Solution to Exam (2025-04-16)

MHA021 Finite Element Method

Problem 1

1.1 Task 1a

Multiply the differential equation with an (arbitrary) test function $v(x)$ and integrate over the interval

$$-\int_0^L v\frac{d}{dx}\left[EA\frac{du}{dx}\right]\,\mathrm{d}x = \int_0^L v\,b\,\mathrm{d}x$$

IBP. of the left-hand side gives (after moving the boundary terms to the right-hand side)

$$\int_0^L EA\,\frac{dv}{dx}\,\frac{du}{dx}\,\mathrm{d}x = \int_0^L v\,b\,\mathrm{d}x + v(L)\left[EA\,\frac{du}{dx}\right]_{x=L} - v(0)\,\left[EA\,\frac{du}{dx}\right]_{x=0}$$

At $x=0$, we can substitute the the boundary term given in the strong form ($EA\,\dfrac{\mathrm{d}u}{\mathrm{d}x}(0) = -P$) $\Rightarrow$ the weak form:

Find u such that

$$\int_0^L EA\,\frac{dv}{dx}\,\frac{du}{dx}\,\mathrm{d}x = \int_0^L v\,b\,\mathrm{d}x + v(L)\,\underbrace{\left[EA\,\frac{du}{dx}\right]_{x=L}}_{\text{reaction force}=R} + v(0)\,P$$

where $u(L)=0$.

1.2 Task 1b

Approximate u using a linear combination of shape functions: $u \approx u_h = \sum_i N_i(x)a_i = \mathbf{N}\,\mathbf{a}$, where $N = [N_1(x)\ N_2(x)\dots N_n(x)]$ is a row vector with the shape functions and $a = \begin{bmatrix}a_1 & a_2 & \dots & a_n\end{bmatrix}^{\mathrm{T}}$ are the degrees of freedom. Use the same shape functions for the test function $v(x) = \sum_i N_i(x)c_i = \mathbf{N}\,\mathbf{c}$. Inserting into the weak form while using the notation $\dfrac{\mathrm{d}}{\mathrm{d}x}\mathbf{N}(x) = \mathbf{B}$ gives

$$\mathbf{c}^{\mathrm{T}}\int_0^L EA\,\mathbf{B}^{\mathrm{T}}\,\mathbf{B}\,\mathrm{d}x\,\mathbf{a} = \mathbf{c}^{\mathrm{T}}\int_0^L \mathbf{N}^{\mathrm{T}}\,b\,\mathrm{d}x + \mathbf{c}^{\mathrm{T}}\,\mathbf{N}^{\mathrm{T}}(L)R + \mathbf{c}^{\mathrm{T}}\,\mathbf{N}^{\mathrm{T}}(0)\,P$$

which should hold for arbitrary $\mathbf{c}$ $\Rightarrow$...

$$\int_0^L EA\,\mathbf{B}^{\mathrm{T}}\,\mathbf{B}\,\mathrm{d}x\,\mathbf{a} = \int_0^L \mathbf{N}^{\mathrm{T}}\,b\,\mathrm{d}x + \mathbf{N}^{\mathrm{T}}(L)R + \mathbf{N}^{\mathrm{T}}(0)\,P$$

or simply $\mathbf{K}\,\mathbf{a} = \mathbf{f}_l + \mathbf{f}_{\mathrm{b}}$ with

$$\mathbf{K} = \int_0^L EA\,\mathbf{B}^{\mathrm{T}}\,\mathbf{B}\,\mathrm{d}x, \quad \mathbf{f}_{\mathrm{l}} = \int_0^L \mathbf{N}^{\mathrm{T}}\,b\,\mathrm{d}x, \quad \mathbf{f}_{\mathrm{b}} = \mathbf{N}^{\mathrm{T}}(L)R + \mathbf{N}^{\mathrm{T}}(0)\,P$$

1.3 Task 1c

Consider one element between coordinates x_i and x_{i+1} and linear shape functions. $N_1^e = -\frac{1}{L}(x - x_{i+1})$ and $N_2^e = \frac{1}{L}(x - x_i)$ which gives

$$\mathbf{N}^e = [N_1^e, N_2^e] \quad \Rightarrow \mathbf{B}^e = \frac{\mathrm{d}}{\mathrm{d}x}\mathbf{N}^e = \frac{1}{L}[-1, 1]$$

