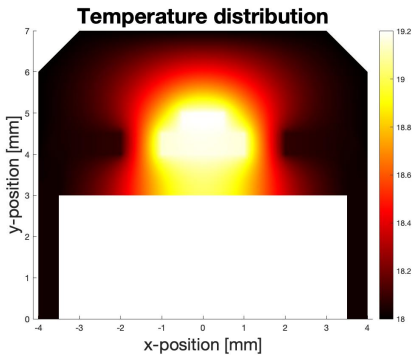
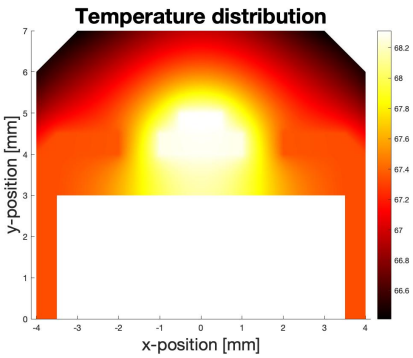
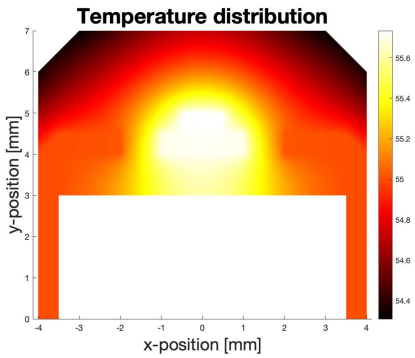


Figure 3: The figure shows the temperature distribution without convection and the temperature at the boundary set to T_∞ = 18 °C.



(a) 100 percent.



(b) 75 percent.

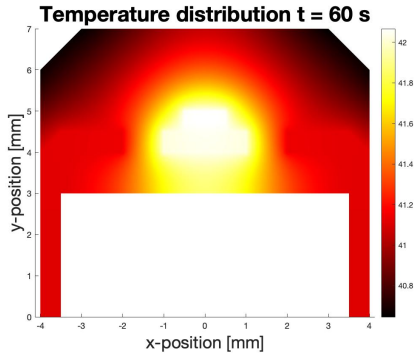
Figure 4: The figures show the temperature distribution in the circuit board with convection for a 25 percent reduction of Q.

Table 3: Maximum temperatures.

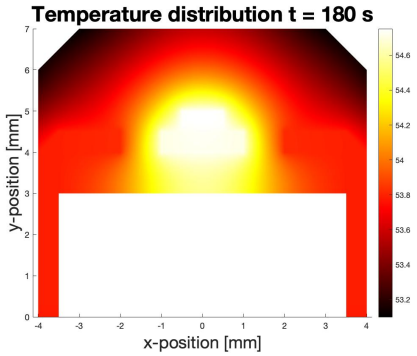
Data quality	Temperature [°C]
Full quality	68.31
Reduced quality	55.73

The difference in maximum temperature because of a reduced data quality is given by $\Delta T_{max} = 68.3111 - 55.7333 = 12.5778 \approx 12.58$ °C, see figure 4.

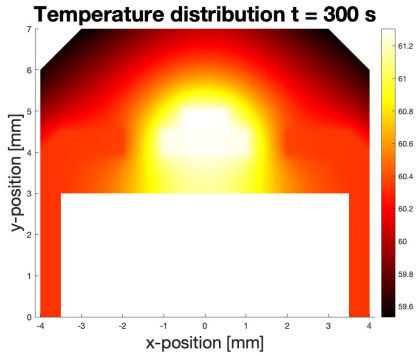
3.2 Task 2



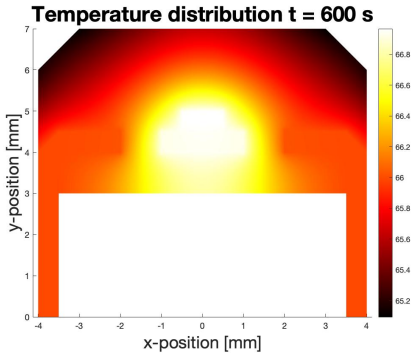
(a) 1 minute.



