Nebenfachpraktikum WW8

Solid Mechanics and Linear Finite Element Analysis

Handwritten Solution and Code Explanations

2.Submission

Rundong Jia (rundong.jia@fau.de)

Supervisor: Jonas Ritter, M. Sc.

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Erlangen 19.05.2024

Task 1.1:

$\rho \ddot{\boldsymbol{u}} = \mathrm{div}(\boldsymbol{\sigma}) + \rho \boldsymbol{b}$, balance eqn. for linear momentum

When steady state : no acceleration

$\Rightarrow$ $\ddot{u}$ will be a zero matrix

$\Rightarrow$ left side can be written as 0 (since it's integral = 0)

$0 = \mathrm{div}(\boldsymbol{\sigma}) + \rho \boldsymbol{b}$

Example body force: forces of electric fields / magnetic fields

Neglect body forces : a non-conductive material in space (vacuum

Task 2.1

$$\sigma_{ij} = \mathbb{C}_{ijkl}\,\varepsilon_{kl}$$

In general, $\mathbb{C}$ has $3^4 = 81$ componenes

Assume a, b are known integers,

a) if stress tensor is symmetric: $\sigma_{ab} = \sigma_{ba}$

From Hooke's law:

$$\sigma_{ab} = \mathbb{C}_{abkl}\,\varepsilon_{kl} \quad , \quad \sigma_{ba} = \mathbb{C}_{bakl}\,\varepsilon_{kl}$$

$$\Rightarrow \mathbb{C}_{abkl}\,\varepsilon_{kl} = \mathbb{C}_{bakl}\,\varepsilon_{kl}$$

$$\Rightarrow \varepsilon_{kl}\left(\mathbb{C}_{abkl} - \mathbb{C}_{bakl}\right) = 0$$

Since ε_{kl} is not completely filled with zero:

$$\mathbb{C}_{abkl} = \mathbb{C}_{bakl}$$

b) if strain tensor is symmetric: $\varepsilon_{ab} = \varepsilon_{ba}$

$$\varepsilon_{ab} = \mathbb{C}_{ijab}^{-1}\,\sigma_{ij} \quad , \quad \varepsilon_{ba} = \mathbb{C}_{ijba}^{-1}\,\sigma_{ij}$$

$$\Rightarrow \mathbb{C}_{ijab}^{-1}\,\sigma_{ij} = \mathbb{C}_{ijba}^{-1}\,\sigma_{ij}$$

$$\Rightarrow \mathbb{C}_{ijab}^{-1} = \mathbb{C}_{ijba}^{-1}$$

$$\Rightarrow \mathbb{C}_{ijab} = \mathbb{C}_{ijba}$$

Thus the number of components can be reduced

from $3^4 = 81$,

to $6 \times 6 = 36$.

i	j		
1	1	2	3
2	1	2	3
3	1	2	3

Task 2.2:

$$\mathbb{C}_{ijkl} = \frac{\partial^2 W}{\partial \varepsilon_{ij} \varepsilon_{kl}} \underset{\substack{\uparrow \\ \text{(symmetry of second derivatives)}}}{=} \frac{\partial^2 W}{\partial \varepsilon_{kl} \varepsilon_{ij}} = \mathbb{C}_{klij}, \text{ combined with } \begin{cases} \mathbb{C}_{ijkl} = \mathbb{C}_{jikl} \\ \mathbb{C}_{ijkl} = \mathbb{C}_{ijlk} \end{cases}$$

$$\mathbb{C}_{ijkl} = \begin{bmatrix} C_{1111} & C_{1122} & C_{1133} & C_{1123} & C_{1113} & C_{1112} \\ & C_{2222} & C_{2233} & C_{2223} & C_{2213} & C_{2212} \\ & & C_{3333} & C_{3323} & C_{3313} & C_{3312} \\ & & & C_{2323} & C_{2313} & C_{2312} \\ & \text{Symm.} & & & C_{1313} & C_{1312} \\ & & & & & C_{1212} \end{bmatrix} \text{ is symmetric.}$$

In total 21 material constants.

