FEM - Projekt

Mosa Hosseini, Simon Löwgren

Maj 2021

1 Introduction

The project presents the task of analyzing a simplified model of a lens used by the NASA rover Perserverance. To ensure its functioning under the rapidly shifting temperatures of Mars, a finite element analysis is used to calculate temperature distribution, stress field and the associated deformation of the lens. The CALFEM package for MATLAB [1] is used for these calculations.

A simplified model of the lens is used, consisting of a titanium alloy (marked with Ti in figure below) and three glass lenses marked with gl in Figure 1), extending a cross section $1cm$ (thickness) into the plane. A cross section of this is shown in figure 1, which includes the measurements of the lens in centimeters. The temperature on the inside of the lens, denoted T_c, is controlled by systems inside the rover to be 20 °C at all times, while the lens is exposed to the outside on the other side (left side in figure 1). This temperature shifts between 40 °C during the daytime, and -96 °C during the nighttime, with heat transfer creating two different stationary conditions for these temperature pairs.

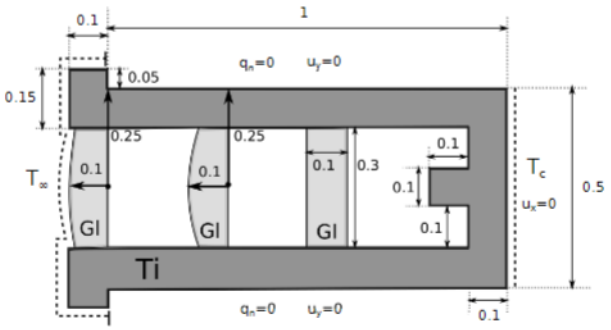


Figure 1: *The rover lens. Source: Project instruction [2]*

In addition to the geometry of the lens, some material parameters for glass and titanium is necessary for this project. Material parameters can be viewed in the figure below.

	Titanitum alloy	Glass
Young's modulus, E [GPa]	110	67
Poisson's ratio, ν [-]	0.34	0.2
Expansion coefficient, α $[1/K]$	$9.4 \cdot 10^{-6}$	$7 \cdot 10^{-6}$
Density, ρ [kg/m^3]	4620	3860
Specific heat, c_p [J/(kg K)]	523	670
Thermal conductivity, k [W/(m K)]	17	0.8

Figure 2: *Material parameters used in the simulations*

The aim of the project was to first solve and plot the stationary temperature distributions for the previously described day- and night-time conditions, and secondly to solve and plot the resulting transient heat flow during the day, given that the initial condition is stationary night temperature distributions and vice versa. Secondly, the Von Mises stress field and corresponding displacement field is calculated and plotted for the same conditions. Lastly, the square sum of the resulting displacements of the outer lens (leftmost in figure 1) is calculated.

2 Theory

The project consists mainly of two part, a heat flow problem and a stress-strain problem. In order to derive FE formulation for these problems, we first need to derive the strong and weak formulations for each problem separately.

2.1 Heat Flow

The differential equation for heat flow can be derived from equilibrium condition, which states the heat that is generated inside a body (Q) is equal to heat that flows out of the body ($\mathbf{q}$). In addition to the equilibrium equation we also need a constitutive law for heat flow, and that is given by Fourier's law of thermal conduction, $q = -k \cdot \frac{dT}{dx}$. In two dimensions k is replaced by a matrix $\mathbf{D}$ which is equal to,

$$\mathbf{D} = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}$$

Moreover, the operator d/dx replaces the gradient operator.

$$q = -\mathbf{D} \cdot \nabla T \tag{1}$$

this gives the strong formulation of the heat equation.

$$div(q) - Q = 0 \tag{2}$$

However, when examining the transient heat flow, there is a time dependent term on right hand side of the equation. Thus, for a material where the specific heat c and density ρ are constant, the equation will take the following form:

$$div(q) - Q + c\rho\dot{T} = 0. \tag{3}$$

In order to formulate the FE-formulation of this problem, the weak formulation has to be derived. To this end, multiply equation (3) with an arbitrary weight function and integrate over the whole domain. After that, the Green-Gauss theorem can be used (see Ottosen & Petersson p.74) to derive the weak formulation,

$$\int_L v\mathbf{q}^T \cdot \mathbf{n}\, dL - \int_A (\nabla v)^T \mathbf{q}\, dA - \int_A vQdA + \int_A c\rho v\dot{T}\, dA = 0 \tag{4}$$

In this case, the system doesn't produce any heat, and thus $Q = 0$ in equation (4). From the weak formulation the FE-formulation can be derived.

2.2 Stress

The differential equation of equilibrium which governs linear elasticity is given by:

$$\tilde{\nabla}^T \cdot \boldsymbol{\sigma} + \mathbf{b} = 0 \tag{5}$$

where $\boldsymbol{\sigma}$ is the stress vector, and $\boldsymbol{b}$ is the body force vector. $\tilde{\nabla}^T$ is given by:

$$\tilde{\nabla}^T = \begin{bmatrix} \frac{\delta}{\delta x} & 0 & \frac{\delta}{\delta y} \\ 0 & \frac{\delta}{\delta y} & \frac{\delta}{\delta x} \end{bmatrix}$$

Deriving the weak form from this equation as done in Ottosen & Peterson leaves the following equation:

$$\int_V (\tilde{\nabla}\mathbf{v})^T \boldsymbol{\sigma}\, dV = \int_S \mathbf{v}^T \mathbf{t}\, dS + \int_V \mathbf{v}^T b\, dV \tag{6}$$

To solve this type of integrals, a constitutive law is needed. In one dimension, the constitutive law is given by Hooke's law; $\sigma = E\epsilon$, where E is Young's module, in two dimensions however, Young's module is replaced by the matrix $\mathbf{D}$, which is defined as:

