Lunds Universitet
Lunds Tekniska Högskola

FHLF25 - Finite Element Method and Introduction to Strength of Materials

Assignment in The Finite Element Method, 2023

Authors:
Waldemar Kulle
Nils Thorin

May 24, 2023

LUNDS UNIVERSITET

Contents

1 Introduction

Most materials are known to expand when exposed to higher temperatures, a phenomenon utilized in various applications like bimetallic thermometers. Figure 1 illustrates another example known as a thermo-mechanical gripper. This gripper is comprised of two materials, copper and nylon. Note that the depicted structure in the figure represents only a quarter of the complete gripper, expanding it along its symmetry lines enables the result to grasp objects effectively when subjected to a thermal load represented by $q_n = h$.

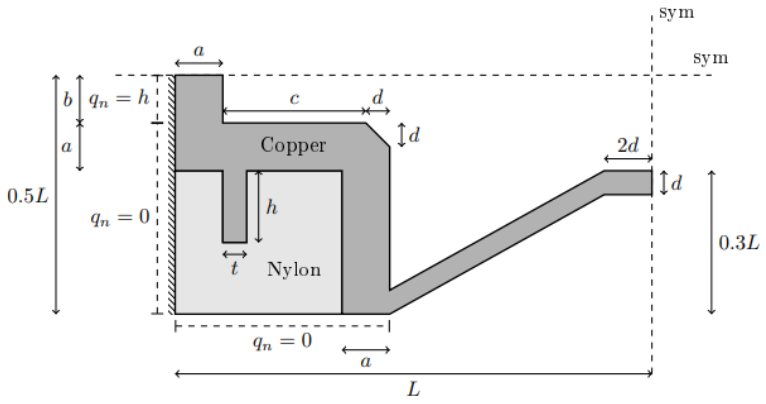


Figure 1: The geometry of our problem, image was taken from the problem formulation.

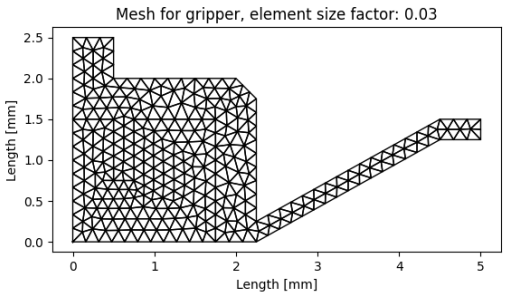
However, the thermal expansion of the materials introduces thermal stresses within the structure, which can be non-trivial due to the presence of multiple materials. To assess the performance of the gripper accurately, it is necessary to conduct a finite element analysis.

In this report, we will develop a finite element program in Python using the `CALFEM` toolbox to analyze the temperature and stress distributions in this thermo-mechanical gripper. Figure 1 provides thermal boundary conditions, denoted by a dashed line, indicating parts with specified heat flux (q_n). The remaining boundary is subjected to Newton's convection with a heat flux represented by $q_n = \alpha_c(T - T_\infty)$, where $\alpha_c = 40$ W/m^{2}K. Additionally, the material data is illustrated in figure 2. The left boundary is clamped, and the gripper is further restricted from moving out-of-plane, as we are assuming a plane strain condition.

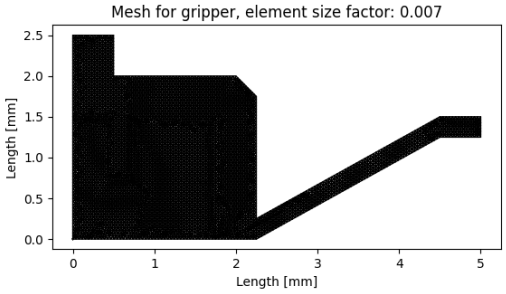
	Copper	Nylon
Young's modulus, E [GPa]	128	3.00
Poisson's ratio, ν [-]	0.36	0.39
Expansion coefficient, α [$1/K$]	$17.6 \cdot 10^{-6}$	$80 \cdot 10^{-6}$
Density, ρ [kg/m^3]	8930	1100
Specific heat, c_p [J/kg-K]	386	1500
Thermal conductivity, k [W/m-K]	385	0.26

Figure 2: Material data for copper & nylon, taken from the problem formulation.

For us to be able to analyze the mechanical load we must first determine the temperature distribution when the flux $h = 1 \cdot 10^5$ is introduced on a part of our boundary, the problem formulation also states that $T_\infty = 18°$C and we expect that our gripper assumes the same temperature as it's surroundings $T = T_\infty$ initially. Both the transient and the stationary solution is of interest.



(a) The coarse mesh.



(b) The fine mesh (the mesh lines are very close together, thus making it look like it is fully black).

Figure 3: The two different meshes used during the assignment.

Furthermore, the temperature distribution enables analysis of the mechanical loads, i.e. the von Mises stress. We assume the plane stress conditions hold, thus ε_{xx}, ε_{yy} and τ_{xy} are the only non-zero strains (note that σ_{zz} generally will be non-zero). We also have boundary conditions where we assume no deformation is possible on the left side of the gripper.

2 Procedure

2.1 Mesh

We created the mesh with the assistance of the `CALFEM` geometry module, first placing nodes at each corner of the gripper, then splines between each node, marked with what boundary condition we would be applying to create our elements. During this assignment we mainly used two finenesses, which corresponded to different *element scale factors*, meaning a lower factor created a finer mesh, see figure 3.

2.2 Derivation of weak form & FE-formulation

In order to find the temperature distribution we needed the FE-formulation of two-dimensional heat flow, and for this, the weak form of two-dimensional heat flow is required. In order to derive the weak form, we assume we can use the strong form, i.e.

$$\text{div}(t\boldsymbol{D}\nabla T) + tQ = \rho t c_p \dot{T}, \tag{1}$$

in the region A, with boundary conditions,

$$q_n = \boldsymbol{q}^T n = h, \tag{2}$$

on the part of the boundary where the flux is known (we call it $\mathcal{L}_h$),

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

on the part of the boundary where the temperature is known ($\mathcal{L}_g$). Here t is the thickness of our material, ρ is the density of the materia, c_p is the heat capacity, $\boldsymbol{D}$ is the constitutive matrix (describing how well heat flows in different directions), T is the temperature, and Q is a heat supply term. As our

problem is formulated to be strictly two-dimensional, from now on we set our $t = 1$. We also apply the constitutive relation $\boldsymbol{q} = -\boldsymbol{D}\nabla T$ for some shorter equations,

$$\operatorname{div}(\boldsymbol{q}) - Q = -\rho c_p \dot{T}. \tag{4}$$

In order to arrive at our weak form, we need to multiply (4) by an arbitrary function (called a *weight function*), $v(x, y)$, and integrate over the entire relevant area A,

$$\int_A v \operatorname{div}(\boldsymbol{q}) \,\mathrm{d}A - \int_A vQ \,\mathrm{d}A = -\int_A v\rho c_p \dot{T} \,\mathrm{d}A. \tag{5}$$

Integrating the first term by parts and applying the Green-Gauss theorem gives us,

$$\int_A v \operatorname{div}(\boldsymbol{q}) \,\mathrm{d}A = \int_{\mathcal{L}} v\boldsymbol{q}^T \boldsymbol{n} \,\mathrm{d}\mathcal{L} - \int_A (\nabla v)^T \boldsymbol{q} \,\mathrm{d}A. \tag{6}$$

We can now write equation (5) as

$$\int_A (\nabla v)^T \boldsymbol{q} \,\mathrm{d}A + \int_A v\rho c_p \dot{T} \,\mathrm{d}A = \int_{\mathcal{L}} v\boldsymbol{q}^T \boldsymbol{n} \,\mathrm{d}\mathcal{L} - \int_A vQ \,\mathrm{d}A. \tag{7}$$