The element stiffness matrix and load vector becomes

$$\mathbf{K}^e = \int_{x_i}^{x_{i+1}} EA\,(\mathbf{B}^e)^{\mathrm{T}}\,\mathbf{B}^e\,\mathrm{d}x = \ldots = \frac{EA}{L^e}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad \mathbf{f}_1^e = \int_{x_i}^{x_{i+1}} (\mathbf{N}^e)^{\mathrm{T}} b_0\,\mathrm{d}x = \ldots = \frac{b_0\,L^e}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

1.4 Task 1d

Element stiffness matrix and load vector becomes the same for both elements ($L^e = L/2$)

$$\mathbf{K}^1 = \mathbf{K}^2 = \frac{2EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \mathbf{f}_1^1 = \mathbf{f}_1^2 = \frac{b_0\,L}{4}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Assemble the system of equations $\Rightarrow$

$$\frac{EA}{L}\begin{bmatrix} 2 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 2 \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \\ 0 \end{bmatrix} = \frac{b\,L}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} P \\ 0 \\ R \end{bmatrix}$$

Solving the system using numerical values gives the axial displacement at $x = 0$ as $u(0) = a_1 = 0\,\mathrm{m}$.

1.5 Task 1e

If we add a spring to the left end we will get the modified boundary condition $EA\,\dfrac{\mathrm{d}u}{\mathrm{d}x}(0) = -P + k\,u(L)$ which gives an additional term in the left side of the weak form

$$\ldots + v(0)\,k\,u(0) = \ldots$$

which in turn gives the additional contribution to the global stiffness matrix

$$\ldots + \mathbf{N}^{\mathrm{T}}(0)\,k\,\mathbf{N}(0) = \ldots$$

The modified problem will lead to a similar system of equations as in the previous task but with the additional contribution of $3EA/L$ to the stiffness matrix in element K_{11}:

$$\frac{EA}{L}\begin{bmatrix} 2+3 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 2 \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \\ 0 \end{bmatrix} = \frac{b\,L}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} P \\ 0 \\ R \end{bmatrix}$$

Solving this new system gives the (same) displacement $u(0) = a_1 = 0\,\text{m}$.

Assembling and solving the system of equations can be done as follows (task d and e):

```
import numpy as np
import calfem.core as cfc

EA = 200e9*2e-4
P = 1e3
L = 1
b0 = -2*P/L
K = np.zeros((3, 3))
f = np.zeros((3, 1))
f[0] = P # Boundary load

Le = L/2
K1 = K2 = EA/Le * np.array([[1, -1],[-1, 1]])
fl1 = fl2 = b0*Le/2 * np.array([[1],[1]])
cfc.assem(np.array([1, 2]), K, K1, f, fl1)
cfc.assem(np.array([2, 3]), K, K2, f, fl2)

# For e-task add spring stiffness to K11
k = 3*EA/L
K[0, 0] += k

a, r = cfc.solveq(K, f, np.array([3]), np.array([0]))
a
```

Problem 2: Heat equation

2.1 Task 2a

$$\nabla^{\mathrm{T}}\mathbf{q} = h \text{ in } \Omega \tag{2.1}$$
$$T(\mathbf{x}) = 0\,^{\circ}\mathrm{C}, \quad \mathbf{x} \text{ on } \Gamma_{\text{left}} \tag{2.2}$$
$$T(\mathbf{x}) = 20\,^{\circ}\mathrm{C}, \quad \mathbf{x} \text{ on } \Gamma_{\text{right}} \tag{2.3}$$
$$\mathbf{q}^{\mathrm{T}}(\mathbf{x})\mathbf{n} = 0, \quad \mathbf{x} \text{ on } \Gamma_{\mathrm{h}} \tag{2.4}$$

where the Neumann boundary, $\Gamma_{\mathrm{h}} = \Gamma \setminus [\Gamma_{\text{left}} \cup \Gamma_{\text{right}}]$, is the entire boundary excluding the left and right parts.