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

In the case where there is no initial strain, e.g. thermal strain, the constitutive law can be written as:

$$\boldsymbol{\sigma} = \mathbf{D}\epsilon$$

But in presence of an initial strain, like the thermal strain present in this problem, the equation can be written as:

$$\boldsymbol{\sigma} = \mathbf{D}(\boldsymbol{\epsilon} - \boldsymbol{\epsilon}^{\Delta T}) \tag{7}$$

Note that the the relation between strain and displacements are:

$$\boldsymbol{\epsilon} = \tilde{\nabla}\mathbf{u} \tag{8}$$

The stress vector $\boldsymbol{\epsilon}$ consists solely of the strains in the x-axis, y-axis and the shear strain, $\boldsymbol{\epsilon} = \begin{bmatrix}\epsilon_{xx} & \epsilon_{yy} & \gamma_{xy}\end{bmatrix}^T$. The thermal strains caused by the thermal expansion are formulated as [3] .

$$\boldsymbol{\epsilon}^{\Delta T} = (1+\nu)\alpha\Delta T \begin{bmatrix}1 & 1 & 0\end{bmatrix}^T \tag{9}$$

Where ν is Poisson's ratio and α is so called expansion coefficient, the value of this parameters can be found in Figure 2.

This gives the complete formulation for the in-plane strains as:

$$\begin{bmatrix}\sigma_{xx}\\ \sigma_{yy}\\ \sigma_{xy}\end{bmatrix} = \frac{E}{(1+v)(1-2v)} \begin{bmatrix}1-v & v & 0\\ v & 1-v & 0\\ 0 & 0 & \frac{1}{2}(1-2v)\end{bmatrix} \begin{bmatrix}\epsilon_{xx}\\ \epsilon_{yy}\\ \gamma_{xy}\end{bmatrix} - \frac{\alpha E\Delta T}{1-2v} \begin{bmatrix}1\\1\\0\end{bmatrix}$$

together with the out-of-plane stress σ_{zz}:

$$\sigma_{zz} = v(\sigma_{xx} + \sigma_{yy}) - \alpha E\Delta T \tag{10}$$

The von Mises stress equation is given by the project description:

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

Note that because we have a plane problem the sheer stresses involving z direction is equals to 0 eg. $\tau_{yz} = \tau_{xz} = 0$ this give us :

$$\sigma_{eff} = \sqrt{\sigma_{xx}^2 + \sigma_{yy}^2 + \sigma_{zz}^2 - \sigma_{xx}\sigma_{yy} - \sigma_{xx}\sigma_{zz} - \sigma_{yy}\sigma_{zz} + 3\sigma_{xy}^2}$$

3 FE-Formulation

3.1 Heatflow

As the approximations use triangular elements with a linear approximation, the following is used for the temperature:

$$T = \alpha_1 + \alpha_2 \cdot x + \alpha_3 \cdot y.$$

It's trivial to check that approximation above fulfills the convergence criterion (see Ottosen & Petersson p.92). Using the C matrix method, the approximation

is written as: $T = \mathbf{N} \cdot \mathbf{a}$, where a_i is the nodal temperature at node number i. As for the time derivative of temperature, we move the derivative along the nodal temperatures according to $\dot{T} = \mathbf{N} \cdot \dot{\mathbf{a}}$.

When choosing a weight function, the Galerkin method is used, which means choosing the weight functions to be the same as the trial function $v = \mathbf{N} \cdot \mathbf{c} = v^T = \mathbf{c}^T \cdot \mathbf{N}^T$. The expression is substituted for v and T in the weak formulation of heat flow, and $\boldsymbol{c}^T$ is factored out. Note that because $\boldsymbol{c}$ can be chosen arbitrarily, the expression that is multiplied by $\boldsymbol{c}$ has to be 0 (see Ottosen & Petersson p.208). The approximation $T = \mathbf{N} \cdot \mathbf{a}$ inserted in (1) gives $\mathbf{q} = \nabla \mathbf{N} \cdot \mathbf{a} = \mathbf{B}\mathbf{a}$.

With these substitutions, the weak formulation, equation (4), can be written as:

$$\int_A \mathbf{B}^T \mathbf{D} \mathbf{B} t \, dA \cdot \mathbf{a} + \int_A \mathbf{N}^T c\rho \mathbf{N} t \, dA \cdot \dot{\mathbf{a}} = -\int_{\mathscr{L}} \mathbf{N}^T q_n t \, dL \tag{11}$$

The equation above can be written as:

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

which is a system of time dependent differential equations. To discretize the system in time, we can approximate $\dot{a}$ in the following way:

$$\dot{\boldsymbol{a}} \approx \frac{\boldsymbol{a_{n+1}} - \boldsymbol{a_n}}{\boldsymbol{\Delta t}} \tag{13}$$

where a_n is the current value for a and a_{n+1} is the value for a after one time step. and $\Delta t = t_{n+1} - t_n$.
Using the Implicit Euler method and substituting (13) in $\dot{\boldsymbol{a}}$ we get:

$$\boldsymbol{C} \cdot \frac{\boldsymbol{a_{n+1}} - \boldsymbol{a_n}}{\Delta t} + \boldsymbol{K} \cdot \boldsymbol{a_{n+1}} = \boldsymbol{f_{n+1}}$$

.
Where $\boldsymbol{C} = c\rho \int_A N^T N dA$ and a_n the nodal temperatures at time n, K the global stiffness matrix, and f the boundary force vector.
After rewriting the equation we arrive at.

$$\boldsymbol{a_{n+1}} = \frac{\Delta t \cdot \boldsymbol{f_{n+1}} + \boldsymbol{C}\boldsymbol{a_n}}{\boldsymbol{C} + \boldsymbol{K}\Delta t} \tag{14}$$

In our case the initial conditions were given by stationary day/night conditions.