Furthermore, we can use the boundary conditions (2 & 3) to split our integral over the boundary in to two parts, $\mathcal{L}_h$ where $q_n = h$, and $\mathcal{L}_g$ where $T = g$,

$$\int_A (\nabla v)^T \boldsymbol{q} \,\mathrm{d}A + \int_A v\rho c_p \dot{T} \,\mathrm{d}A = \int_{\mathcal{L}_h} vh \,\mathrm{d}\mathcal{L} + \int_{\mathcal{L}_g} vq_n \,\mathrm{d}\mathcal{L} - \int_A vQ \,\mathrm{d}A. \tag{8}$$

Especially note that $q_n = \boldsymbol{q}^T \boldsymbol{n}$, $\boldsymbol{n}$ being the boundary normal vector. Inserting the constitutive relation again gives us the weak form of two-dimensional heat flow:

$$\int_A (\nabla v)^T \boldsymbol{D}\nabla T \,\mathrm{d}A + \int_A v\rho c_p \dot{T} \,\mathrm{d}A = -\int_{\mathcal{L}_h} vh \,\mathrm{d}\mathcal{L} - \int_{\mathcal{L}_g} vq_n \,\mathrm{d}\mathcal{L} + \int_{\mathcal{L}_A} vQ \,\mathrm{d}A. \tag{9}$$

Now we're ready to start formulating our FE approach. We approximate the solution as $T = \boldsymbol{N}\boldsymbol{a}$, where $\boldsymbol{N}$ is the global shape function vector, the derivation of which, one finds on page 118 in the course book, and $\boldsymbol{a}$ is a vector containing the temperature in each node. We choose $v = \boldsymbol{N}\boldsymbol{c}$, $\boldsymbol{c}$ simply being an arbitrary constant vector, according to Galerkin's method for choice of weight function. These approximations gives us a modified weak form for two-dimensional heat flow:

$$\int_A (\nabla(\boldsymbol{N}\boldsymbol{c}))^T \boldsymbol{D}\nabla(\boldsymbol{N}\boldsymbol{a}) \,\mathrm{d}A + \int_A \boldsymbol{N}\boldsymbol{c}\rho c_p \boldsymbol{N}\dot{\boldsymbol{a}} \,\mathrm{d}A = -\int_{\mathcal{L}_h} \boldsymbol{N}\boldsymbol{c}h \,\mathrm{d}\mathcal{L} - \int_{\mathcal{L}_g} \boldsymbol{N}\boldsymbol{c}q_n \,\mathrm{d}\mathcal{L} + \int_{\mathcal{L}_A} \boldsymbol{N}\boldsymbol{c}Q \,\mathrm{d}A. \tag{10}$$

Especially note that since $v = \boldsymbol{N}\boldsymbol{c}$ and $v = v^T$ we have $\boldsymbol{N}\boldsymbol{c} = \boldsymbol{c}^T\boldsymbol{N}^T$. We also introduce $\boldsymbol{B} = \nabla\boldsymbol{N}$, implementing this gives us,

$$\boldsymbol{c}^T \left(\left(\int_A \boldsymbol{B}^T \boldsymbol{D}\boldsymbol{B} \,\mathrm{d}A \right) \boldsymbol{a} + \left(\int_A \boldsymbol{N}^T \rho c_p \boldsymbol{N} \,\mathrm{d}A \right) \dot{\boldsymbol{a}} + \int_{\mathcal{L}_h} \boldsymbol{N}h \,\mathrm{d}\mathcal{L} + \int_{\mathcal{L}_g} \boldsymbol{N}q_n \,\mathrm{d}\mathcal{L} - \int_{\mathcal{L}_A} \boldsymbol{N}Q \,\mathrm{d}A \right) = 0. \tag{11}$$

If we choose $\boldsymbol{c}^T$ equal to our sum of integrals in (11), then we can interpret this equation as an inner product $\langle \boldsymbol{c}, \boldsymbol{c} \rangle = 0$, which means that our $\boldsymbol{c}$, and therefore, our sum of integrals, must equal zero, from the definition of an inner product. Thus, we get our FE form

$$\boldsymbol{K}\boldsymbol{a} + \boldsymbol{C}\dot{\boldsymbol{a}} = \boldsymbol{f}, \tag{12}$$

with

$$\boldsymbol{K} = \int_A \boldsymbol{B}^T \boldsymbol{D}\boldsymbol{B}\,\mathrm{d}A, \tag{13}$$

$$\boldsymbol{C} = \int_A c_p \rho \boldsymbol{N}^T \boldsymbol{N}\,\mathrm{d}A, \tag{14}$$

$$\boldsymbol{f}_b = -\int_{\mathcal{L}_h} \boldsymbol{N}^T h\,\mathrm{d}\mathcal{L} - \int_{\mathcal{L}_g} \boldsymbol{N}^T q_n\,\mathrm{d}\mathcal{L}, \tag{15}$$

$$\boldsymbol{f}_l = \int_A \boldsymbol{N}^T Q\,\mathrm{d}A. \tag{16}$$

2.3 Stationary temperature distribution

According to our problem formulation, there's no boundary where the temperature is known, so the integral over $\mathcal{L}_g$ is redundant, we don't have any internal heat supply either, thus $Q = 0$, and furthermore, $\boldsymbol{f}_l = 0$. Also, as we only seek the stationary temperature for this first problem, $\dot{\boldsymbol{a}} = 0$ shows us this, i.e.

$$\boldsymbol{K}\boldsymbol{a} = \boldsymbol{f}, \tag{17}$$

with

$$\boldsymbol{K} = \int_A \boldsymbol{B}^T \boldsymbol{D}\boldsymbol{B}\,\mathrm{d}A, \tag{18}$$

$$\boldsymbol{f} = -\int_{\mathcal{L}_h} \boldsymbol{N}^T h\,\mathrm{d}\mathcal{L}. \tag{19}$$

However, according to the problem formulation, we in actuality have three types of boundaries. Namely, where the flux is zero (which just gives us a zero-term, so we don't have to amend this), where the flux is $h = 10^5$ W/m^2 ($\mathcal{L}_h$), and where we have Newton convection ($q_n = \alpha_c(\boldsymbol{T} - T_\infty)$). Notice we haven't taken convection into account in our current $\boldsymbol{f}$, amending this adds the term

$$-\int_{\mathcal{L}_c} \boldsymbol{N}^T \alpha_c(\boldsymbol{T} - T_\infty)\,\mathrm{d}\mathcal{L}, \tag{20}$$

where $T_\infty = 18°$C according to the problem formulation. We split this into two terms, also inserting $\boldsymbol{T} = \boldsymbol{N}\boldsymbol{a}$ according to our past approximation and arrive at our final FE form for the stationary problem:

$$\left(\int_A \boldsymbol{B}^T \boldsymbol{D}\boldsymbol{B}\,\mathrm{d}A + \alpha_c \int_{\mathcal{L}_c} \boldsymbol{N}^T \boldsymbol{N}\,\mathrm{d}\mathcal{L}\cdot\right)\boldsymbol{a} = -h\int_{\mathcal{L}_h} \boldsymbol{N}^T\,\mathrm{d}\mathcal{L} - \alpha_c T_\infty \int_{\mathcal{L}_c} \boldsymbol{N}^T\,\mathrm{d}\mathcal{L}. \tag{21}$$

With our mesh grid & elements constructed, and the FE form derived, we can start constructing our element stiffness matrices $\boldsymbol{K}^e$ (the contribution from the first integral on the left hand side in equation (21)) with the `CALFEM` function `flw2te` which builds $\boldsymbol{K}^e$ from the coordinates of each element and our $\boldsymbol{D}$. For the $\boldsymbol{D}$ we assume, according to the given material data, that copper & nylon is isotropic, i.e. we can write $\boldsymbol{D} = k\boldsymbol{I}$, k being the thermal conductivity for each element respectively.