2.2 Task 2b

We multiply the Partial Differential Equation (PDE) by an arbitrary scalar test function, $v(\mathbf{x})$, and integrate over the domain, Ω, resulting in

$$\int_{\Omega} v\ \nabla^{\mathrm{T}}\mathbf{q}\ \mathrm{d}\Omega = \int_{\Omega} v\ h\ \mathrm{d}\Omega \tag{2.5}$$

Applying Green-Gauss theorem to the left hand side, results in

$$\int_{\Gamma} v\ \mathbf{n}^{\mathrm{T}}\mathbf{q}\ \mathrm{d}\Gamma - \int_{\Omega} [\nabla v]^{\mathrm{T}}\ \mathbf{q}\ \mathrm{d}\Omega = \int_{\Omega} v\ h\ \mathrm{d}\Omega \tag{2.6}$$

Finally, we split the boundary terms into the Dirichlet ($\Gamma_g = \Gamma_{\text{left}} \cup \Gamma_{\text{right}}$) and Neumann ($\Gamma_h$) parts, insert the constitutive law, $\mathbf{q} = -k\nabla T$, and re-arrange to get

$$\int_{\Omega} [\nabla v]^{\mathrm{T}}\ [k\ \nabla T]\ \mathrm{d}\Omega = \int_{\Omega} v\ h\ \mathrm{d}\Omega - \int_{\Gamma_h} v\ q_{\mathrm{n}}\ \mathrm{d}\Gamma - \int_{\Gamma_g} v\ \mathbf{n}^{\mathrm{T}}\mathbf{q}\ \mathrm{d}\Gamma \tag{2.7}$$

Still subject to the constraints, $T(\mathbf{x}) = 0\,^{\circ}\mathrm{C}$ for x on Γ_{left} and $T(\mathbf{x}) = 20\,^{\circ}\mathrm{C}$ for x on Γ_{right}.

2.3 Task 2c

To get the FE form, we introduce the approximation of the test function, $v(\mathbf{x}) \approx v_h(\mathbf{x}) = \sum_{i=1}^{N_d} N_i(\mathbf{x})c_i = \mathbf{N}\ \mathbf{c}$, where c_i are arbitrary weights, that are coordinate independent. This results

in

$$\sum_{i=1}^{N_d} c_i \underbrace{\left[\int_\Omega \left[\nabla N_i(\mathbf{x})\right]^{\mathrm{T}} \left[k\ \nabla T\right]\ \mathrm{d}\Omega - \left[\int_\Omega N_i(\mathbf{x})\ h\ \mathrm{d}\Omega - \int_{\Gamma_h} N_i(\mathbf{x})\ q_{\mathrm{n}}\ \mathrm{d}\Gamma - \int_{\Gamma_g} N_i(\mathbf{x})\ \mathbf{n}^{\mathrm{T}}\mathbf{q}\ \mathrm{d}\Gamma\right]\right]}_{r_i} = 0 \tag{2.8}$$

Since c_i are arbitrary coefficients, and this has to hold for any c_i, this implies that $r_i = 0$ has to hold. We can see this by considering the case that $c_\alpha = 1$, and $c_i = 0$ if $i \neq 0$. Then $r_\alpha = 0$ follows, and this has to hold for any $1 \leq \alpha \leq N_d$. We thus obtain the FE form,

$$\int_\Omega \left[\nabla N_i(\mathbf{x})\right]^{\mathrm{T}} \left[k\ \nabla T\right]\ \mathrm{d}\Omega = \int_\Omega N_i(\mathbf{x})\ h\ \mathrm{d}\Omega - \int_{\Gamma_h} N_i(\mathbf{x})\ q_{\mathrm{n}}\ \mathrm{d}\Gamma - \int_{\Gamma_g} N_i(\mathbf{x})\ \mathbf{n}^{\mathrm{T}}\mathbf{q}\ \mathrm{d}\Gamma \tag{2.9}$$

and can finally insert the FE-approximation for the temperature, $T(\mathbf{x}) \approx T_h(\mathbf{x}) = \sum_{i=1}^{N_d} N_i(\mathbf{x}) a_i = \mathbf{Na}$, to obtain,