3.2 Stress

For the stress, there are two degrees of freedom per node which means that a nodal approximation in both the x- and y-direction is required.

$$\mathbf{U} = \begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} \beta_1 + \beta_2 x + \beta_3 y \\ \gamma_1 + \gamma_2 x + \lambda_3 y \end{bmatrix}$$

Similarly to the derivation of heat flow, we can write it as $T = \boldsymbol{N}\boldsymbol{a}$, where $\boldsymbol{a}$ is the nodal displacement such that $a_1 = u_x$ and $a_2 = u_y$ as usual and $\boldsymbol{N}$ is the form function. Note that $\mathbf{N}$ is a $2n \times 2$ matrix where n is the number of nodes per element.

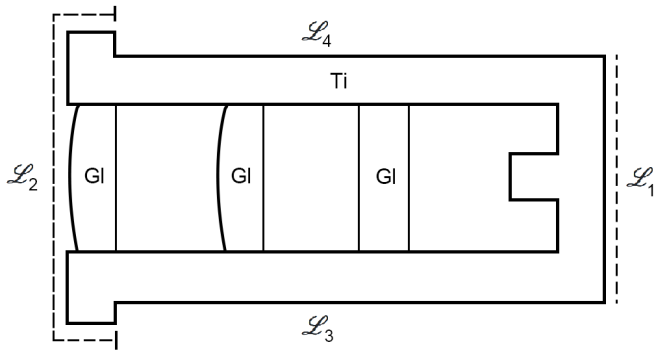
In accordance with the heat flow formulation, weight functions are picked according to the Galerkin method. Since $\boldsymbol{v} = \boldsymbol{N}\boldsymbol{c}$ and $\boldsymbol{v}^T = \boldsymbol{c}^T\boldsymbol{N}^T$, the result becomes $(\tilde{\nabla}\boldsymbol{v}) = (\boldsymbol{B}\boldsymbol{c})^T = \boldsymbol{c}^T\boldsymbol{B}^T$, where $\boldsymbol{B} = \tilde{\nabla}N$ (Ottosen & Petersson p.296). An approximation is needed for equation 8, which is given by $\boldsymbol{\epsilon} = \boldsymbol{B}\boldsymbol{a}$.
Using equation (7) Inserting the above mentioned approximation in (6) gives us:

$$\int_A \mathbf{B}^T\mathbf{D}\mathbf{B}t\,dA \cdot \mathbf{a} = \int_A \mathbf{B}^T\mathbf{D}\epsilon^{\Delta T}t\,dA + \int_{\mathscr{L}} \mathbf{N}^T\mathbf{t}t\,d\mathscr{L} \tag{15}$$

Where $\boldsymbol{t}$ is the traction vector, and t is the thickness.
For the implementation of these calculations, the CALFEM package for Matlab and pdetool was used. This will be expanded on in the section concerning implementation.

4 Implementation and solution

First of all, we want to define boundary conditions.



(a)

Figure 3: *Implemented model with marked boundaries and materials, dashed lines mark boundaries where convection occurs*

4.1 Heat flow

The boundary condition for heat flow is:

$$\begin{aligned}\mathscr{L}_4, \mathscr{L}_3 &: q_n = 0 \\ \mathscr{L}_1 &: q_n = a_c(T - T_c) \\ \mathscr{L}_2 &: q_n = a_c(T - T_\infty)\end{aligned}$$

where $a_c = 100W/m^2K$, $T_c = 20°C$ and T_∞ is given by the environmental temperature, which is $-96°C$ during the night and $40°C$ during the day. Given the boundary conditions, we can rewrite (16) but note that in stationary temperature distribution the term with $\dot{\boldsymbol{a}} = \mathbf{0}$:

$$\int_A \mathbf{B}^T\mathbf{D}\mathbf{B}t\,dA\cdot\mathbf{a} = -\int_{\mathscr{L}_4,\mathscr{L}_3} \mathbf{N}^T q_n t\,dL - \int_{\mathscr{L}_1} \mathbf{N}^T\cdot a_c(\mathbf{N}\mathbf{a} - T_c)t\,d\mathscr{L} - \int_{\mathscr{L}_2} \mathbf{N}^T\cdot a_c(\mathbf{N}\mathbf{a} - T_\infty)t\,d\mathscr{L} \tag{16}$$

Writing it in a compact form we get:

$$\mathbf{K} = \int_A \mathbf{B}^T\mathbf{D}\mathbf{B}t\,dA + a_c\int_{\mathscr{L}_1,\mathscr{L}_2} \mathbf{N}^T\cdot\mathbf{N}t\,d\mathscr{L} \tag{17}$$

$$\mathbf{f} = a_c\int_{\mathscr{L}_1} \mathbf{N}^T\cdot T_c t\,d\mathscr{L} + a_c\int_{\mathscr{L}_2} \mathbf{N}^T\cdot T_\infty t\,d\mathscr{L} - \int_{\mathscr{L}_3,\mathscr{L}_4} \mathbf{N}^T\cdot T q_n t\,d\mathscr{L} \tag{18}$$

4.1.1 Modifying stiffness matrix and load vector

The convection affects how the material acts, and as a result K and f have to be changed. Added to K is a term K_c given by $K_c = \alpha_c t\int_{\mathscr{L}} N^T N d\mathscr{L}$. [3] The integration may be simplified along the edges by realizing that the shape function N_i of a node i linearly decreases according to $1-n$ along the edge while traveling to another node j, and the value of N_j will then be n, also linearly increasing. We may multiply and integrate these values to get $\int_0^1 n - n^2 dv = \frac{1}{6}$, or $\int_0^1 (1-v)^2 dv = \frac{1}{3}$ if $i = j$.