The second integral on the left hand side, so-called $\boldsymbol{K}_c$ requires manual calculations. We start by calculating our element shape function matrix as

$$\boldsymbol{N}^e = \begin{bmatrix} -\frac{1}{L}(x - x_2) & \frac{1}{L}(x - x_1) \end{bmatrix} \tag{22}$$

with $x_1 = 0$ & $x_2 = L$ (length between nodes). We can now write

$$\boldsymbol{K}_c^e = \alpha_c \int_{\mathcal{L}_c} \boldsymbol{N}^{e^T} \boldsymbol{N}^e\,\mathrm{d}\mathcal{L} = \frac{\alpha_c}{L^2}\int_0^L \begin{bmatrix} L^2 - 2Lx + x^2 & Lx - x^2 \\ Lx - x^2 & x^2 \end{bmatrix}\mathrm{d}x = \cdots = \frac{\alpha_c L}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}. \tag{23}$$

These matrices $\boldsymbol{K}^e$ & $\boldsymbol{K}_c^e$, when iterated over all elements we assembled into the global stiffness matrix $\boldsymbol{K}$ with the `CALFEM` function `assem`.

Now, we construct our $\boldsymbol{f}$, which consists of two very similar integrals. We calculate a general

$$\int_{\mathcal{L}} \boldsymbol{N}^{e^T} \, \mathrm{d}\mathcal{L} = \frac{1}{L} \int_0^L \begin{bmatrix} L-x \\ x \end{bmatrix} \mathrm{d}x = \cdots = \frac{L}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{24}$$

and multiply the result with the corresponding constants depending on what boundary each element is on when we're iterating over them all. Summing up all these vectors gives us the global boundary vector $\boldsymbol{f}$, and we can subsequently solve our static FE-equation using another `CALFEM` function `solveq`, where the output (the $\boldsymbol{a}$-vector) is our temperature distribution solution.

2.4 Transient temperature distribution

For transient temperature, we constructed the $\boldsymbol{C}$-matrix in the same way as we constructed $\boldsymbol{K}$, we iterate over each element, but this time, use `plantml` instead, with the density of each element ρ multiplied with their respective specific heat c_p instead of our $\boldsymbol{D}$ as input.

Now, we want to solve for $\boldsymbol{a}$, however when solving for a transient solution we need to use a time stepping method. Thus, we use the Euler time-stepping method, approximating the time derivative $\dot{\boldsymbol{a}}$ as

$$\dot{\boldsymbol{a}} = \frac{\boldsymbol{a}_{n+1} - \boldsymbol{a}_n}{\Delta t}, \tag{25}$$

and the function value $\boldsymbol{a}$ as

$$\boldsymbol{a} = \theta \boldsymbol{a}_{n+1} + (1-\theta)\boldsymbol{a}_n. \tag{26}$$

For this to be an implicit function we choose $\theta = 1$. Inserting these in (12) gives us

$$\boldsymbol{C}\frac{\boldsymbol{a}_{n+1} - \boldsymbol{a}_n}{\Delta t} + \boldsymbol{K}\boldsymbol{a}_{n+1} = \boldsymbol{f}, \tag{27}$$

which we rewrite to

$$(\boldsymbol{C} + \Delta t \boldsymbol{K})\boldsymbol{a}_{n+1} = \boldsymbol{C}\boldsymbol{a}_n + \Delta t \boldsymbol{f}, \tag{28}$$

and solve iteratively with NumPy's `solve`, for a number of step sizes $\Delta t = 0.1$. To find at what time step our temperature had reached 90% of the stationary solutions temperature, we simply checked for each time step if any of the values of our a-vector had reached 90% of the stationary temperature, and broke the loop if they did.

2.5 Mechanical load

For the mechanical problem of calculating stress we start with the strong form of the mechanical equilibrium equation

$$\tilde{\boldsymbol{\nabla}}^T \boldsymbol{\sigma} + \boldsymbol{b} = \boldsymbol{0}, \tag{29}$$

where

$$\tilde{\boldsymbol{\nabla}}^T = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}, \quad \boldsymbol{\sigma} = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_x \\ b_y \end{bmatrix}. \tag{30}$$

Here $\boldsymbol{\sigma}$ contains the stresses, and $\boldsymbol{b}$ contains the body forces. For this assignment we were to assume plane strain, so we only have the x and y components.

Working from this strong form in a similar manner as we did for the thermal problem we acquire the weak form of the two-dimensional mechanical equation,

$$\int_A (\tilde{\boldsymbol{\nabla}}\boldsymbol{v})^T \boldsymbol{\sigma} \, \mathrm{d}A = \int_{\mathcal{L}} \boldsymbol{v}^T \boldsymbol{t} \, \mathrm{d}\mathcal{L} + \int_A \boldsymbol{v}^T \boldsymbol{b} \, \mathrm{d}A. \tag{31}$$

Here $\boldsymbol{t}$ is the traction vector along the boundary and $\boldsymbol{v}$ is an arbitrary weight function vector.

To connect this to the actual displacement $\boldsymbol{u}$, we set $\boldsymbol{u} = \boldsymbol{N}\boldsymbol{a}$, and in accordance with Galerkin's method we set $\boldsymbol{v} = \boldsymbol{N}\boldsymbol{c}$, where $\boldsymbol{c}$ is arbitrary. From here it follows that $\tilde{\boldsymbol{\nabla}}\boldsymbol{v} = \boldsymbol{B}\boldsymbol{c}$, where $\boldsymbol{B} = \tilde{\boldsymbol{\nabla}}\boldsymbol{N}$. Inserting this into (31) yields

$$\boldsymbol{c}^T\left(\int_A \boldsymbol{B}^T\boldsymbol{\sigma}\,\mathrm{d}A\right) = \boldsymbol{c}^T\left(\int_{\mathcal{L}} \boldsymbol{N}^T\boldsymbol{t}\,\mathrm{d}\mathcal{L} + \int_A \boldsymbol{N}^T\boldsymbol{b}\,\mathrm{d}A\right), \tag{32}$$

and as $\boldsymbol{c}$ was arbitrary we land in

$$\int_A \boldsymbol{B}^T\boldsymbol{\sigma}\,\mathrm{d}A = \int_{\mathcal{L}} \boldsymbol{N}^T\boldsymbol{t}\,\mathrm{d}\mathcal{L} + \int_A \boldsymbol{N}^T\boldsymbol{b}\,\mathrm{d}A. \tag{33}$$

Now to land in the FE-formulation we need to relate the stresses $\boldsymbol{\sigma}$ to the displacements $\boldsymbol{u}$. This relation is found on page 303 in the course book as

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\varepsilon} - \boldsymbol{D}\boldsymbol{\varepsilon}_0 \tag{34}$$

where $\boldsymbol{\varepsilon}$ are the strains and $\boldsymbol{\varepsilon}_0$ are the initial strains (this is also known as *Hooke's generalized law*). Using the kinematic relation $\boldsymbol{\varepsilon} = \tilde{\boldsymbol{\nabla}}\boldsymbol{u}$ together with (34) and how we defined $\boldsymbol{u}$ and $\boldsymbol{B}$ above we get the expression