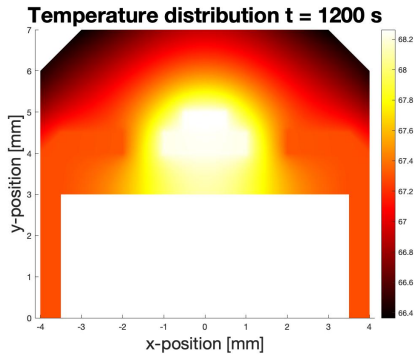
(b) 3 minutes.



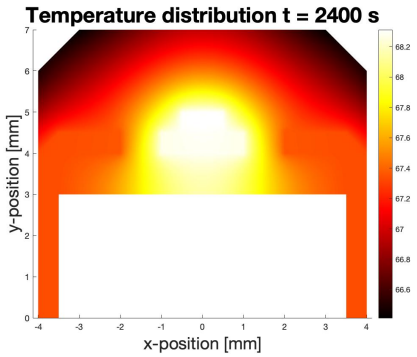
(c) 5 minutes.



(d) 10 minutes.



(e) 20 minutes.



(f) 40 minutes.

Figure 5: The figures show the temperature distribution in the circuit board after different amounts of time.

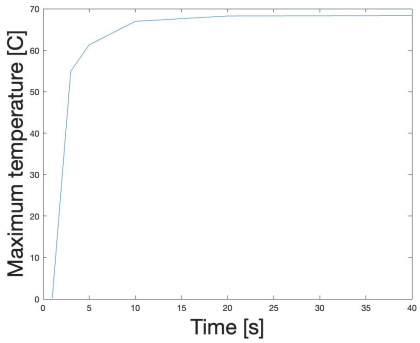
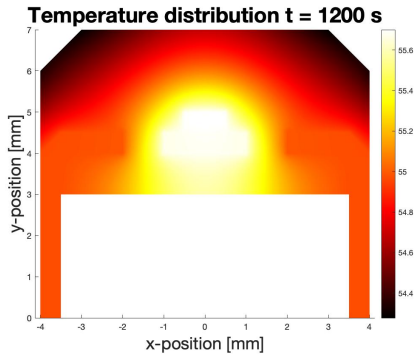
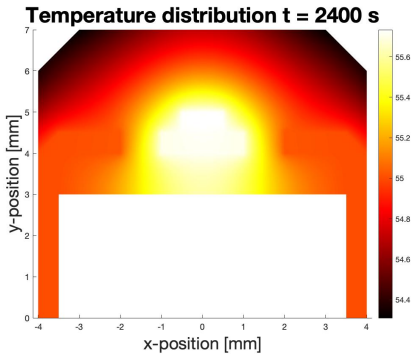


Figure 6: The figure shows the maximum temperature in the circuit board as a function of time.



(a) 20 minutes.



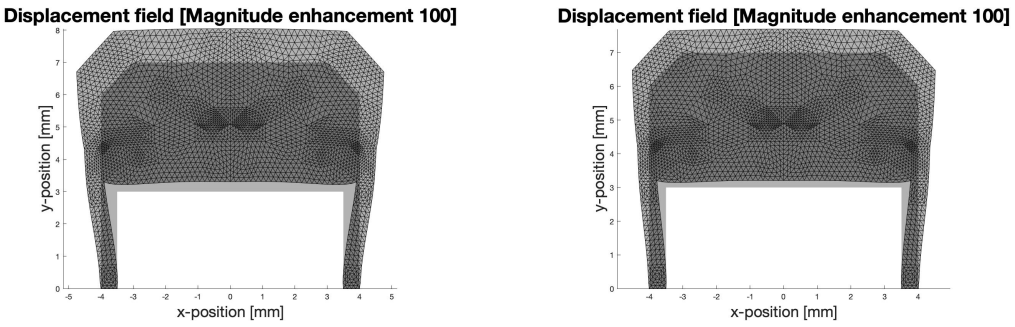
(b) 40 minutes.