For the three nodes of an element N_i, N_j and N_k, where N_i and N_j are on the edge, the shape functions are then:

$$N_1 = 1 - \frac{x}{L}, N_2 = \frac{x}{L}, N_3 = 0$$

Where L is the length of the edges. For every element, this gives a 3x3 matrix, which is integrated from 0 to L. This gives:

$$\mathbf{K}_c^e = \alpha t\int_0^L \sum_{i=1}^{3}\sum_{j=1}^{3} \mathbf{N}_i\mathbf{N}_j dx$$

The integrals have the results $\frac{L}{3}$, $\frac{L}{6}$ and 0, resulting in the previously mentioned matrix taking the form:

$$\alpha t \begin{pmatrix} \frac{L}{3} & \frac{L}{6} & 0 \\ \frac{L}{6} & \frac{L}{3} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

However, the order of nodes within an element means that the matrix can take one of three different forms, with the other two being:

$$\alpha t \begin{pmatrix} 0 & 0 & 0 \\ \frac{L}{3} & \frac{L}{6} & 0 \\ \frac{L}{6} & \frac{L}{3} & 0 \end{pmatrix}, \alpha t \begin{pmatrix} \frac{L}{3} & 0 & \frac{L}{6} \\ 0 & 0 & 0 \\ \frac{L}{6} & 0 & \frac{L}{3} \end{pmatrix}$$

Further, as the third element is taken to be zero during this integration, the problem can be simplified to one dimension. The matrix can be calculated using:

$$\frac{\alpha_c t}{L^2} \int_0^L \begin{bmatrix} (L-x)^2 & x(L-x) \\ x(L-x) & x^2 \end{bmatrix} dx$$

which means that every element has a $\mathbf{K}_c^e$ matrix of:

$$\frac{\alpha_c L t}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

The load vector is modified similarly by adding a term f_c, given by:

$$\int_{\mathscr{L}} N^T = \frac{1}{L} \int_0^L \begin{bmatrix} L-x \\ x \end{bmatrix} dx = \frac{L}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} d\mathscr{L}$$

using the same terms as used for K_c. This integral will result in zero for any node not on the boundary and affected by convection. The sum of the shape functions for an edge node is simply the sum of the distance to every other adjacent edge node, divided by two (due to the shape functions forming right-sided triangles with the base length equal to the distance to an adjacent node, and the height one).

4.1.2 Calculating stationary temperature distributions

As the geometry is simplified and approximated to be two-dimensional, the element stiffness vector K^e and element load vector f^e can be calculated using the CALFEM function $flw2te$ (two dimensional heat flow for a triangular element), which uses the previously calculated nodal coordinates, thickness (defined as 1 cm) and predefined thermal conductivity. As the creation of the geometry in pdetoolbox also supplies the different regions of the element, this was used to check whether an element was glass or titanium alloy.

Looping over all elements, and calculating K^e, f^e and the C-matrix (through the *plantml* function), a prebuilt function *assem* (or it's alternative which was

offered in the project manual) was used to add the element stiffness matrix to the global stiffness matrix in accordance with the elemental degree of freedom. Results were then extracted from the solution vector according to the global topology matrix using the function *extract*. The solution was drawn using the $fill$ function.

4.1.3 Calculating transient temperature evolution

The $C^e - matrix$ was calculated using *plantml* and assembled using the CALFEM function *assem*, and using equation 14, the temperature distributions at different points in time was extracted and plotted.

4.2 Stress

The boundary conditions governing displacement of the lens are:

$$\begin{aligned} \mathscr{L}_1 &: u_x = 0 \\ \mathscr{L}_3, \mathscr{L}_4 &: u_y = 0 \end{aligned}$$

Where u_x and u_y represent the displacements in the x- and y-axis respectively. As $\mathscr{L}_2$ has no fixed position, it is free to expand and contract as long as it does so without violating the boundary conditions of other edges.

Following the FE-formulation of linear elasticity given in equation 15, K matrix for an element is:

$$\mathbf{K^e} = \int_A \mathbf{B^{eT}} \mathbf{D} \mathbf{B^e} t \, dA$$

The element stiffness matrix $\boldsymbol{K_e}$ is calculated using the function *plante*, which is supplied the nodal coordinate vectors, a vector consisting of the *ptype* and element thickness **ep**, the previously calculated constitutive matrix D and a vector containing loads per unit volume.

The force vector $\boldsymbol{f}$ is equal to:

$$\mathbf{f} = \int_A \mathbf{B^T} \mathbf{D} \epsilon^{\Delta T} t \, dA + \int_{\mathscr{L}} \mathbf{N}^T \mathbf{t} t \, d\mathscr{L} \tag{19}$$

The $plantf$ function in CALFEM was used to calculate the first term of (19), the second term is simply zero. allowing the usage of *assem* to calculate the global K and f-matrices. After applying the boundary conditions, after which *solve* was used to calculate displacements. to calculate stress and strains the function *plants* is used. After which inserted in (7) and using the expression for $\boldsymbol{\epsilon}^{\Delta T}$ see equation (9) we get an expression for stresses, but we also have to calculate σ_{zz} using (10), then we can insert it into expression for vonmisses to calculate vonmises stressfield.