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{B}\boldsymbol{a} - \boldsymbol{D}\boldsymbol{\varepsilon}_0. \tag{35}$$

This can now be inserted into (33), where we get

$$\left(\int_A \boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}\,\mathrm{d}A\right)\boldsymbol{a} = \int_{\mathcal{L}_h} \boldsymbol{N}^T\boldsymbol{h}\,\mathrm{d}\mathcal{L} + \int_{\mathcal{L}_g} \boldsymbol{N}^T\boldsymbol{t}\,\mathrm{d}\mathcal{L} + \int_A \boldsymbol{N}^T\boldsymbol{b}\,\mathrm{d}A + \int_A \boldsymbol{B}^T\boldsymbol{D}\boldsymbol{\varepsilon}_0\,\mathrm{d}A, \tag{36}$$

with the boundary conditions $\boldsymbol{t} = \boldsymbol{h}$ on $\mathcal{L}_h$ and $\boldsymbol{u} = \boldsymbol{g}$ on $\mathcal{L}_g$. We can once again write

$$\boldsymbol{K}\boldsymbol{a} = \boldsymbol{f}, \tag{37}$$

where

$$\boldsymbol{K} = \int_A \boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}\,\mathrm{d}A, \tag{38}$$

$$\boldsymbol{f}_b = \int_{\mathcal{L}_h} \boldsymbol{N}^T\boldsymbol{h}\,\mathrm{d}\mathcal{L} + \int_{\mathcal{L}_g} \boldsymbol{N}^T\boldsymbol{t}\,\mathrm{d}\mathcal{L}, \tag{39}$$

$$\boldsymbol{f}_l = \int_A \boldsymbol{N}^T\boldsymbol{b}\,\mathrm{d}A, \tag{40}$$

$$\boldsymbol{f}_0 = \int_A \boldsymbol{B}^T\boldsymbol{D}\boldsymbol{\varepsilon}_0\,\mathrm{d}A, \tag{41}$$

$$\boldsymbol{f} = \boldsymbol{f}_b + \boldsymbol{f}_l + \boldsymbol{f}_0. \tag{42}$$

Now that we have formulated the general FE-formulation, we can turn to our problem of the gripper. The first thing we can note is that both $\boldsymbol{f}_b$ and $\boldsymbol{f}_l$ will be zero, as we assume no outside forces are at work and that we have no bodily forces $\boldsymbol{b}$.

In our case the only initial strains are from thermal loads, and thus it takes the form (cf. page 254 in the course book)

$$\boldsymbol{\varepsilon}_0 = \alpha\Delta T\begin{bmatrix}1 & 1 & 0\end{bmatrix}^T, \tag{43}$$

where α is the thermal expansion coefficient and ΔT is the temperature difference from a reference temperature where the thermal strains are zero (in our case the reference temperature is 18°C). When

we have the plane strain assumption for isotropic materials it can be shown that the constitutive matrix $\boldsymbol{D}$ can be written as[1]

$$\boldsymbol{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix}, \tag{44}$$

where E is Young's modulus and ν is the Poisson ratio.

To calculate the stiffness matrix $\boldsymbol{K}$ the `CALFEM` function `plante` was used, whilst `plantf` was used to calculate $\boldsymbol{f}_0$. In the case of `plantf` $\boldsymbol{D\varepsilon}_0$ had to be calculated for each element. Looking at (43) and (44) we see that α, E and ν simply are material parameters, but ΔT had to be calculated for each element as the average increase over the initial temperature for all three nodes.

To then find the displacements, the system $\boldsymbol{Ka} = \boldsymbol{f}_0$ was solved using `solveq` after having applied the boundary conditions, namely that the displacement for all nodes along the left side is zero, and that the displacements perpendicular to the symmetry cuts are zero.

After having found the displacements we now are interested in the stress. The `CALFEM` method `plants` does exactly this, calculating the vector $\begin{bmatrix} \sigma_{xx} & \sigma_{yy} & \tau_{xy} \end{bmatrix}$ given the displacement for an element. As we have plane strain, σ_{zz} in general isn't zero, but can rather be found by (see page 256 in the course book)

$$\sigma_{zz} = \nu(\sigma_{xx} + \sigma_{yy}) - \alpha E \Delta T. \tag{45}$$

For each element the von Mises stress can now be calculated by

$$\sigma_{eff} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2 - \sigma_{xx}\sigma_{yy} - \sigma_{xx}\sigma_{zz} - \sigma_{yy}\sigma_{zz} + 3\tau_{xy}^2 + 3\tau_{xz}^2 + 3\tau_{yz}^2}, \tag{46}$$

where τ_{xz} and τ_{yz} both are zero.

[1] See for instance `https://canvas.education.lu.se/courses/22259/discussion_topics/313621`.

3 Results

3.1 Stationary temperature distribution

For problem a) we used our finer mesh and achieved the stationary temperature distribution and max temperature as seen in figure 4. As expected the temperature is higher near the top left corner where we have the thermal load and cooler on the arm.

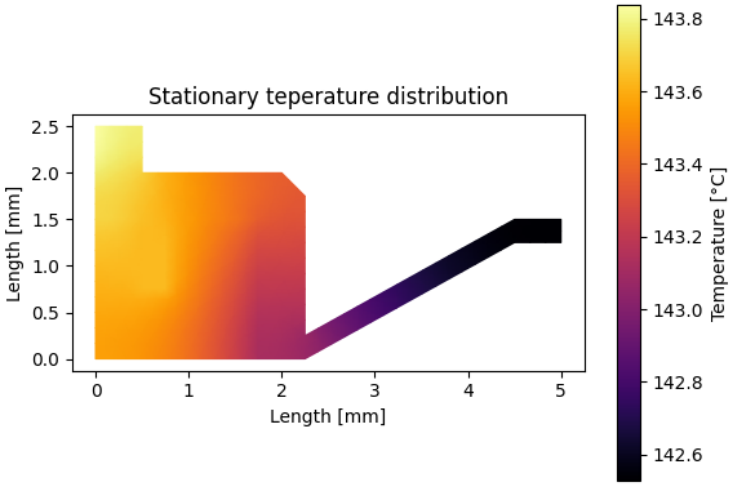


Figure 4: Stationary temperature distribution when we assume the that the gripper is in an environment with $T_\infty = 18°C$.

3.2 Transient temperature distribution

For problem b) we used our coarser mesh and found that the transient temperature reached 90% of the stationary temperature after 77.7 seconds, and 99.9% after 243.6 seconds, see figure 5 & 6.

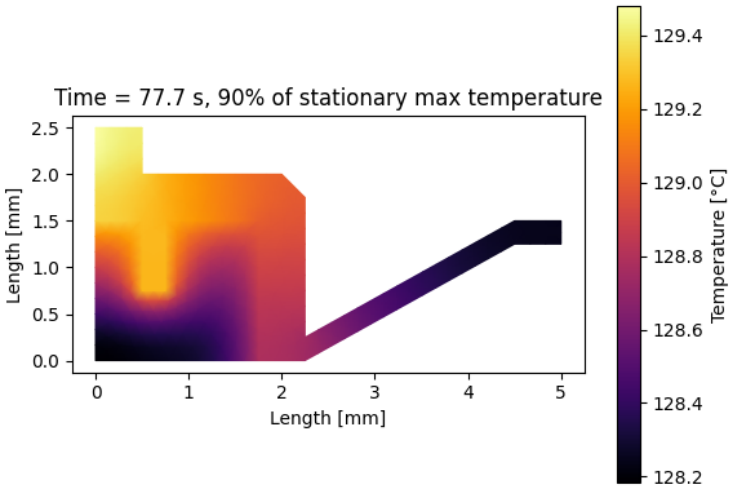


Figure 5: Transient temperature after 77.7 s, which achieves 90% of the stationary temperature.