$$\int_\Omega \left[\nabla N_i(\mathbf{x})\right]^{\mathrm{T}} \left[k\ \nabla N_j\right]\ \mathrm{d}\Omega a_j = \int_\Omega N_i(\mathbf{x})\ h\ \mathrm{d}\Omega - \int_{\Gamma_h} N_i(\mathbf{x})\ q_{\mathrm{n}}\ \mathrm{d}\Gamma - \int_{\Gamma_g} N_i(\mathbf{x})\ \mathbf{n}^{\mathrm{T}}\mathbf{q}\ \mathrm{d}\Gamma \tag{2.10}$$

$$T_h(\mathbf{x}) = 0\,^\circ\mathrm{C} \text{ on } \Gamma_{\text{left}} \tag{2.11}$$

$$T_h(\mathbf{x}) = 20\,^\circ\mathrm{C} \text{ on } \Gamma_{\text{right}} \tag{2.12}$$

Using that $q_n(\mathbf{x}) = 0$ on Γ_h, we finally get

$$\underbrace{\int_\Omega \left[\nabla N_i(\mathbf{x})\right]^{\mathrm{T}} \left[k\ \nabla N_j\right]\ \mathrm{d}\Omega}_{K_{ij}}\, a_j = \underbrace{\int_\Omega N_i(\mathbf{x})\ h\ \mathrm{d}\Omega - \int_{\Gamma_g} N_i(\mathbf{x})\ \mathbf{n}^{\mathrm{T}}\mathbf{q}\ \mathrm{d}\Gamma}_{f_i} \tag{2.13}$$

$$T_h(\mathbf{x}) = 0\,^\circ\mathrm{C} \text{ on } \Gamma_{\text{left}} \tag{2.14}$$

$$T_h(\mathbf{x}) = 20\,^\circ\mathrm{C} \text{ on } \Gamma_{\text{right}} \tag{2.15}$$

where the reaction flux, $q_n(\mathbf{x})$ is unknown on Γ_g.

2.4 Task 2d

We now have the equation system,

$$\mathbf{Ka} = \mathbf{f} \tag{2.16}$$

But degree of freedoms, a_i, that are on the Dirichlet boundaries are known. The corresponding reaction fluxes, f_i, are then unknown. Therefore, we partition our equation system by indicating

degree of freedoms where a_i is constrained (known) with index c, and free (unknown) as f. This gives the partitioned equation system

$$\begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{fc} \\ \mathbf{K}_{cf} & \mathbf{K}_{cc} \end{bmatrix} \begin{bmatrix} \mathbf{a}_f \\ \mathbf{a}_c \end{bmatrix} = \begin{bmatrix} \mathbf{f}_f \\ \mathbf{f}_c \end{bmatrix} \tag{2.17}$$

Such that we have the coupled problems

$$\mathbf{K}_{ff}\mathbf{a}_f + \mathbf{K}_{fc}\mathbf{a}_c = \mathbf{f}_f \tag{2.18}$$
$$\mathbf{K}_{cf}\mathbf{a}_f + \mathbf{K}_{cc}\mathbf{a}_c = \mathbf{f}_c \tag{2.19}$$

Which upon reorganization to solve the problem yields,

$$\mathbf{a}_f = \mathbf{K}_{ff}^{-1}\left[\mathbf{f}_f - \mathbf{K}_{fc}\mathbf{a}_c\right] \tag{2.20}$$
$$\mathbf{f}_c = \mathbf{K}_{cf}\mathbf{a}_f + \mathbf{K}_{cc}\mathbf{a}_c \tag{2.21}$$

which, when solved in sequence, all variables on the right-hand-side are known.

2.5 Task 2e

When approximated with the element's base functions, the flux inside the element becomes

$$\mathbf{q} = -k\nabla N_i^e(\mathbf{x})a_i^e \tag{2.22}$$

For a three-noded triangle, the global shape functions are linear wrt. $\mathbf{x}$, such that the gradients become constant, i.e.

$$\nabla N_1^e = \left[y_2^e - y_3^e,\ x_3^e - x_2^e\right]^{\mathrm{T}}/2A^e \tag{2.23}$$
$$\nabla N_2^e = \left[y_3^e - y_1^e,\ x_1^e - x_3^e\right]^{\mathrm{T}}/2A^e \tag{2.24}$$
$$\nabla N_3^e = \left[y_1^e - y_2^e,\ x_2^e - x_1^e\right]^{\mathrm{T}}/2A^e \tag{2.25}$$

Therefore, the flux $\mathbf{q}$ is also constant inside the element if the conductivity, k, is constant.