4.2.1 Calculating displacement magnitude square sum

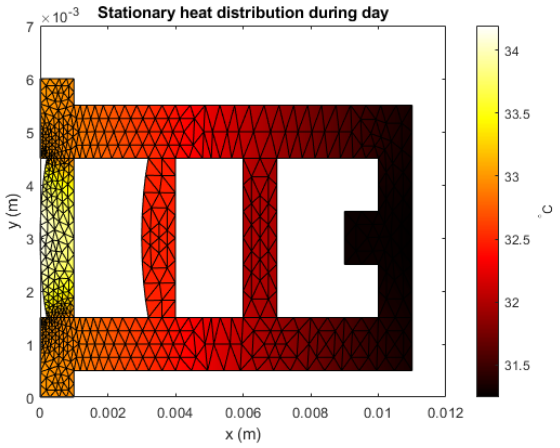
In order for the lenses to maintain their intended optical properties their curvature should remain as close as possible to their original design during temperature shifts. One way to quantify the displacement is to compute the square sum of the displacements in the lens. The square sum of the most left lens calculated using the equation supplied by the project description, namely:

$$\int_{\Omega lens} \mathbf{u}^T \mathbf{u} dV = \mathbf{a}^T \int_{\Omega lens} \mathbf{N}^T \mathbf{N} dV \mathbf{a} = \mathbf{a}^T \mathbf{T} \mathbf{a}$$

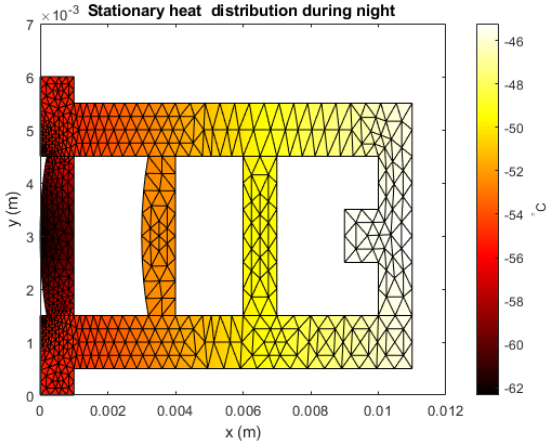
We modify *Plantml* so called *plantmlmod* to calculate the integral.

5 Results

The plot of stationary distribution can be seen in figures below.



(a) Daytime stationary temperature distribution

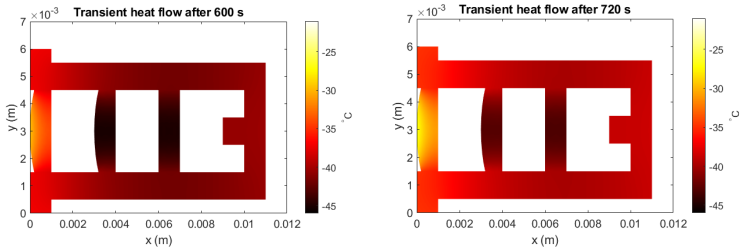


(b) Nighttime stationary temperature distribution

Figure 4: *Stationary temperature distributions*

Maximum temperature during daytime was 34.1953°C, and maximum temperature during nighttime was -45.2203°C.

The second part of the was to solve transient heat flow assignment, the figure below shows transient heat flow starting at night conditions.

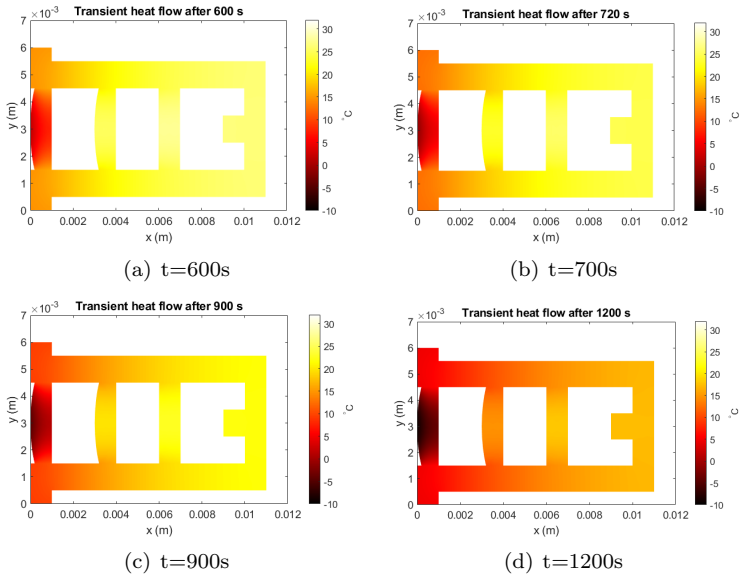


(a) t=600s (b) t=720s

(c) t=900s (d) t=1200

Figure 5: *Heat flow over time, starting from stationary nighttime conditions*

The temperatu is also of interest to see how the transient solution looks when we have stationary day time solution as initial condition and the temperature outside is night times temperature e.g $T_\infty = -96$ see figures below.

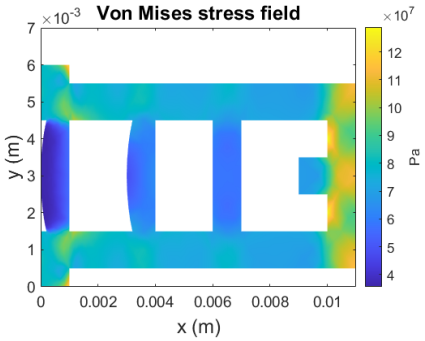


(a) t=600s (b) t=700s

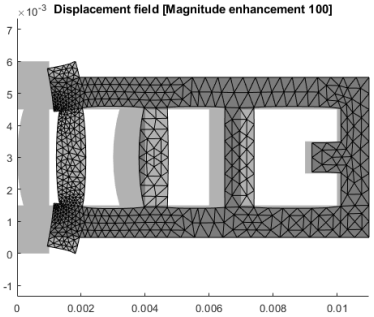
(c) t=900s (d) t=1200s

Figure 6: *Heat flow over time, starting from stationary day conditions.*

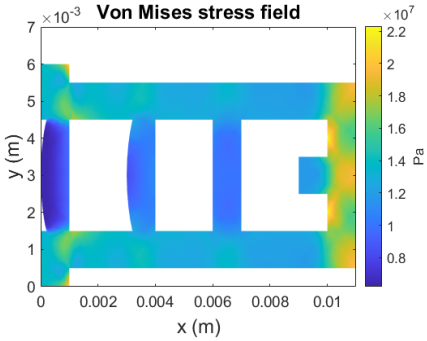
The plot of Von Mises Stressfield and displacement field can be seen in figures below. As we kan see in the figures the stress is highest at the end of the camera (in x=0.001 m) and about 6 times higher during the night compared to daytime.