Task 2.3:

In isotropic case:

$$\mathbb{C}_{ijkl} = \begin{bmatrix} C_{1111} & C_{1122} & C_{1133} & C_{1123} & C_{1113} & C_{1112} \\ & C_{2222} & C_{2233} & C_{2223} & C_{2213} & C_{2212} \\ & & C_{3333} & C_{3323} & C_{3313} & C_{3312} \\ & & & C_{2323} & C_{2313} & C_{2312} \\ & \text{Symm.} & & & C_{1313} & C_{1312} \\ & & & & & C_{1212} \end{bmatrix}$$

$$= \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ & & \lambda+2\mu & 0 & 0 & 0 \\ & & & \mu & 0 & 0 \\ & \text{Symm.} & & & \mu & 0 \\ & & & & & \mu \end{bmatrix}$$

$$\sigma_{ij} = \mathbb{C}_{ijkl}\varepsilon_{kl} = \left[\lambda\delta_{ij}\delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})\right]\varepsilon_{kl}$$

$$= \lambda\delta_{ij}\underbrace{\delta_{kl}\varepsilon_{kl}}_{k=l \Rightarrow \varepsilon_{kk}} + \underbrace{\mu\delta_{ik}\delta_{jl}\varepsilon_{kl}}_{\substack{i=k,\, j=l \\ \Rightarrow \varepsilon_{ij}}} + \underbrace{\mu\delta_{il}\delta_{jk}\varepsilon_{kl}}_{\substack{i=l,\, j=k \\ \Rightarrow \varepsilon_{ji}}}$$

$$= \lambda\delta_{ij}\varepsilon_{kk} + \mu(\varepsilon_{ij} + \varepsilon_{ji})$$

given: $\nu = \frac{\lambda}{2(\lambda+\mu)} = 0 \Rightarrow \quad \lambda = 0$

$$\Rightarrow E = 2\mu$$

$$\Rightarrow \mu = \frac{1}{2}E$$

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl} = \begin{bmatrix} E & 0 & 0 & 0 & 0 & 0 \\ & E & 0 & 0 & 0 & 0 \\ & & E & 0 & 0 & 0 \\ & \text{sym.} & & \frac{1}{2}E & 0 & 0 \\ & & & & \frac{1}{2}E & 0 \\ & & & & & \frac{1}{2}E \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ 2\varepsilon_{23} \\ 2\varepsilon_{13} \\ 2\varepsilon_{12} \end{bmatrix} = \begin{bmatrix} E & 0 & 0 & 0 & 0 & 0 \\ & E & 0 & 0 & 0 & 0 \\ & & E & 0 & 0 & 0 \\ & \text{sym.} & & \frac{1}{2}E & 0 & 0 \\ & & & & \frac{1}{2}E & 0 \\ & & & & & \frac{1}{2}E \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial x} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} E\frac{\partial u}{\partial x} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \Rightarrow \sigma_{11} = E\frac{\partial u}{\partial x}$$

Balance equation: $\rho \ddot{\mathbf{u}} = \operatorname{div}(\boldsymbol{\sigma}) + \rho \mathbf{b}$

$$\rho \ddot{u}_{11} = \frac{\partial \sigma_{11}}{\partial x} + \rho b(x)$$

Since statically loaded, $\ddot{u}_{11} = 0$

$$\Rightarrow \quad 0 = \frac{d E\frac{\partial u}{\partial x}}{dx} + \rho b(x)$$

Since only deformation in x direction:

$$\Rightarrow \quad 0 = \frac{d}{dx}\left(E\frac{du}{dx}\right) + \rho b(x)$$

$$\Rightarrow \quad 0 = \frac{d}{dx}\left(EA\frac{du}{dx}\right) + A\rho b(x)$$

prooved.