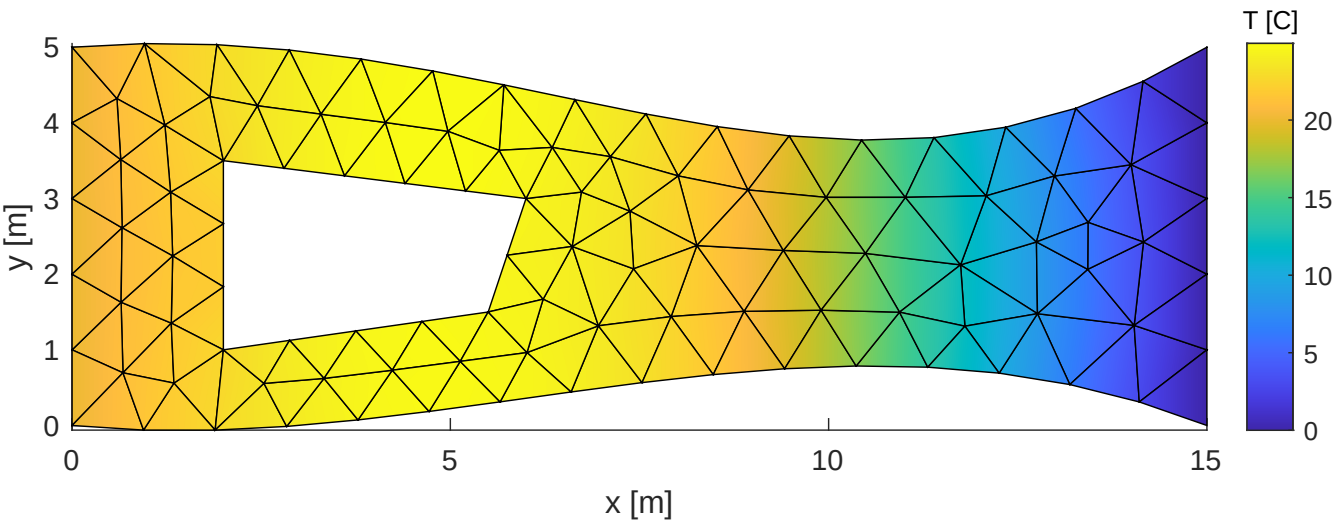
2.6 Task 2f

$\mathbf{q} = -\mathbf{D}\ \nabla T = -k\mathbf{I}\ \nabla T = -k\nabla T$, i.e. $\mathbf{D} = k\mathbf{I}$.

```
# General packages
import numpy as np
```

```
import scipy.io
import matplotlib.pyplot as plt
import matplotlib.tri as tri
# CALFEM packages
import calfem.core as cfc
import calfem.vis_mpl as cfv



# Load mesh data
mesh = scipy.io.loadmat('mesh_task2.mat')

Coord = mesh['Coord']                          # [x, y] coords for each node
Dofs = mesh['Dofs']
Edof = mesh['Edof']                            # [element number, dof1, dof2, dof3]
Ex = mesh['Ex']
Ey = mesh['Ey']
right_dofs = mesh['right_dofs']
left_dofs = mesh['left_dofs']

# Plot the mesh
plotpar = np.array([1, 1, 2]) # parameters for line style, color, marker
# cfv.eldraw2(Ex, Ey, plotpar)
cfv.eldraw2(Ex, Ey)
plt.xlabel("x [m]")
plt.ylabel("y [m]")
plt.show()

nel=np.shape(Edof)[0]
ndofs=np.max(Edof[:,1:])
dimension = 2

T1=20
T2=0

k=np.eye(dimension)

h=0.5
t=0.01

Ke=np.zeros([3,3])
K=np.zeros([ndofs,ndofs])

fe=np.zeros([3,1])
```

```
f=np.zeros([ndofs,1])


for el in range(nel):
    Ke,fe=cfc.flw2te(Ex[el,:],Ey[el,:],[t],k,[h])
    K,f=cfc.assem(Edof[el,1:],K,Ke,f,fe)


bcpresc=np.vstack([right_dofs,left_dofs]).flatten()

bcVal1=np.zeros_like(right_dofs)
bcVal2=np.ones_like(left_dofs)*T1


bcVal=np.vstack([bcVal1,bcVal2]).flatten()
a,q=cfc.solveq(K,f,bcpresc,bcVal)

print(a)

a = np.array(a).flatten()
x,y = Coord.T

Elements_dofs = np.array(Edof[:, 1:4] - 1)

triang = tri.Triangulation(x, y, Elements_dofs)

plt.figure()
color = plt.tripcolor(triang, a, shading='gouraud', cmap='hot')
plt.colorbar(color)
plt.triplot(triang, color='k', lw=0.7)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Temperature profile')
plt.axis('equal')
plt.show()
```