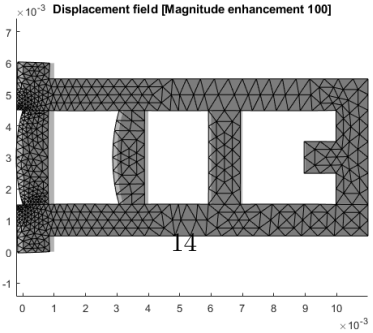
(a) Von Mises' stress graph,with night condition



(b) Displacement field given night condition , magnitude enhanced 100



(c) Von Mises' stress graph,with daytime condition



(d) Displacement field given day condition, magnitude enhanced 100

Figure 7: *The von Mises stress fields and corresponding displacements, for day- and nighttime conditions*

For the leftmost lens, the square sum of displacements is $1.1272 \cdot 10^{-17}$ during the day and $3.7541 \cdot 10^{-16}$ for the night.

6 Discussion

The temperature distribution, both stationary and transient, show probable results for this combination of materials. As the glass has a higher specific heat than the titanium alloy, resulting in a slower temperature change and the retaining of a colder temperature from the previous nighttime conditions. Notably, the outer lens immediately gets hotter on the edge, while it takes much longer for the heat to 'bleed into' the material. The transient fades quickly, as it is powered by temperature differences. This is evidenced by the relatively minor differences between the rather widely spaced points in time t = 600 s and t = 720 s, where the structure has heated up quickly from the stationary nighttime temperature of -96 °C. The middle lens appears to be the best isolated, retaining its stationary (starting condition) temperature fairly well. This is likely due to it being further away from the internal temperature as well as not directly exposed to the external. Generally, we would expect the lens to have a temperature closer to that of the outside, as a larger area of the structure is exposed to atmospheric conditions as opposed to the controlled heat of the rover. It goes without saying that an instant temperature change between these specific daytime and nighttime conditions is an unrealistic model of the temperature changes an actual lens would go through on Mars, which would both vary in magnitude and slowly change over time.

The calculations of stress show a predictable pattern of having a higher stress in the areas where the lens is exposed to a boundary condition, both in the sense of temperature changes and the boundaries between different materials. Logically, the material expansions and contractions in these areas causes a higher pressure than e.g. the areas of titanium alloy along the sides that don't border a lens. The titanium appears to suffer the majority of the stress, though the small deformations of the the glass lenses could still inhibit the function of the entire unit. The back (right side) of the lens suffers a lot of stress due to both the fixed boundary condition, which prevents the expansion of the lens, and the small titanium knob. We see high points of stress in the outer 'lip' of the titanium alloy (left side), which likely receives this due to the shrinking of the structure's length. As the sides are fixed, the lens has to contract inwards, as seen in image (b) of figure 7. Finally, the points where the glass lenses meet the titanium have a slightly higher stress than their surroundings, likely due to the materials pushing on each other as they expand and contract.

The difference between the von Mises stress field and displacement field with respect to day- and nighttime conditions is reasonable, as the temperature difference between our two boundary conditions is vastly higher during nighttime. The stress appears to scale linearly with temperature, which is in accordance

with theory. As the force of gravity was omitted and inner stresses were assumed to be zero (in addition to previous simplifications), the results will of course not entirely reflect reality.
Generally, these results should not be taken to be even close to accurate. A lot of calculations and concepts have had to be simplified at every step due to constraints in both time, scope and available computing power. A finer mesh can be generated to calculate heat and stress with better accuracy, but this will only marginally improve the results, as the major simplifications come from the modeling of the problem itself, e.g. the lens is not modeled as round, the omission of gravity and unrealistic modeling of the boundary conditions of temperature and fixed positions. However, for a similar project, simplifications can be a necessary first step before developing proper models, as even a greatly simplified model of a project may turn out to be doomed from the start, giving insight into the potential problems one may have to overcome while trying to solve said problem.

We also tried to reduce amount of calculation by using symmetry , but Vonmisses stress field looked different when we just used half of data , if we had more time we could fix that issue. The $MATLAB$ Code for half of data will be attached to this file.

References

[1] Austrell et al. Calfem, a finite element toolbox, 2004.

[2] Division of Solid Mechanics. Assignment in the finite element method, 2021, 2021.

[3] Ottosen Peterson. *Introduction to the finite element method*. Prentice Hall, 1992.

7 Computer code

```
%% setup
clear
close all

p =  matfile('p.mat').p;
e =  matfile('e.mat').e;
t =  matfile('t.mat').t;

%%  f r temperatur.
% a)  Calculate stationary temperature distribution

% Constants

Tinf =40;
Tc = 20;
ac = 100;
Q = 0;
k_g = 0.8;
k_ti = 17;
ko = [k_g k_ti];
L = [];
D = eye(2);
rog = 3860;
roti = 4620;
crog = 670;
croti = 523;
nen = 3;
coord = p';
ndof = length(coord);
enod = t(1:3,:)'; % nodes of elements
nelm = size(enod,1); % number of elements
nnod = size(coord,1);
dof = (1:nnod)'; % dof number is node number


tiarea=2;      % Subdomain lable of titanium part.


Finit = [];

% edof
for ie = 1:nelm
    edof(ie,:) = [ie,enod(ie,:)];
end



[ex,ey] = coordxtr(edof,coord,dof,nen);
K = zeros(ndof);
F =  zeros(ndof,1);
C = zeros(ndof);
thickness = 0.01;