Task 3.1:

Derive weak form from 1D strong form

Strong form: $\mathrm{div}(\boldsymbol{\sigma})+\rho\mathbf{b}=0$

weak form: $\int_{\mathcal{B}^e} w\left[\mathrm{div}(\boldsymbol{\sigma})+\rho\mathbf{b}\right]dV$

Since δu is the test function, for 1D-element problem:

$$\int_{\mathcal{B}^e}\delta\mathbf{u}\left[\mathrm{div}(\boldsymbol{\sigma})+\rho\mathbf{b}\right]dV=0$$

$$\int_{\mathcal{B}^e}\delta\mathbf{u}\,\mathrm{div}(\boldsymbol{\sigma})\,dV+\int_{\mathcal{B}^e}\delta\mathbf{u}\,\rho\mathbf{b}\,dV=0$$

(with the product rule $(uv)'=u'v+uv'$:

$$\mathrm{div}(\delta\mathbf{u}\cdot\boldsymbol{\sigma})=\mathrm{div}(\delta\mathbf{u})\cdot\boldsymbol{\sigma}+\delta\mathbf{u}\cdot\mathrm{div}(\boldsymbol{\sigma})$$

$$\Rightarrow\delta\mathbf{u}\cdot\mathrm{div}(\boldsymbol{\sigma})=\mathrm{div}(\delta\mathbf{u}\cdot\boldsymbol{\sigma})-\mathrm{div}(\delta\mathbf{u})\cdot\boldsymbol{\sigma}$$)

$$\int_{\mathcal{B}^e}\mathrm{div}(\delta\mathbf{u}\cdot\boldsymbol{\sigma})\,dV-\int_{\mathcal{B}^e}\mathrm{div}(\delta\mathbf{u})\cdot\boldsymbol{\sigma}\,dV+\int_{\mathcal{B}^e}\delta\mathbf{u}\,\rho\mathbf{b}\,dV=0$$

$$\underbrace{\int_{\mathcal{B}^e}\mathrm{div}(\delta\mathbf{u}\cdot\boldsymbol{\sigma})\,dV}_{①}-\underbrace{\int_{\mathcal{B}^e}\mathrm{grad}(\delta\mathbf{u})\cdot\boldsymbol{\sigma}\,dV}_{②}+\underbrace{\int_{\mathcal{B}^e}\delta\mathbf{u}\,\rho\mathbf{b}\,dV}_{③}=0 \quad (*)$$

(For ①: divergence theorem:

$$\int_{\mathcal{B}^e}\mathrm{div}(\delta\mathbf{u}\boldsymbol{\sigma})\,dV=\int_{\Gamma_{\mathcal{B}}^e}(\delta\mathbf{u}\cdot\boldsymbol{\sigma})\cdot\mathbf{n}\,dA=\int_{\Gamma_t^e}\delta u\cdot\boldsymbol{\sigma}\mathbf{n}\,dA+\underbrace{\int_{\Gamma_u^e}\boldsymbol{\sigma}\,\delta u\,n\,dA}_{=0}$$

$$=\int_{\Gamma_t^e}\bar{\mathbf{t}}\,\delta\mathbf{u}\,dA$$)

The equation (*) becomes:

$$\underbrace{\int_{\mathcal{B}^e}\boldsymbol{\sigma}\,\mathrm{grad}(\delta\mathbf{u})\,dV}_{②}=\underbrace{\int_{\Gamma_t^e}\bar{\mathbf{t}}\,\delta\mathbf{u}\,dA}_{①}+\underbrace{\int_{\mathcal{B}^e}\delta\mathbf{u}\,\rho\mathbf{b}\,dV}_{③}$$

$$\int_{\mathcal{B}^e}(\mathbb{C}:\boldsymbol{\varepsilon}):\mathrm{grad}(\delta\mathbf{u})\,dV=\int_{\mathcal{B}^e}\rho\mathbf{b}\cdot\delta\mathbf{u}\,dV+\int_{\Gamma_t^e}\bar{\mathbf{t}}\cdot\delta\mathbf{u}\,dA$$

$$\int_B (\mathbb{C}:\varepsilon):\text{grad}(\delta\mathbf{u})\,dV = \iiint (\mathbb{C}:\varepsilon):\text{grad}(\delta\mathbf{u})\,dx\,dy\,dz$$