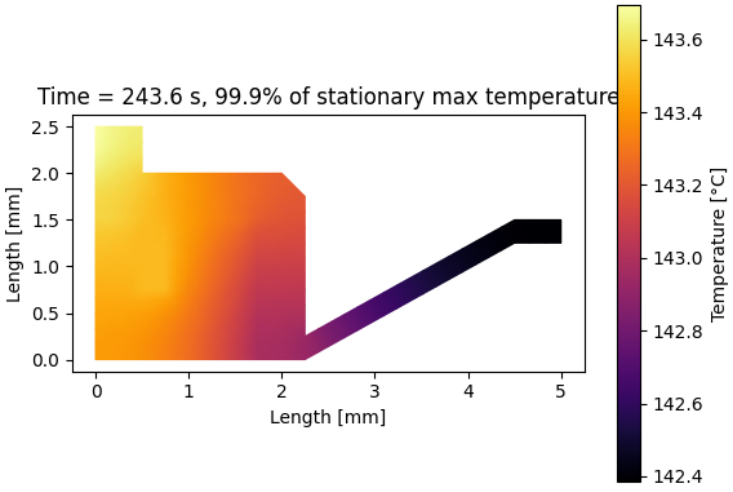
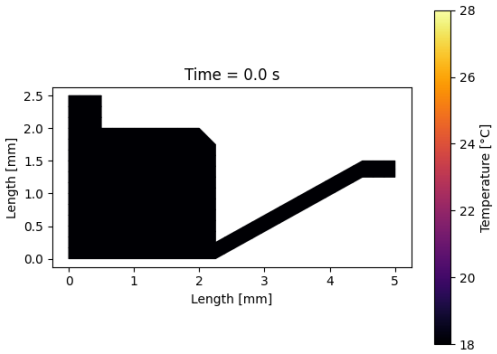
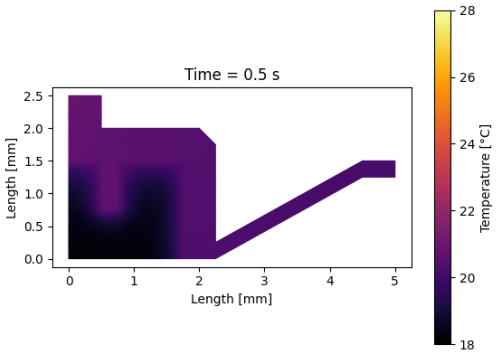


Figure 6: Transient temperature after 243.6 s, which achieves 99.9% of the stationary temperature.

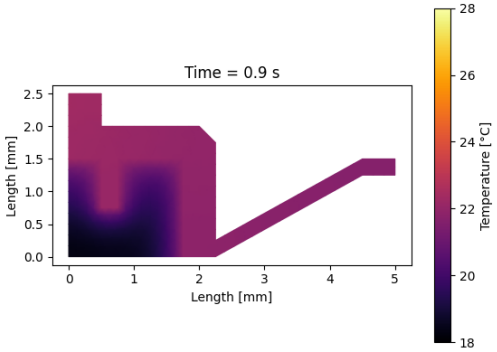
We also plotted 5 snapshots of the distribution for the first 3% of that time at equal intervals, see figure 7.



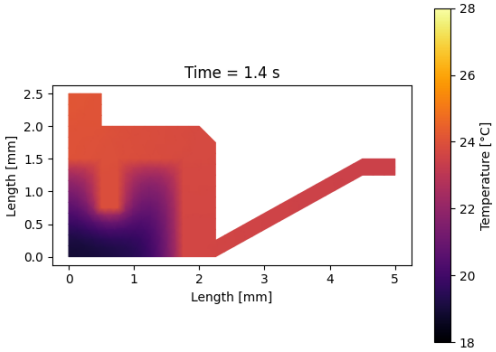
(a) Snapshot 0 of 5, $t = 0.0$s.



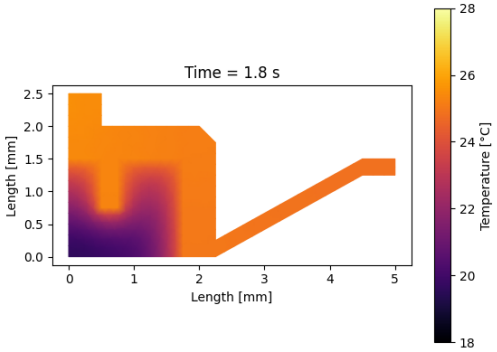
(b) Snapshot 1 of 5, $t = 0.5$s.



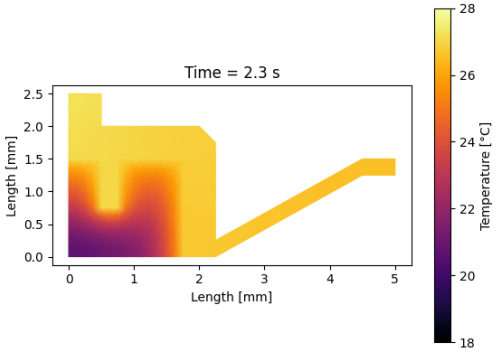
(c) Snapshot 2 of 5, $t = 0.9$s.



(d) Snapshot 3 of 5, $t = 1.4$s.



(e) Snapshot 4 of 5, $t = 1.8$s.



(f) Snapshot 5 of 5, $t = 2.3$s.

Figure 7: Transient temperature distribution of all snapshots.

3.3 Displacement and von Mises stress

As showcased in figure 8 our coarse mesh was used to showcase the deformation of the gripper. The gray mesh is the initial structure, whilst the black is post-deformation. The deformation has been magnified to make it more noticeable.

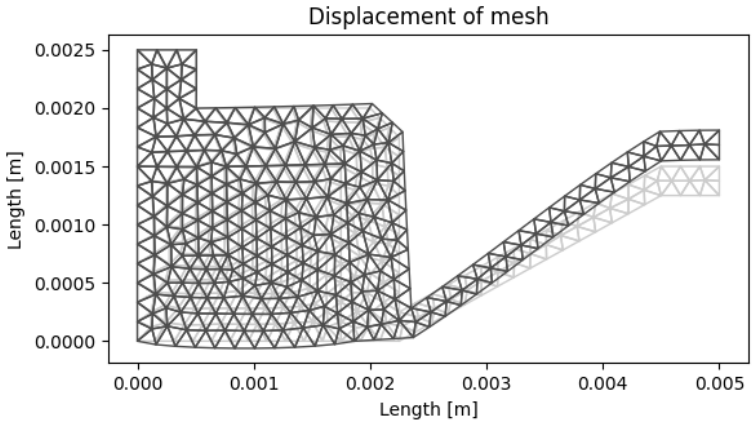


Figure 8: Plotting the initial mesh in gray and the displaced mesh in black. The shift is magnified by a factor 5 to make the deformations clear. For the same reason our coarsest mesh was used.

In the simulation of the stress the fine mesh once again was used. The stress overlaid with the deformation is shown for the full gripper in figure 9, and the nodal strain is shown in figure 10.