```

```
% kod

for elnr = 1:nelm      % Here we assemble K , C and f.

    if t(4,elnr)==tiarea
        Ke = flw2te(ex(elnr,:),ey(elnr,:),thickness,ko(2).*D);
        Ce = plantml(ex(elnr,:),ey(elnr,:),croti*roti*thickness);
        %         K = assem(edof(elnr,:),K,Ke);
        %         C = assem(edof(elnr,:),C,Ce);
    else
        Ke= flw2te(ex(elnr,:),ey(elnr,:),thickness,ko(1).*D);
        Ce = plantml(ex(elnr,:),ey(elnr,:),crog*rog*thickness);

    end
    K = assem(edof(elnr,:),K,Ke);
    C = assem(edof(elnr,:),C,Ce);
end



convind = [];
conv_segments = [2 12 25 39 32 11 8];   % the sagments with ...
    convection baundary conditions
Finit = F;                              %initial f for transient ...
    (b question)


 % convection boundarys condicions are implemented here.
for kant = 1:length(e)
    %we calculate the length of a  secment here.
    x1 = coord(e(1,kant),1);
    y1 = coord(e(1,kant),2);
    x2 = coord(e(2,kant),1);
    y2 = coord(e(2,kant),2);
    L =  hypot((x1-x2),y1-y2);

    if ismember(e(5,kant),conv_segments)
        F(e(1,kant)) = F(e(1,kant)) + thickness*(L/2)*ac*Tinf;
        F(e(2,kant)) = F(e(2,kant)) +thickness* (L/2)*ac*Tinf;



        if Tinf == -96
            Tinit = 40;
        else
            Tinit = -96;
        end
        Finit(e(1,kant)) =  Finit(e(1,kant)) + ...
            thickness*(L/2)*ac*Tinit;
        Finit(e(2,kant)) =  Finit(e(2,kant)) + ...
            thickness*(L/2)*ac*Tinit;

        Kc = thickness*((ac*L)/6)* [2 1;1 2];
        te =  [e(1,kant) e(2,kant)];
        K(te,te) = K(te,te)+Kc;

    elseif e(5,kant)== 1
```

```
107
108         F(e(1,kant)) = F(e(1,kant)) + thickness*(L/2)*ac*Tc;
109         F(e(2,kant)) = F(e(2,kant)) + thickness*(L/2)*ac*Tc;
110
111         Finit(e(1,kant)) = Finit(e(1,kant)) + thickness*(L/2)*ac*Tc;
112         Finit(e(2,kant)) = Finit(e(2,kant)) + thickness*(L/2)*ac*Tc;
113
114         Kc =thickness*((ac*L)/6)* [2 1;1 2];
115         te =  [e(1,kant) e(2,kant)];
116         K(te,te) = K(te,te)+Kc;
117
118     end
119
120
121
122 end
123
124 %solution and plots of stationary heat distribution
125
126 a_sol = solveq(K,F);
127 %plott
128 ed = extract(edof,a_sol);
129 figure(1)
130 h = fill(ex',ey',ed');
131 if Tinf==-96
132     title('Stationary heat  distribution during night ');
133 else
134     title('Stationary heat distribution during day');
135 end
136 c = colorbar;
137 xlabel('x (m)');
138 ylabel('y (m)');
139 ylabel(c,'^{\circ}C','FontSize',18);
140 set(gca,'fontsize');
141 colormap(hot);
142 hold off;
143
144 %% transient temperatur
145 a = a_sol;
146 time  = 1440;
147 dt = 1;
148
149 avt = zeros(length(a),length(time));
150 avt(:,1) = a;
151
152 % using crank-nickolson we can calculate the transient heatflow ...
        numerically
153
154 for i = 2:time
155     a =  (C+K*dt)\(Finit*dt+C*a);
156     avt(:,i) = a;
157
158
159 end
160
161
162
```

```
163 times=[10,12,15,20];  % time step is in minutes, so  plots ...
        corresponds...
164 % transient solutions after 600s,720s,900s,1200s
165
166 for i = 1:length(times)
167
168     ed = extract(edof,avt(:,times(i)));
169
170     figure
171     h = fill(ex',ey',ed');
172     title(['Transient heat flow after ' num2str(60*times(i)) ' ...
            s'] ,'FontSize', 20);
173     c = colorbar;
174     xlabel('x (m)','FontSize',18);
175     ylabel('y (m)','FontSize',18);
176     ylabel(c,'^{\circ}C','FontSize',18);
177     if Tinf==40
178         caxis([-10, 32]);
179     else
180         caxis([-46,-21]);
181     end
182     set(gca,'fontsize',14)
183     colormap(hot);
184     set(h,'EdgeColor','none');
185
186 end
187
188
189 %% von miesse
190
191 %constants
192
193 ptype = 2;
194 tcond =20;
195 E_ti = 110E9;
196 E_gl = 67E9 ;
197 v_ti = 0.34;
198 v_gl = 0.2;
199 alpha_ti = 9.4E-6;
200 alpha_gl = 7E-6;
201 thickness = 0.01;
202
203 ep = [ ptype , thickness];
204
205
206
207
208 %D matrices
209 D_gl = E_gl / ((1+v_gl)*(1-2*v_gl)) .* [1-v_gl, v_gl, 0;
210     v_gl, 1-v_gl, 0;