Since no chang in y & z directions, the formula can be written as:

$$\text{left hand side} = \int_0^l (\mathbb{C}:\varepsilon):\text{grad}(\delta\mathbf{u})\,dx$$
$$= \int_0^l \boldsymbol{\sigma}:\text{grad}(\delta\mathbf{u})\,dx$$
$$= \int_0^l E\frac{du}{dx}\frac{d\delta u(x)}{dx}\,dx$$

$$\text{right hand side} = \underbrace{\int_B \rho\mathbf{b}\,\delta\mathbf{u}\,dV}_{①} + \underbrace{\int_{\Gamma_t} \bar{t}\,\delta\mathbf{u}\,dA}_{②}$$

$$① = \int_0^l \rho\mathbf{b}\,\delta\mathbf{u}\,dx$$
$$= \int_0^l \rho\, b(x)\,\delta u(x)\,dx$$

$$② = \bar{t}\int_{\Gamma_t} \delta u\,dA$$
$$= \bar{t}\,\delta u|_0^l$$
$$= \bar{t}\,\delta u(l)$$

$$\Rightarrow \int_0^l E\frac{du}{dx}\frac{d\delta u(x)}{dx}\,dx = \int_0^l \rho\, b(x)\,\delta u(x)\,dx + \bar{t}\,\delta u(l)$$

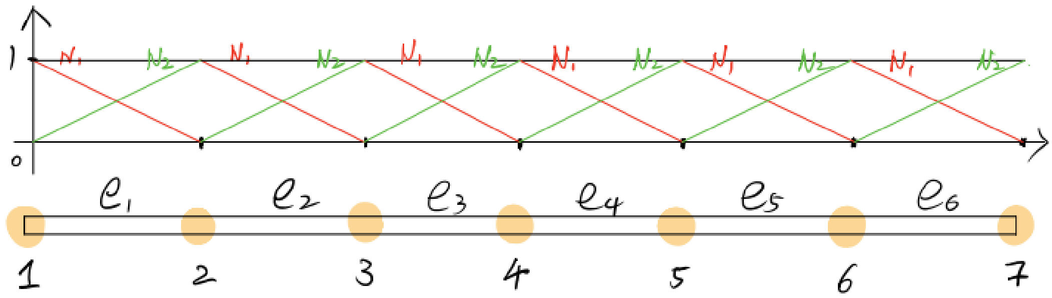
multiply both sides with A:

$$\int_0^l \frac{du(x)}{dx}EA\frac{d\delta u(x)}{dx}\,dx = \int_0^l A\rho\, b(x)\,\delta u(x)\,dx + \bar{t}A\delta u(l)$$

prooved.

Task 4.1:

Handwritten solution:



$$N_1(x) = \frac{x - x_2}{x_1 - x_2} \qquad N_2(x) = \frac{x - x_1}{x_2 - x_1}$$

$$f(x)_{1-2} = \hat{u}_1 \frac{x - x_2}{x_1 - x_2} + \hat{u}_2 \frac{x - x_1}{x_2 - x_1}$$

$$f(x)_{2-3} = \hat{u}_2 \frac{x - x_3}{x_2 - x_3} + \hat{u}_3 \frac{x - x_2}{x_3 - x_2}$$

$$f(x)_{3-4} = \hat{u}_3 \frac{x - x_4}{x_3 - x_4} + \hat{u}_4 \frac{x - x_3}{x_4 - x_3}$$

$$\vdots$$

$$f(x)_{6-7} = \hat{u}_6 \frac{x - x_7}{x_6 - x_7} + \hat{u}_7 \frac{x - x_6}{x_7 - x_6}$$

Code and results:

*[code in exp2.py **class Interpolation()** and test.py **class Test()**: task4_1()]*

3 different functions (binomial, parabola and natural logarithm were chosen. Figure 1 shows the curves of these functions, with nodes and evenly distributed evaluation points.

The evaluation points, which are taken from the linear interpolation, show in the most regions a quite decent approximation of the curves. In Figure 1 (c), there exist a relatively significant deviation between real and interpolated values. Since the interpolation is done only in a linear way, the deviation becomes larger when the real segment is “rounder”. This problem can be overcome by making elements “finer”. Figure 1 (d) shows that the interpolation result becomes better when the curve is divided into smaller segments, thus overcoming the “roughness” caused by the “roundness” of the curves.

Approximated function graphs are shown in Figure 2 for better observation.