Figure 7: The figures show the temperature distribution in the circuit board after different amounts of time for a 25 percent reduction of Q.

Table 4: Maximum temperatures for transient heat flow, taken at 20 and 40 minutes.

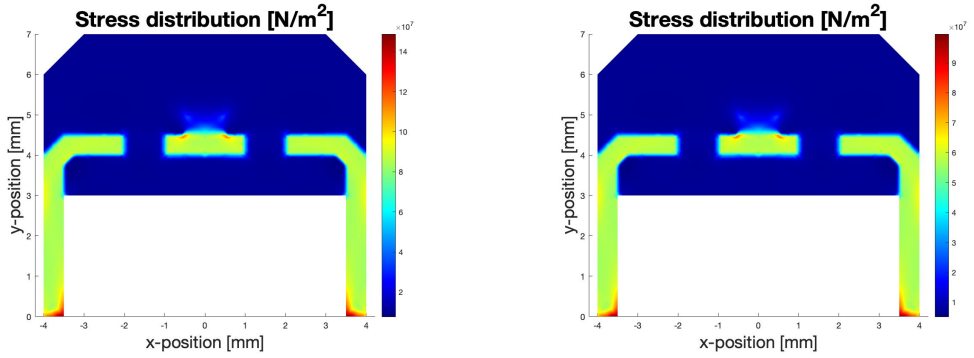
Data quality	Temperature [°C]
Full quality 20 min	68.26
Full quality 40 min	68.31
Reduced quality 20 min	55.70
Reduced quality 40 min	55.73

3.3 Task 3



(a) 100 percent. (b) 75 percent.

Figure 8: The figures show the displacement (magnified 100 times) of the circuit board due to an increase in temperature for full and reduced quality.



(a) 100 percent. (b) 75 percent.

Figure 9: The figures show the stress distribution in the circuit board due to an increase in temperature for full and reduced quality.

Table 5: Maximum stress in the circuit board.

Data quality	Stress [N/m^2]	Coordinates [mm]
Full quality	$1.49 \cdot 10^8$	x=3,5 y=0
Reduced quality	$9.97 \cdot 10^7$	x=3,5 y=0

4 Discussion

Below follows a discussion of the results for the different tasks.

4.1 Task 1

Figure 3 shows that the first part of the Matlab code worked well when convection had not been added. The temperature reaches its maximum value in the silicone die and the boundary temperature at the boundaries. This picture also displays how the different values of thermal conductivity, k, makes the temperature different in different regions.

A comparison between figure 3 and figure 4(a) shows that having a set temperature as the boundary condition and using convection has a big impact on the results. Figures 4(a) and 4(b) show how the amount of supplied heat determines the temperature distribution in the circuit board. A lower heat supply gives a lower maximum temperature, see table 3.

The maximum temperatures seen in table 3 are reasonable since an integrated circuit will get warmer over time but reach a stationary solution at a temperature below 100 $^\circ$ C. The difference between the different data qualities was 12.6 $^\circ$C but because of lack of data it is impossible to determine a relationship between heat supply and variation in maximum temperature.

The chosen mesh is sufficient for answering the questions to the given task. A finer mesh would perhaps give a more exact solution but would require a lot more time for the computer to process.

4.2 Task 2

Given the small difference between the maximum temperatures in figures 5(e) and 5(f), see table 4, it is estimated that the latter has reached the stationary solution. This can also be confirmed when comparing figure 5(f) to figure 4(a). Looking over all the figures in figure 5 as well as the graph presented in figure 6, it can also be shown that the temperature changes faster in the beginning compare to the end which is reasonable. This is because the change in temperature depends on the temperature difference according to $q_n = \alpha_c(T - T_\infty)$. The same can be stated regarding the stationary solution when comparing figure 4(b) with figure 7(b). A comparison of figures 7(a) and 7(b) or table 4 also concludes that the temperature changes very little towards the end.