211     0    0      0.5*(1-2*v_gl)];
212
213 D_ti = E_ti / ((1+v_ti)*(1-2*v_ti)) .* [1-v_ti, v_ti, 0;
214     v_ti, 1-v_ti, 0;
215     0    0      0.5*(1-2*v_ti)];
216
217
```

```
218
219 %kod
220
221
222 % building up edof_S matrisen
223 dof_S = [(1:nnod)',(nnod+1:2*nnod)']; % give each dof a number
224 for ie = 1:nelm
225     edof_S(ie,:) = [ie dof_S(enod(ie,1),:), ...
            dof_S(enod(ie,2),:),dof_S(enod(ie,3),:)];
226
227 end
228
229 Ks = zeros(2*nnod);
230 Fs = zeros(2*nnod,1);
231 temperature = a_sol;
232 Δ_t_enod = zeros(nelm,3);
233
234
235 % assemblig KS and Fs to calculate displacements
236 for i = 1:nelm
237     Δ_t_enod(i,:) = [temperature(t(1,i)) temperature(t(2,i))...
238      temperature(t(3,i))]; % Δ_t  f  r  varje element
239     Δ_t=max(Δ_t_enod(i,:))-tcond;
240
241     if t(4,i)==tiarea
242         De = D_ti;
243         D_eps_Δ_t_ti = ((E_ti*alpha_ti*Δ_t)/(1-2*v_ti))*[1;1;0];
244         fue = plantf(ex(i,:),ey(i,:),ep,D_eps_Δ_t_ti');
245     else
246         De = D_gl;
247         D_eps_Δ_t_gl = (E_gl/(1-2*v_gl))*alpha_gl*Δ_t.*[1;1;0];
248         fue = plantf(ex(i,:),ey(i,:),ep,D_eps_Δ_t_gl');
249     end
250     Keu = plante(ex(i,:),ey(i,:),ep,De);
251     [Ks, Fs] = assem(edof_S(i,:),Ks,Keu, Fs, fue);
252 end
253
254
255
256
257
258 % boundary conditions
259
260
261 % segments with defined boundary conditions
262 uyo = [21 22 23 24  ];
263 uxo=1;
264 bc = [];
265
266
267 for i = 1:length(e)
268
269     if ismember(e(5,i),uyo)      % om vi hittar en
270
271         bc = [bc;e(2,i)+nnod , 0];
272         bc = [bc;e(1,i)+nnod , 0 ];
273     end
```

```
274     if(e(5,i) == uxo)
275
276         bc  = [bc; e(1,i),0];
277         bc = [bc; e(2,i),0];
278
279     end
280 end
281
282 %calculating vonmises
283
284 F = zeros(nelm,1);
285 u  = solveq(Ks,Fs,bc);
286 ed = extract(edof_S,u);
287 vonmises_e = zeros(nelm,1); %vonmises  f  r  varje element
288
289 sigma = [];
290 for i = 1:nelm
291
292
293     Δ_t=max(Δ_t_enod(i,:))-tcond;
294     if t(4,i)==2
295         [sig,eps] = plants(ex(i,:),ey(i,:),ep,D_ti,ed(i,:)) ;
296         %         D_eps_Δ_t_ti = ...
                (E_ti/(1-2*v_ti))*alpha_ti*Δ_t*[1 1 0];
297         eps_Δ_t_ti = (1+v_ti)*alpha_ti*Δ_t*[1 1 0];
298
299         sigma=D_ti*(eps-eps_Δ_t_ti)';
300
301         sigma_zz = v_ti*(sigma(1)+sigma(2))-alpha_ti*E_ti*Δ_t;
302
303
304     else
305         [sig,eps] = plants(ex(i,:),ey(i,:),ep,D_gl,ed(i,:));
306         eps_Δ_t_gl =(1+v_gl)*alpha_gl*Δ_t*[1 1 0];
307         sigma = D_gl*(eps-eps_Δ_t_gl)';
308         sigma_zz = v_gl*(sigma(1)+sigma(2))-alpha_gl*E_gl*Δ_t;
309
310     end
311
312     es = sigma;
313     vonmises_e(i) = sqrt((es(1)^2 + es(2)^2+ sigma_zz^2 ...
            -(es(1)*es(2)) ...
314         -(es(1)*sigma_zz) - (es(2)*sigma_zz) + 3*es(3)^2));
315
316
317
318 end
319
320
321 Vonmises_n = zeros(ndof,1);
322
323 for i = 1:nnod
324
325     [r,¬] =  find(edof(:,2:4)==i);
326     vonmises_n(i) = sum(vonmises_e(r)/length(r));
327
328 end
```

```
329
330
331 %plot
332
333 ed = extract(edof,vonmises_n);
334 figure
335 h = fill(ex',ey',ed');
336 set(h,'EdgeColor','none')
337 set(gca,'fontsize',14)
338 title('Von Mises stress field','FontSize', 18);
339 c = colorbar;
340 xlabel('x (m)','FontSize',18);
341 ylabel('y (m)','FontSize',18);
342 ylabel(c,'Pa','FontSize',18);
343
344 %%
345
346 % Calculate displaced coordinates
347
348
349 ed = extract(edof_S,u);
350 mag = 100; % Magnification (due to small deformations)
351 exd = ex + mag*ed(:,1:2:end);
352 eyd = ey + mag*ed(:,2:2:end);
353 figure()
354 patch(ex',ey',[0 0 0],'EdgeColor','none','FaceAlpha',0.3)
355 hold on
356 patch(exd',eyd',[0 0 0],'FaceAlpha',0.3)
357 axis equal
358 title('Displacement field [Magnitude enhancement 100]')
359
360
361
362 %% frontlens displacement
363 ed = extract(edof_S,u);
364 squaresum = 0;
365 for i = 1:nelm
366
367     if t(4,i)==3
368         intNTN = plantmlmod(ex(i,:),ey(i,:));
369         squaresum = squaresum+ed(i,:)*intNTN*ed(i,:)';
370
371     end
372 end
```