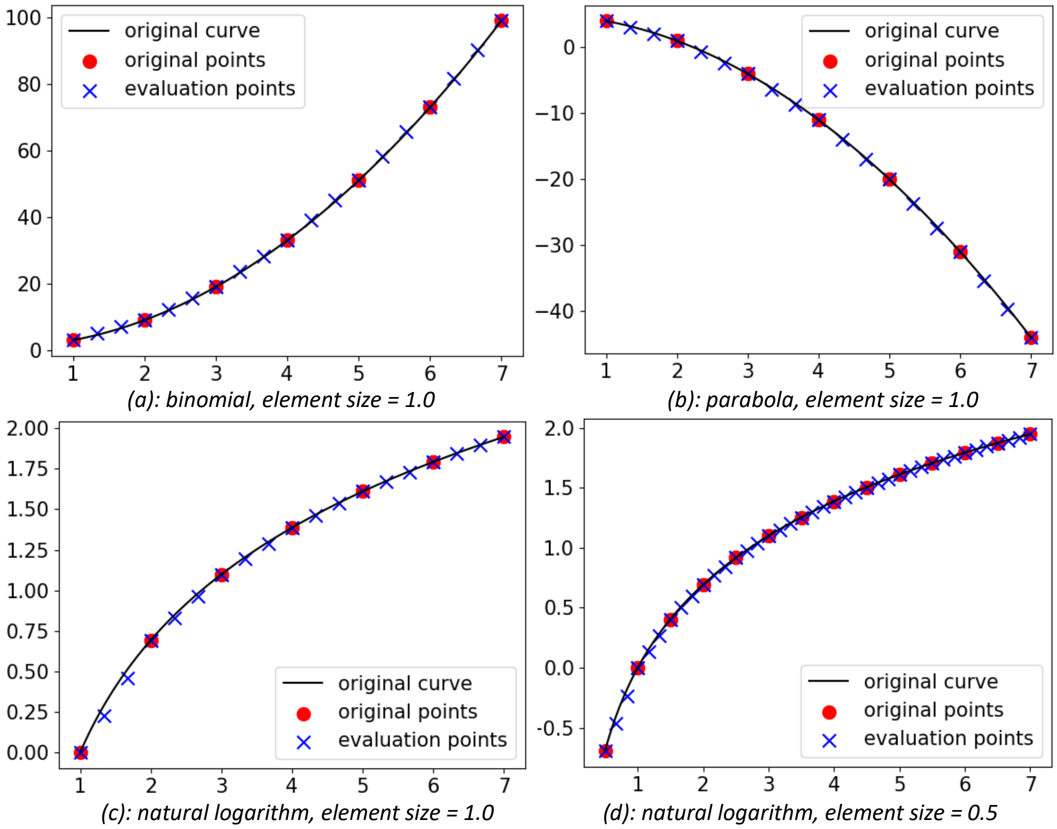


Figure 1: curve, nodes and evaluation points of different functions.