The chosen size of the time steps was 5 seconds (see Appendix, Task 2). A decrease of this time would provide a better solution but require more time for the computer to process. It is most important that the same size of time step is used to be able to compare the results.

4.3 Task 3

The given boundary conditions that $u_x = u_y = 0$ at the bottom of the circuit and $u_x = 0$ at the symmetry line correspond well to the plots of the displacement seen in figure 8. Since heated things expand, the result is very reasonable. A quality decrease of 25% leads to a lower temperature (see tables 3 and 4) than when having full quality. Therefore the displacement for lower quality should be less compared to full quality. Comparing figure 8(a) with figure 8(b) clearly shows that this is the case, there is more displacement when there is higher quality.

The stress distributions seen in figure 9(a) and 9(b) show that the stress differs in different parts of the circuit. Table 5 presents the coordinates for the maximum values of the stress and these are located at the bottom of the circuit. Since the boundary conditions of the circuit are that these coordinates are fix and therefore cannot move, it is very reasonable that this is where most stress occurs. It is also reasonable that the maximum stress value is lower for reduced quality because the displacement was lower.

The reason to why the copper and the silicone reach higher values of stress than the surrounding silver is because of the materials. Young's modulus is 7 GPa for silver which is very small compared to 165 GPa and 128 GPa for silicone and copper respectively. Since the stresses per node are calculated depending on where the nodes are situated, the different values of material parameters make a difference in the result, which is easily seen when looking at figures 9(a) and 9(b).

The stress in every node was taken to be the mean value of the surrounding elements. This is an approximation which gives an error in the results. This error is assumed to be negligible.

References

[1] Division of Solid Mechanics *Assignment in The Finite Element Method*. 2020.

[2] N. Ottosen and H. Petersson. *Introduction to the finite element method*. Edinburgh: Prentice Hall Europe, 1992.

[3] M. Wallin *Introdution to the Finite Element Method Exerises*. Lund University, 2013.

[4] *Finite element formulation of transient heat transfer*. URL: http://www.solid.lth.se/fileadmin/hallfasthetslara/utbildning/kurser/FHL064_FEM/transheat.pdf Date used: 2020-05-21

[5] *Time integration procedures for transient heat transfer*. URL: http://www.solid.lth.se/fileadmin/hallfasthetslara/utbildning/kurser/FHL064_FEM/transheat.pdf Date used: 2020-05-21