Problem 3: Linear elasticity

3.1 Task 3a

Considering the local coordinates of $\mathbf{x}_\mathrm{A}$, $\xi_\mathrm{A} = [0.25, 0.15]^\mathrm{T}$, the global coordinates of $\mathbf{x}_\mathrm{A}$ are given as,

$$\mathbf{x}_\mathrm{A} = \sum_{\alpha=1}^{3} \mathbf{x}_\alpha^\mathrm{e} M_\alpha^\mathrm{e}(\xi_\mathrm{A}) \tag{3.1}$$

where M_α^e are the shape functions on the reference quadrilateral. Implementing this in MATLAB gives, $\mathbf{x}_\mathrm{A} = [0.0169,\ 0.0139]^\mathrm{T}\mathrm{m}$ for the ordering in the new formula sheet.

3.2 Task 3b

The displacement field is calculated as,

$$\mathbf{u}(\xi) = \sum_{\alpha=1}^{6} a_\alpha^\mathrm{e} \mathbf{N}_\alpha^\mathrm{e}(\xi) \tag{3.2}$$

where a_α^e represents the degrees of freedom and $\mathbf{N}_\alpha^\mathrm{e}(\xi)$ the vector shape functions at position ξ. These are given as,

$$\mathbf{N}_1^\mathrm{e} = \begin{bmatrix} M_1^e \\ 0 \end{bmatrix},\ \mathbf{N}_2^\mathrm{e} = \begin{bmatrix} 0 \\ M_1^e \end{bmatrix},\ \mathbf{N}_3^\mathrm{e} = \begin{bmatrix} M_2^e \\ 0 \end{bmatrix},\ \mathbf{N}_4^\mathrm{e} = \begin{bmatrix} 0 \\ M_2^e \end{bmatrix},$$
$$\mathbf{N}_5^\mathrm{e} = \begin{bmatrix} M_3^e \\ 0 \end{bmatrix},\ \mathbf{N}_6^\mathrm{e} = \begin{bmatrix} 0 \\ M_3^e \end{bmatrix},\ \mathbf{N}_7^\mathrm{e} = \begin{bmatrix} M_4^e \\ 0 \end{bmatrix},\ \mathbf{N}_8^\mathrm{e} = \begin{bmatrix} 0 \\ M_4^e \end{bmatrix}$$

where M_α^{e} is the scalar shape functions in the formula sheet and we have skipped the function dependency $f(\xi)$ for brevity on $\mathbf{N}_\alpha^{\mathrm{e}}$ and M_α^{e}. Implementing this in MATLAB gives $\mathbf{u}(\xi_{\mathrm{A}}) = [0.03747, 0.01519]\mathrm{mm}$