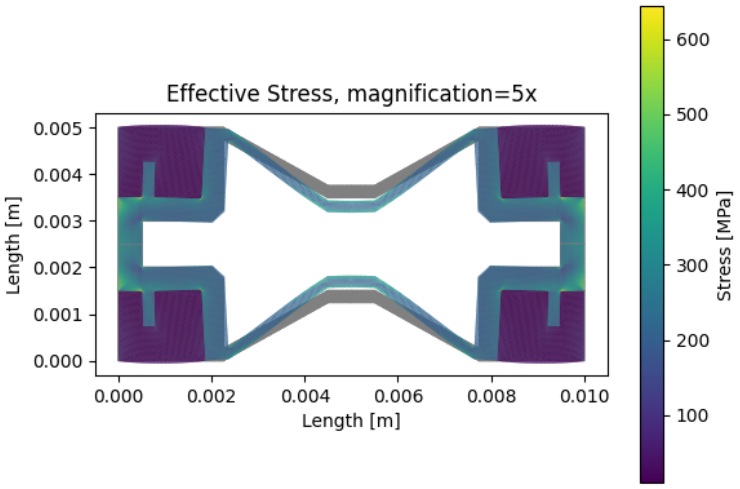


Figure 9: Deformation of the whole gripper with the elemental von Mises stresses coloured and the initial mesh in gray, with a magnification factor of 5. Here our finest mesh was used.

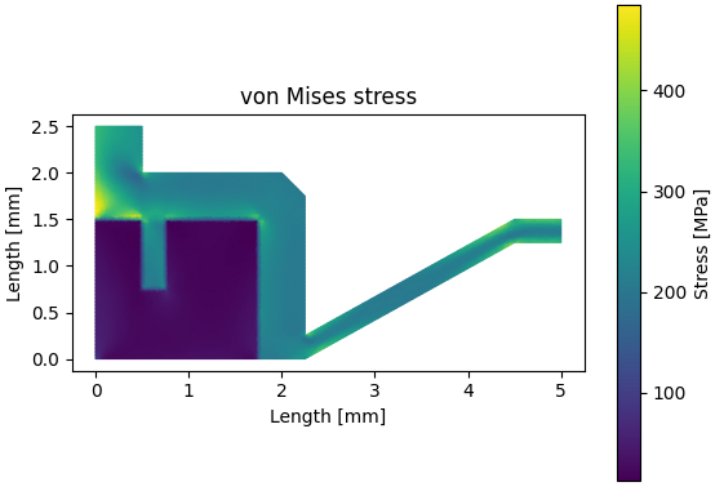


Figure 10: The von Mises stress for over the gripper for our fine mesh.

4 Discussion

4.1 Heat effects

Although the absolute temperature difference may not be significant, the heat distribution is similar to our anticipated distribution prior to the simulation, e.g. the arm will be the coolest as its the furthest away from the source of flux, and so disconnected from the rest of the body, disallowing a lot of the heat transfer. We also expected the nylon to heat up slower as it has a fraction of coppers thermal conductivity, slowing down heat transfer, and more than 3 times higher specific heat capacity, requiring more heat for it to heat up.

When observing the initial few seconds of the flux being introduced, the latter statements become even more clear. As one can see from figures 7b-7f, the copper elements of our body reacts to the thermal load substantially faster than the nylon.

Also take notice of the fact that our transient solution achieved 90% of the stationary temperature in 77.7 seconds (see figure 5), whilst it took 243.6 seconds to reach a reasonable approximation of the stationary temperature (99.9% of its max temperature), see figure 6. This shows that it takes roughly double the amount of time for our transient solution to go from 90% to 99.9% than for it to go from 0% to 90%. This indicates an inverse exponential relationship between time and temperature in our transient solution.

4.2 Mechanical effects

We found that we on average had a considerably higher stress in our copper elements than in the nylon. This was predicted as Young's modulus for copper is three orders of magnitude larger than nylon's, which can be interpreted as copper being more stiff than nylon, and thus we'll find more stress in copper when a thermo-mechanical load is applied.

The peaks of stress are most noticeable where the copper meets the nylon on the left part of the gripper. This reason for this might be because both the copper and the nylon wishes to expand there, but as we assumed they were clamped to the y-axis, so the stress becomes a lot higher there.

Another theory for the higher stress points could be that there occurs a torque around the corners facing the arms as the copper moves with the arm. This would explain why one of the corners doesn't have any stress, it would also explain the slightly higher strain in the leftmost corner, as the copper part above it has less room for movement since it's mostly clamped, whereas for the right most corner, the copper above it can move slightly with the arm as well.

We also note that the peak von Mises stress is roughly 500 MPa, as seen in figure 10. The yield strength of copper is roughly 70 MPa[2], meaning that the von Mises stress is much higher than the yield strength. This might mean that the material won't deform elastically but rather plastically. That would result in both that the copper becomes permanently deformed, as well as that our model - which assumes elastic deformation via Hooke's law - is partially incorrect, and that a non-linear model would be needed instead.

[2] https://en.wikipedia.org/wiki/Yield_(engineering)

5 Code

5.1 create_mesh.py

```
import calfem.geometry as cfg
import calfem.mesh as cfm
import calfem.vis_mpl as cfv
from matplotlib import pyplot as plt

# Values for surface- and boundary-marks
MARK_NYLON = 11
MARK_COPPER = 12
MARK_CONVECTION = 1
MARK_FLUX = 2
MARK_CLAMP = 3
MARK_XCLAMP = 4
MARK_YCLAMP = 5

L = 5 * 10**-3

def define_geometry():
    a = 0.1*L
    b = 0.1*L
    c = 0.3*L
    d = 0.05*L
    h = 0.15*L
    t = 0.05*L

    g = cfg.Geometry()

    NYLON_NBR_POINTS = 8
    COPPER_NBR_POINTS = 14

    # Declare points
    g.point([0, 0])
    g.point([0, 0.5*L - a - b])
    g.point([0, 0.5*L - b])
    g.point([0, 0.5*L])
    g.point([a, 0.5*L])
    g.point([a, 0.5*L - b])
    g.point([a + c, 0.5*L - b])
    g.point([a + c + d, 0.5*L - b - d])
    g.point([a + c + d, d])
    g.point([L - 2*d, c])
    g.point([L, c])
    g.point([L, c - d])
    g.point([L - 2*d, c - d])
    g.point([a + c + d, 0])
    g.point([c + d, 0])
    g.point([a, 0.5*L - a - b])
    g.point([a, 0.5*L - a - b - h])
    g.point([a + t, 0.5*L - a - b - h])
    g.point([a + t, 0.5*L - a - b])
    g.point([c + d, 0.5*L - a - b])