5 Appendix

5.1 Task 1

```
close all
clear all
clf

%%%% Given values
Tinf = 18;
Q = (5*10^7)/(10^9);
Q = 0.75*Q;
alphac = 40/(10^6);
nen = 3;
kAg = 5/10^3;
kSi = 149/10^3;
kCu = 385/10^3;
Ep = 10;

%%%% Load values from PDE-tool
t = load('t.mat').t;
p = load('p.mat').p;
e = load('e.mat').e;

%%%% Get point coordinates
coord = p';

%%%% Compute Dof and Edof
enod = t(1:3,:)';
nelm = size(enod,1);
nnod = size(coord,1);
dof = (1:nnod)';
for ie = 1:nelm
    edof(ie,:) = [ie,enod(ie,:)];
end

%%%% Get element coordinates
[ex, ey] = coordxtr(edof,coord,dof,nen);
```

```
%%%%% Check which segments that have convections
er = e([1 2 5],:);
conv_segments = [1 9 16]; % Boundary segments with T = Tinf
edges_conv = [];
for i = 1:size(er,2)
    if ismember(er(3,i),conv_segments)
        edges_conv = [edges_conv er(1:2,i)];
    end
end

%%%%% Define boundary conditions
points_conv = [edges_conv(1,:) edges_conv(2,:)];
unique_nodes = unique(points_conv);
bc = [unique_nodes',Tinf*ones(length(unique_nodes),1)];

%%%%% Compute K and fl
K = zeros(nnod);
fl = zeros(nnod,1);
for elnr = 1:nelm
    eq = 0;
    if t(4,elnr) == 1
        k = kAg;
    elseif t(4,elnr) == 2
        k = kCu;
    elseif t(4,elnr) == 3
        k = kSi;
        eq = Q; % Q comes from Si, subdomain 3
    elseif t(4,elnr) == 4
        k = kCu;
    end
    D = k*eye(2);
    [Ke, fle] = flw2te(ex(elnr,:), ey(elnr,:), Ep, D, eq);
    indx = edof(elnr,2:end);
    K(indx,indx) = K(indx,indx) + Ke;
    fl(indx) = fl(indx) + fle;
end

%%%%% Compute Kc and fb
[Kc, fb] = assem_conv(alphac, edges_conv, p, size(K,1), Tinf);
```

```
%%%%% Get F and the new K
F = fl + fb*Ep;
Knew = K + Kc*Ep;

%%%%% Different solutions with or without boundary conditions
a = solveq(Knew,F, bc);
a = solveq(Knew,F);

%%%%% Plot the temperature distribution over the entire circuit
et = extract(edof,a);
ex = [-flip(ex); ex];
ey = [flip(ey); ey];
et = [flip(et); et];
patch(ex',ey',et','EdgeColor','none')
title('Temperature distribution','fontsize',25)
colormap(hot);
colorbar;
xlabel('x-position [mm]','fontsize',20)
ylabel('y-position [mm]','fontsize',20)
axis equal

%%%%% Calculate maximum temperature
max(a)

%%%%% Function for assembling Kc and fb
function [Kc,fb]=assem_conv(alpha, econv, p, sizeKc, Tinf)

Kc = zeros(sizeKc);
fb = zeros(sizeKc, 1);

%%%%% Compute the length of each element and assemble Kc and fb
for i = 1:length(econv)
    x1 = p(1,econv(1,i));
    x2 = p(1,econv(2,i));
    y1 = p(2,econv(1,i));
    y2 = p(2,econv(2,i));
    L = sqrt((x2-x1)^2+(y2-y1)^2);

    Kc(econv(:,i),econv(:,i)) = Kc(econv(:,i),econv(:,i)) ...
        + (L*alpha/6).*[2 1; 1 2];
```

```
    fb(econv(:,i),1) = fb(econv(:,i),1) + alpha*Tinf*L.*[1/2; 1/2];
end
```

5.2 Task 2

```
%%%%% Adding to existing code from Task 1

%%%%% Given values
Tempstart = 30;
tfirst = 0;
tend = 20*60;
deltat = 5;
rhoAg = 2500/10^9;
rhoSi = 2530/10^9;
rhoCu = 8930/10^9;
cpAg = 1000;
cpSi = 703;
cpCu = 386;

%%%%% Compute the C-matrix
C = zeros(nnod);
for i = 1:length(t)
    if t(4,i) == 1
        rho = rhoAg;
        cp = cpAg;
    elseif t(4,i) == 2
        rho = rhoCu;
        cp = cpCu;
    elseif t(4,i) == 3
        rho = rhoSi;
        cp = cpSi;
    elseif t(4,i) == 4
        rho = rhoCu;
        cp = cpCu;
    end
    x = cp*rho;
    Ce = Ep.*plantml(ex(i,:),ey(i,:),x);
    C(t(1:3,i)',t(1:3,i)') = C(t(1:3,i)',t(1:3,i)') + Ce;
end
```

```
%%%% Implicit time-stepping
aold = Tempstart*ones(length(K),1);
for i = tfirst:deltat:tend
    A = C + deltat.*Knew;
    B = C*aold + deltat.*F;
    anew = A\B;
    aold = anew;
end

%%%% Plot the temperature distribution over the entire circuit
et = extract(edof,anew);
ex = [-flip(ex); ex];
ey = [flip(ey); ey];
et = [flip(et); et];
patch(ex',ey',et','EdgeColor','none')
title('Temperature distribution t = 1200 s','fontsize',25)
colormap(hot);
colorbar;
xlabel('x-position [mm]','fontsize',20)
ylabel('y-position [mm]','fontsize',20)
axis equal

%%%% Calculate maximum temperature
max(anew)

%%%% Plot the maximum temperatures as a function of time
x = [1 3 5 10 20 40];
y = [42.0667 54.7517 61.3055 66.9669 68.2616 68.3110];
f = polyfit(x, y, 10);
plot(x,polyval(f,x))
xlabel('Time [s]','fontsize',25)
ylabel('Maximum temperature [C]','fontsize',25)
```