3.3 Task 3c

We aim to calculate the stiffness matrix,

$$\mathbf{K}^{\mathrm{e}} = \int_{\Omega^{\mathrm{e}}} [\mathbf{B}^{\mathrm{e}}]^{\mathrm{T}} t\mathbf{D}\mathbf{B}^{\mathrm{e}} \mathrm{d}\Omega$$

with a single quadrature point. Taking from the quadrature rule in the formula sheet, we have the quadrature point $\xi_1 = [0, 0]^{\mathrm{T}}$ with the weight, $w_1 = 4$. This implies that the integral can be approximated as

$$\mathbf{K}^{\mathrm{e}} \approx g(\xi_1) \det(J(\xi_1)) w_1, \quad g(\xi) = [\mathbf{B}^{\mathrm{e}}]^{\mathrm{T}} t\mathbf{D}\mathbf{B}^{\mathrm{e}}$$

Here, the B-matrix is given as,

$$\mathbf{B}^{\mathrm{e}} = \begin{bmatrix} \frac{\partial M_1}{\partial x_1} & 0 & \frac{\partial M_2}{\partial x_1} & 0 & \frac{\partial M_3}{\partial x_1} & 0 & \frac{\partial M_4}{\partial x_1} & 0 \\ 0 & \frac{\partial M_1}{\partial x_2} & 0 & \frac{\partial M_2}{\partial x_2} & 0 & \frac{\partial M_3}{\partial x_2} & 0 & \frac{\partial M_4}{\partial x_2} \\ \frac{\partial M_1}{\partial x_2} & \frac{\partial M_1}{\partial x_1} & \frac{\partial M_2}{\partial x_2} & \frac{\partial M_2}{\partial x_1} & \frac{\partial M_3}{\partial x_2} & \frac{\partial M_3}{\partial x_1} & \frac{\partial M_4}{\partial x_2} & \frac{\partial M_4}{\partial x_1} \end{bmatrix}$$

Furthermore, we have that

$$\frac{\partial M_i}{\partial \xi} = \frac{\partial M_i}{\partial \mathbf{x}} \underbrace{\frac{\partial \mathbf{x}}{\partial \xi}}_{\mathbf{J}} \Rightarrow \frac{\partial M_i}{\partial \mathbf{x}} = \frac{\partial M_i}{\partial \xi} \mathbf{J}^{-1} = \mathbf{J}^{-T} \frac{\partial M_i}{\partial \xi}$$

where $\mathbf{J}$ is the Jacobian of the isoparametric mapping,

$$J_{ij} = \sum_{\alpha=1}^{3} \mathbf{x}_\alpha^{\mathrm{e}} \frac{\partial M_\alpha^e}{\partial \xi_j} \tag{3.3}$$

The derivatives, $\partial M_i / \partial \xi$, in the reference element, is easily calculated, i.e.

$$\begin{pmatrix} M_1(\xi) = \frac{[1-\xi_1][1-\xi_2]}{4} & M_2(\xi) = \frac{[1+\xi_1][1-\xi_2]}{4} & M_3(\xi) = \frac{[1+\xi_1][1+\xi_2]}{4} & M_4(\xi) = \frac{[1-\xi_1][1+\xi_2]}{4} \\ \frac{\partial M_1}{\partial \xi} = \frac{[-1+\xi_2,\ -1+\xi_1]^{\mathrm{T}}}{4} & \frac{\partial M_2}{\partial \xi} = \frac{[1-\xi_2,\ -1-\xi_1]^{\mathrm{T}}}{4} & \frac{\partial M_3}{\partial \xi} = \frac{[1+\xi_2,\ 1+\xi_1]^{\mathrm{T}}}{4} & \frac{\partial M_4}{\partial \xi} = \frac{[-1-\xi_2,\ 1-\xi_1]^{\mathrm{T}}}{4} \end{pmatrix}$$