```

```
    # Declare edges (with marks where necessary)
    g.spline([0, 1], marker=MARK_CLAMP)
    g.spline([1, 2], marker=MARK_CLAMP)
    g.spline([2, 3], marker=MARK_FLUX)
    g.spline([3, 4], marker=MARK_YCLAMP)
    g.spline([4, 5], marker=MARK_CONVECTION)
    g.spline([5, 6], marker=MARK_CONVECTION)
    g.spline([6, 7], marker=MARK_CONVECTION)
    g.spline([7, 8], marker=MARK_CONVECTION)
    g.spline([8, 9], marker=MARK_CONVECTION)
    g.spline([9, 10], marker=MARK_CONVECTION)
    g.spline([10, 11], marker=MARK_XCLAMP)
    g.spline([11, 12], marker=MARK_CONVECTION)
    g.spline([12, 13], marker=MARK_CONVECTION)
    g.spline([13, 14])
    g.spline([14, 0])
    g.spline([1, COPPER_NBR_POINTS + 1])
    for i in range(1, NYLON_NBR_POINTS - 3):
        g.spline([COPPER_NBR_POINTS + i, COPPER_NBR_POINTS + i + 1])
    g.spline([19, 14])

    # Declare the two types of surfaces
    g.surface([0] + list(range(15, 21)) + [14], marker=MARK_NYLON)
    ↪  # Nylon
    g.surface(list(range(1, 14)) + list(reversed(range(15, 21))), marker=MARK_COPPER)
    ↪  # Copper

    return g


def create_mesh(draw=True, dofs_per_node=1, element_size_factor = 0.03):
    g = define_geometry()
    mesh = cfm.GmshMesh(g)
    mesh.elType = 2
    mesh.dofsPerNode = dofs_per_node
    mesh.elSizeFactor = element_size_factor
    mesh.return_boundary_elements = True

    coords, edof, dofs, bdofs, elementmarkers, boundary_elements = mesh.create()

    # Draw the mesh.

    if draw:
        cfv.draw_geometry(g, draw_points=False, label_points=False, label_curves=False,
        ↪  draw_axis=True)
        cfv.drawMesh(
            coords=coords*10**3,
            edof=edof,
            dofs_per_node=mesh.dofsPerNode,
            el_type=mesh.elType,
            filled=False,
            title=f"Mesh for gripper, element size factor: {element_size_factor}"
                )
        plt.xlabel('Length [mm]')
        plt.ylabel('Length [mm]')
```

```
        cfv.showAndWait()
    return coords, edof, dofs, bdofs, elementmarkers, boundary_elements

```

5.2 stationary.py

```
import calfem.vis_mpl as cfv
import calfem.core as cfc
import numpy as np
from matplotlib import pyplot as plt
from math import dist
from create_mesh import create_mesh, MARK_NYLON, MARK_COPPER, MARK_CONVECTION, MARK_FLUX

def stationary_solver(draw_mesh=False, draw=False, element_size_factor=0.03):
    coords, edof, dofs, bdofs, elementmarkers, boundary_elements =
    ↪  create_mesh(draw=draw_mesh, element_size_factor=element_size_factor)

    # Constants
    NELEM, NDOF = len(edof), len(dofs)
    k_copper = 385
    k_nylon = 0.26
    alpha_c = 40
    h = 10**5
    T_inf = 18


    K = np.zeros((NDOF, NDOF))
    fb = np.zeros((NDOF, 1))

    # Assemble the global stiffness matrix
    ex, ey = cfc.coord_extract(edof, coords, dofs)
    for i in np.arange(0, NELEM):
        if elementmarkers[i] == MARK_NYLON:     # Nylon
            Ke = cfc.flw2te(ex[i], ey[i], [1], k_nylon * np.eye(2))
        if elementmarkers[i] == MARK_COPPER:    # Copper
            Ke = cfc.flw2te(ex[i], ey[i], [1], k_copper*np.eye(2))
        cfc.assem(edof[i,:], K, Ke)

    # Access the nodes for the respective boundaries
    nodes_conv = [node['node-number-list'] for node in
    ↪  boundary_elements[MARK_CONVECTION]]
    nodes_flux = [node['node-number-list'] for node in boundary_elements[MARK_FLUX]]

    # Assemble Kc and fc (convection)
    for node in nodes_conv:
        Le = dist(coords[node[0] - 1], coords[node[1] - 1])
        Kce = alpha_c*Le/6 * np.array([[2, 1], [1, 2]])
        fb[node[0]-1] += alpha_c*Le*T_inf/2
        fb[node[1]-1] += alpha_c*Le*T_inf/2
        cfc.assem(np.array([node[0], node[1]]), K, Kce)

    # Add flux to fb
    for node in nodes_flux:
        Le = dist(coords[node[0] - 1], coords[node[1] - 1])
```

```
        fb[node[0]-1] += h*Le/2
        fb[node[1]-1] += h*Le/2

    # Solve the FE-system
    bcPresc = np.array([], 'i')
    a, _ = cfc.solveq(K, fb, bcPresc)

    if draw:
        cfv.draw_nodal_values_shaded(a, coords*10**3, edof, title="Stationary teperature
        ↪ distribution")
        cbar = cfv.colorbar()
        cbar.set_label('Temperature [°C]', rotation=90)
        plt.xlabel('Length [mm]')
        plt.ylabel('Length [mm]')
        plt.set_cmap("inferno")
        cfv.show_and_wait()

    return a, K, fb, ex, ey, coords, edof, dofs, bdofs, elementmarkers

if __name__ == '__main__':
    stationary_solver(draw=True)

```

5.3 transient.py

```
import calfem.vis_mpl as cfv
import calfem.core as cfc
import numpy as np
from matplotlib import pyplot as plt
from plantml import plantml
from create_mesh import MARK_NYLON
from stationary import stationary_solver

def transient_solver():
    # First solve the stationary problem, and access the topology variables as well as K
    ↪ and f
    a_stationary, K, f, ex, ey, coords, edof, dofs, _, elementmarkers =
    ↪ stationary_solver(element_size_factor=0.03)
    NELEM, NDOF = len(edof), len(dofs)

    # Initial temperature
    a = 18 * np.ones((NDOF, 1))
    # Time step
    dt = 0.1
    tf = 100
    # Material constants
    rho_nylon = 1100
    rho_copper = 8930
    c_nylon = 1500
    c_copper = 386
    stat_temp = np.max(a_stationary)

    # Assemble the C-matrix
    C = np.zeros((NDOF, NDOF))
```

```
    for i in np.arange(0, NELEM):
        if elementmarkers[i] == MARK_NYLON: # Nylon
            Ce = plantml(ex[i], ey[i], rho_nylon * c_nylon)
        else:                               # Copper
            Ce = plantml(ex[i], ey[i], rho_copper * c_copper)
        cfc.assem(edof[i], C, Ce)

    j = 1
    for t in np.arange(0, tf, step=dt):
        # Implicit Euler time step
        a = np.linalg.solve(C + dt*K, C @ a + dt*f)
        if t > j * 0.44 and j <= 5: # Draw 5 pictures under the first 3% of the time
        ↪  under 90%
                                    # 0.44 was calculated to satisfy this
            plt.figure()
            cfv.draw_nodal_values_shaded(a, coords * 10**3, edof)
            cbar = cfv.colorbar()
            cbar.set_label('Temperature [°C]', rotation=90)
            plt.clim(18, 28) # Force colorbar limits to be the same
            plt.title(f"Time = {float(t):.1f} s")
            plt.xlabel('Length [mm]')
            plt.ylabel('Length [mm]')
            plt.set_cmap("inferno")
            j += 1
        if np.max(a) > 0.9 * stat_temp: # Or 0.999
            break

    plt.figure()
    cfv.draw_nodal_values_shaded(a, coords * 10**3, edof)
    cbar = cfv.colorbar()
    cbar.set_label('Temperature [°C]', rotation=90)
    plt.title(f"Time = {float(t):.1f} s, 90% of stationary max temperature")
    plt.xlabel('Length [mm]')
    plt.ylabel('Length [mm]')
    plt.set_cmap("inferno")
    cfv.show_and_wait()

if __name__ == '__main__':
    transient_solver()