5.3 Task 3

```
close all
clear all
clf
```

```
%%%%% Given values

nen = 3;
TempStart = 30;
Ep = [2 10];
EAg = 7*10^9;
ECu = 128*10^9;
ESi = 165*10^9;
vAg = 0.3;
vCu = 0.36;
vSi = 0.22;
alphaAg = 4*10^(-5);
alphaCu = 17.6*10^(-6);
alphaSi = 2.6*10^(-6);

%%%%% Load values from PDE-tool
t = load('t.mat').t;
p = load('p.mat').p;
e = load('e.mat').e;

%%%%% Load values from Task 1
a100 = load('a100.mat').a100;
a75 = load('a75.mat').a75;
a = a100;

%%%%% Get point coordinates
coord = p';

%%%%% Compute Dof and Edof
enod = t(1:3,:)';
nelm = size(enod,1);
nnod = size(coord,1);
dof = [(1:nnod)',(nnod+1:2*nnod)'];
for ie = 1:nelm
    edof(ie,:) = [ie dof(enod(ie,1),:), ...
        dof(enod(ie,2),:),dof(enod(ie,3),:)];
    edofold(ie,:) = [ie,enod(ie,:)];
end

%%%%% Get element coordinates
```

```
[ex, ey] = coordxtr(edof,coord,dof,nen);

%%%%% Check which segments that have ux=uy=0
er = e([1 2 5],:);
fix_segments1 = [5];
edges_fix1 = [];
for i = 1:size(er,2)
    if ismember(er(3,i),fix_segments1)
        edges_fix1 = [edges_fix1 er(1:2,i)];
    end
end

%%%%% Define boundary conditions for those segments
points_fix1 = [edges_fix1(1,:) edges_fix1(2,:)];
unique_nodes1 = unique(points_fix1);
dofbc = [];
for i = 1:length(unique_nodes1)
    dofbc = [dofbc dof(unique_nodes1(1,i),:)];
end

%%%%% Check which segments that have ux=0
fix_segments2 = [10 11 12 13];
edges_fix2 = [];
for i = 1:size(er,2)
    if ismember(er(3,i),fix_segments2)
        edges_fix2 = [edges_fix2 er(1:2,i)];
    end
end

%%%%% Define boundary conditions for those segments
points_fix2 = [edges_fix2(1,:) edges_fix2(2,:)];
unique_nodes2 = unique(points_fix2);
for i = 1:length(unique_nodes2)
    dofbc = [dofbc dof(unique_nodes2(1,i),1)];
end

%%%%% Final boundary conditions
bc = [dofbc' zeros(length(dofbc),1)];

%%%%% Compute K and f0
```

```
K = zeros(2.*nnod);
f0 = zeros(2.*nnod,1);
for elnr = 1:nelm
    if t(4,elnr) == 1
        E = EAg;
        v = vAg;
        alpha = alphaAg;
    elseif t(4,elnr) == 2
        E = ECu;
        v = vCu;
        alpha = alphaCu;
    elseif t(4,elnr) == 3
        E = ESi;
        v = vSi;
        alpha = alphaSi;
    elseif t(4,elnr) == 4
        E = ECu;
        v = vCu;
        alpha = alphaCu;
    end
    D = E/((1+v)*(1-2*v)).*[1-v v 0; v 1-v 0; ...
        0 0 (1/2).*(1-2*v)];
    deltaT = (1/3).*((a(t(1,elnr),1)-TempStart) + ...
        (a(t(2,elnr),1)-TempStart) + (a(t(3,elnr),1)-TempStart));
    e0 = (1+v)*alpha*deltaT.*[1; 1; 0];
    Es = D*e0;

    Ke = plante(ex(elnr,:), ey(elnr,:), Ep, D);
    f0e = plantf(ex(elnr,:), ey(elnr,:), Ep, Es');