Calculating these in Python (including task a and b), yields

```
import numpy as np
import calfem.core as cfc

### 3)a)
t=0.025
E=210*10**9
nu=0.3
xi1=0.25
xi2=0.15

N1e=0.25*(1-xi1)*(1-xi2)
N2e=0.25*(1+xi1)*(1-xi2)
N3e=0.25*(1+xi1)*(1+xi2)
N4e=0.25*(1-xi1)*(1+xi2)

Ne=np.array([N1e,N2e,N3e,N4e])

xe=[0.014,0.021,0.018,0.012]
xe = np.array(xe)
ye=[0.010,0.009,0.018,0.016]
ye = np.array(ye)

x=Ne@xe
y=Ne@ye

x_A = [x,y]
print("x_A = ", x_A)
print("\n")


### 3)b)

Ne = np.array([
    [N1e,0,N2e,0,N3e,0,N4e,0],
    [0,N1e,0,N2e,0,N3e,0,N4e]
])

ae = np.array([4.1,1.0,3.5,1.6,3.8,1.7,3.7,1.5])

u_A = (Ne @ ae).T
print("u_A = ", u_A)
print("\n")

### 3)c)
```

```
xi1 = 0
xi2 = 0

w = 4

D = cfc.hooke(1, E, nu)

dN1dxi1=-0.25*(1-xi2)
dN1dxi2=-0.25*(1-xi1)
dN2dxi1=0.25*(1-xi2)
dN2dxi2=-0.25*(1-xi1)
dN3dxi1=0.25*(1+xi2)
dN3dxi2=0.25*(1+xi1)
dN4dxi1=-0.25*(1+xi2)
dN4dxi2=0.25*(1-xi1)

dNdxi1xi2=np.array([
     [dN1dxi1,dN2dxi1,dN3dxi1,dN4dxi1],
     [dN1dxi2,dN2dxi2,dN3dxi2,dN4dxi2]
])

print(dNdxi1xi2)

dxdxi1=np.array([
    [dN1dxi1,dN2dxi1,dN3dxi1,dN4dxi1]
])@xe

dxdxi2=np.array([
    [dN1dxi2,dN2dxi2,dN3dxi2,dN4dxi2]
])@xe

dydxi1=np.array([
    [dN1dxi1,dN2dxi1,dN3dxi1,dN4dxi1]
])@ye

dydxi2=np.array([
    [dN1dxi2,dN2dxi2,dN3dxi2,dN4dxi2]
])@ye

J=np.array([
    [dxdxi1,dxdxi2],
    [dydxi1,dydxi2]
]).reshape([2,2])
```

```
print("J = ", J)
print("\n")
Jinv = np.linalg.inv(J)
print("J_inv = ", Jinv)
print("\n")
dNdx = Jinv.T @ dNdxi1xi2
print("dNdx = ", dNdx)
print("\n")
Be=np.array([
    [dNdx[0][0],0,dNdx[0][1],0,dNdx[0][2],0,dNdx[0][3],0],
    [0,dNdx[1][0],0,dNdx[1][1],0,dNdx[1][2],0,dNdx[1][3]],
    [dNdx[1][0],dNdx[0][0],dNdx[1][1],dNdx[0][1],dNdx[1,2],dNdx[0][2],dNdx[1][3],dNdx[0][3]]
])
print("Be = ", Be)
print("\n")
detJ=np.linalg.det(J)


Ke=Be.T@D@Be*detJ*w*t

print("Ke = ", Ke/1e9)
```

$$\mathbf{K}^{\mathrm{e}} = \begin{bmatrix} 2.2313 & 1.1813 & -1.2519 & -0.4846 & -2.2313 & -1.1813 & 1.2519 & 0.4846 \\ 1.1813 & 2.8313 & -0.3404 & 0.4731 & -1.1813 & -2.8313 & 0.3404 & -0.4731 \\ -1.2519 & -0.3404 & 2.0077 & -0.6000 & 1.2519 & 0.3404 & -2.0077 & 0.6000 \\ -0.4846 & 0.4731 & -0.6000 & 1.1077 & 0.4846 & -0.4731 & 0.6000 & -1.1077 \\ -2.2313 & -1.1813 & 1.2519 & 0.4846 & 2.2313 & 1.1813 & -1.2519 & -0.4846 \\ -1.1813 & -2.8313 & 0.3404 & -0.4731 & 1.1813 & 2.8313 & -0.3404 & 0.4731 \\ 1.2519 & 0.3404 & -2.0077 & 0.6000 & -1.2519 & -0.3404 & 2.0077 & -0.6000 \\ 0.4846 & -0.4731 & 0.6000 & -1.1077 & -0.4846 & 0.4731 & -0.6000 & 1.1077 \end{bmatrix} \mathrm{kN/m}$$