```

5.4 vonMises.py

```
import calfem.vis_mpl as cfv
import calfem.core as cfc
import calfem.utils as cfu
import numpy as np
from matplotlib import pyplot as plt
from create_mesh import MARK_NYLON, MARK_FLUX, MARK_XCLAMP, MARK_YCLAMP, MARK_CLAMP, L
from stationary import stationary_solver

def vonMises_solver():
    a_stat, _, _, ex, ey, coords, edof, dofs, bdofs, elementmarkers =
    ↪  stationary_solver(element_size_factor=0.007)
```

```
    NELEM, NDOF = len(edof), len(dofs)

    T_inf = 18
    E_copper = 128 * 10**9
    E_nylon = 3 * 10**9
    v_copper = 0.36
    v_nylon = 0.39
    alpha_copper = 17.6 * 10**-6
    alpha_nylon = 80 * 10**-6

    # Create new dofs, edofs and bdofs as we now have two degrees of freedom for displacement
    NDOF_S = 2*NDOF

    # New dofs through the map n -> [2n-1, 2n]
    dofs_S = np.array([[2*n-1, 2*n] for n in range(1, NDOF+1)])

    # New edof
    edof_S = np.zeros((NELEM, 6), dtype=int)
    for i, row in enumerate(edof):
        edof_S[i] = [2*row[0]-1, 2*row[0], 2*row[1]-1, 2*row[1], 2*row[2]-1, 2*row[2]]

    # New bdofs
    bdofs_S = {}
    for mark, nodes in bdofs.items():
        bdofs_S[mark] = [dof for node in nodes for dof in [2*node-1, 2*node]]

    # Create our D-matrix as done in the "Termoelasticetet - plan töjning - uppdatering"-post on Canvas
    def Dmatrix(E, v):
        return E/((1+v)*(1-2*v)) * np.array([[1-v, v, 0],
                                             [v, 1-v, 0],
                                             [0, 0, 0.5*(1-2*v)]])

    D_copper = Dmatrix(E_copper, v_copper)
    D_nylon = Dmatrix(E_nylon, v_nylon)

    # Assemble the global stiffnes matrix for the stress problem
    Ks = np.zeros((NDOF_S, NDOF_S))
    fs0 = np.zeros((NDOF_S, 1))

    def create_element_values(ex, ey, alpha, dT, D):
        Ke = cfc.plante(ex, ey, [2, 1], D)
        epsilon_dT = alpha * dT * np.array([[1], [1], [0]])
        fe = cfc.plantf(ex, ey, [2, 1], (D @ epsilon_dT).T)
        return Ke, fe

    for i in range(NELEM):
        deltaT = np.mean([a_stat[edof[i][j] - 1] for j in range(3)]) - T_inf # Average increase in temperature for an element
        if elementmarkers[i] == MARK_NYLON: # Nylon
            Kse, fs0e = create_element_values(ex[i], ey[i], alpha_nylon, deltaT, D_nylon)
        else:                               # Copper
```

```
            Kse, fs0e = create_element_values(ex[i], ey[i], alpha_copper, deltaT,
            ↪  D_copper)

        # Add the forces to the force vector f0
        for j in range(6):
            fs0[edof_S[i][j] - 1] += fs0e[j]

        cfc.assem(edof_S[i], Ks, Kse)

    # Apply boundary conditions
    bc_S = np.array([], 'i')
    bc_val_S = np.array([], 'f')
    bc_S, bc_val_S = cfu.apply_bc(bdofs_S, bc_S, bc_val_S, MARK_CLAMP)
    bc_S, bc_val_S = cfu.apply_bc(bdofs_S, bc_S, bc_val_S, MARK_FLUX)
    bc_S, bc_val_S = cfu.apply_bc(bdofs_S, bc_S, bc_val_S, MARK_XCLAMP, dimension=1) #
    ↪  Only allowed to move in y-direction
    bc_S, bc_val_S = cfu.apply_bc(bdofs_S, bc_S, bc_val_S, MARK_YCLAMP, dimension=2) #
    ↪  Only allowed to move in x-direction

    # Solve the FE-system
    a_S, _ = cfc.solveq(Ks, fs0, bc_S)

    # Create the von Mises stress
    ed = cfc.extract_eldisp(edof_S, a_S)
    vonMises = np.zeros(NELEM)
    for i in range(NELEM):
        if elementmarkers[i] == MARK_NYLON: # Nylon
            D_nylon = cfc.hooke(2, E_nylon, v_nylon)
            es, _ = cfc.plants(ex[i], ey[i], [2, 1], D_nylon, ed[i])
            E, v, alpha = E_nylon, v_nylon, alpha_nylon
        else:                               # Copper
            D_copper = cfc.hooke(2, E_copper, v_copper)
            es, _ = cfc.plants(ex[i], ey[i], [2, 1], D_copper, ed[i])
            E, v, alpha = E_copper, v_copper, alpha_copper
        sigx, sigy, sigz, tauxy = es[0]
        deltaT = np.mean([a_stat[edof[i][j] - 1] for j in range(3)]) - T_inf #
        ↪  Temperature increase

        # Factor in the stress from temperature increase
        k_T = alpha*E*deltaT/(1-2*v)
        sigx -= k_T
        sigy -= k_T
        sigz -= k_T
        stress = np.sqrt(sigx**2 + sigy**2 + sigz**2 - sigx*sigy - sigx*sigz - sigy*sigz
        ↪  + 3*tauxy**2)
        vonMises[i] = stress

    # Calculate the stress for each node by averaging the stress for all elements the
    ↪  node is included in
    node_stresses = np.zeros((NDOF, 1))
    for i in range(NDOF):
        x = dofs_S[i][0]
        idxs = np.where(x == edof_S)[0]
        node_stresses[i] = np.mean([vonMises[idx] for idx in idxs])

```

```
110     plt.figure()
111     cfv.draw_nodal_values_shaded(node_stresses * 10**-6, coords * 10**3, edof,
        ↪  title="von Mises stress")
112     cbar = cfv.colorbar()
113     cbar.set_label('Stress [MPa]', rotation=90)
114     plt.xlabel('Length [mm]')
115     plt.ylabel('Length [mm]')
116 
117     # Draw the displacement of the whole gripper
118     plt.figure()
119     magnification = 5
120     flip_y = np.array([([1, -1]*int(a_S.size/2))]).T
121     flip_x = np.array([([-1, 1]*int(a_S.size/2))]).T
122     cfv.draw_element_values(vonMises * 10**-6, coords, edof_S, 2, 2, a_S,
123                             draw_elements=False, draw_undisplaced_mesh=True,
124                             title="Effective Stress, magnification=%ix" % magnification,
                                ↪  magnfac=magnification)
125     cfv.draw_element_values(vonMises * 10**-6, [0, L]+[1, -1]*(coords), edof_S, 2, 2,
        ↪  np.multiply(flip_y, a_S),
126                             draw_elements=False, draw_undisplaced_mesh=True,
                                ↪  magnfac=magnification)
127     cfv.draw_element_values(vonMises * 10**-6, [2*L, L]+[-1, -1]*(coords), edof_S, 2, 2,
        ↪  np.multiply(flip_y*flip_x, a_S),
128                             draw_elements=False, draw_undisplaced_mesh=True,
                                ↪  magnfac=magnification)
129     cfv.draw_element_values(vonMises * 10**-6, [2*L, 0]+[-1, 1]*(coords), edof_S, 2, 2,
        ↪  np.multiply(flip_x, a_S),
130                             draw_elements=False, draw_undisplaced_mesh=True,
                                ↪  magnfac=magnification)
131     cbar = cfv.colorbar()
132     cbar.set_label('Stress [MPa]', rotation=90)
133     plt.xlabel('Length [m]')
134     plt.ylabel('Length [m]')
135 
136     plt.figure()
137     cfv.draw_displacements(a_S, coords, edof, 1, 2, draw_undisplaced_mesh=True,
138                            title="Displacement of mesh", magnfac=magnification)
139     plt.xlabel('Length [m]')
140     plt.ylabel('Length [m]')
141 
142     cfv.show_and_wait()
143 
144 if __name__ == '__main__':
145     vonMises_solver()
146 
```