    indx = [edof(elnr,2:3) edof(elnr,4:5) edof(elnr,6:7)];
    K(indx,indx) = K(indx,indx) + Ke;
    f0(indx) = f0(indx) + f0e;
end

%%%% Get F
F = f0*Ep(1,2);

%%%% Solve the problem and extract element displacements
u = solveq(K*Ep(1,2),F,bc);
```

```
ed = extract(edof,u);

%%%% Calculate displaced coordinates
mag = 100;
exd = ex + mag*ed(:,1:2:end);
eyd = ey + mag*ed(:,2:2:end);

%%%% Compute the element stresses
es = zeros(nelm, 3);
Seff_el = zeros(nelm,1);
for elnr = 1:nelm
    if t(4,elnr) == 1
        E = EAg;
        v = vAg;
        alpha = alphaAg;
    elseif t(4,elnr) == 2
        E = ECu;
        v = vCu;
        alpha = alphaCu;
    elseif t(4,elnr) == 3
        E = ESi;
        v = vSi;
        alpha = alphaSi;
    elseif t(4,elnr) == 4
        E = ECu;
        v = vCu;
        alpha = alphaCu;
    end
    D = E/((1+v)*(1-2*v)).*[1-v v 0; v 1-v 0; ...
        0 0 (1/2).*(1-2*v)];
    deltaT = (1/3).*((a(t(1,elnr),1)-TempStart) + ...
        (a(t(2,elnr),1)-TempStart) + (a(t(3,elnr),1)-TempStart));
    e0 = (1+v)*alpha*deltaT.*[1; 1; 0];
    sig0 = D*e0;
    es = plants(ex(elnr,:), ey(elnr,:), Ep, D, ed(elnr,:));

    sigxx = es(1,1) - sig0(1,1);
    sigyy = es(1,2) - sig0(2,1);
    sigzz = v*(sigxx + sigyy)-alpha*E*deltaT;
    tauxy = es(1,3) - sig0(3,1);
```

```
    tauxz = 0;
    tauyz = 0;

    Seff_el(elnr, 1) = sqrt(sigxx^2 + sigyy^2 + sigzz^2 - sigxx*...
        sigzz - sigyy*sigzz + 3*tauxy^2 + 3*tauxz^2 + 3*tauyz^2);
end

%%%% Approximate the node stresses
for i = 1:2*nelm
    [c0,c1] = find(edofold(:,2:4) == i);
    Seff_nod(i,1) = sum(Seff_el(c0))/size(c0,1);
end

%%%% Plot the stress distribution and the displacement field
et = extract(edofold, Seff_nod);
et = [flip(et); et];
ex = [-flip(ex); ex];
ey = [flip(ey); ey];
exd = [-flip(exd); exd];
eyd = [flip(eyd); eyd];

figure()
patch(ex',ey', et','EdgeColor','none')
title('Stress distribution [N/m^2]','fontsize',25)
colormap(jet(5000));
colorbar;
xlabel('x-position [mm]','fontsize',20)
ylabel('y-position [mm]','fontsize',20)
axis equal

figure()
patch(ex',ey',[0 0 0],'EdgeColor','none','FaceAlpha',0.3)
hold on
patch(exd',eyd',[0 0 0],'FaceAlpha',0.3)
axis equal
title('Displacement field [Magnitude enhancement 100]', ...
    'fontsize', 25)
xlabel('x-position [mm]','fontsize',20)
ylabel('y-position [mm]','fontsize',20)
axis equal
```

```
%%%% Calculate the maximum stress, size and placement
[row, col] = find(ismember(et, max(et(:))));
maxstress = et(row(1),col(1))
for i = 1:length(row)
    exmax(i) = ex(row(i),col(i));
    eymax(i) = ey(row(i),col(i));
end
```