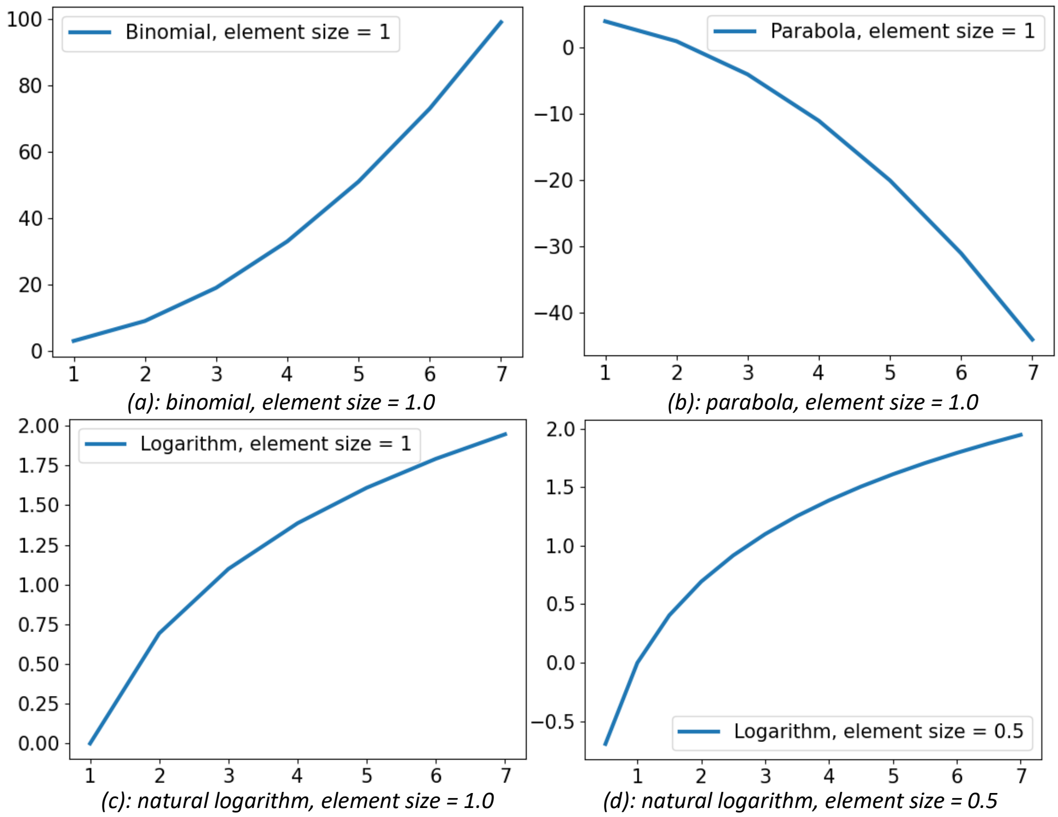


Figure 2: approximated function graphs of different functions and element sizes.

Task 4.2:

Handwritten solution:

$$B_1(x) = \frac{dN_1(x)}{dx} = \frac{1}{x_1-x_2}, \quad B_2(x) = \frac{dN_2(x)}{dx} = \frac{1}{x_2-x_1}$$

$$\frac{d\tilde{u}(x)}{dx} = B_1(x)\hat{u}_1 + B_2(x)\hat{u}_2 = \frac{1}{x_1-x_2}\hat{u}_1 + \frac{1}{x_2-x_1}\hat{u}_2$$

assume $x_2 > x_1$, then: $\frac{d\tilde{u}(x)}{dx} = \frac{1}{x_2-x_1}(\hat{u}_2-\hat{u}_1)$

if element length $=1 \Rightarrow x_2-x_1=1 \Rightarrow \frac{d\tilde{u}(x)}{dx} = \hat{u}_2-\hat{u}_1$,

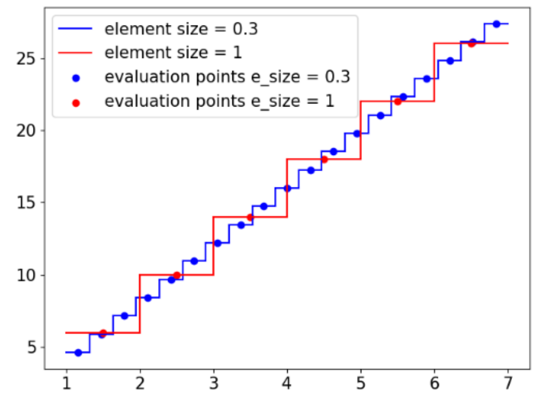
just the difference of displacements

Code and results:

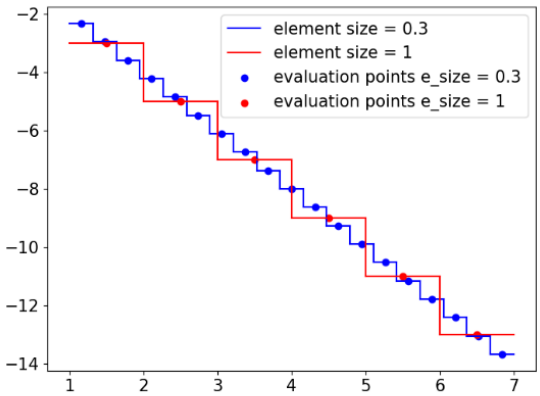
*[code in exp2.py **class Interpolation()** and test.py **class Test()**: task4_2]*

Figure 3 shows how the displacement gradient looks like for the same functions in task 4.1. The gradient is taken over the elements with different sizes. The evaluation points are taken in the middle of each element.

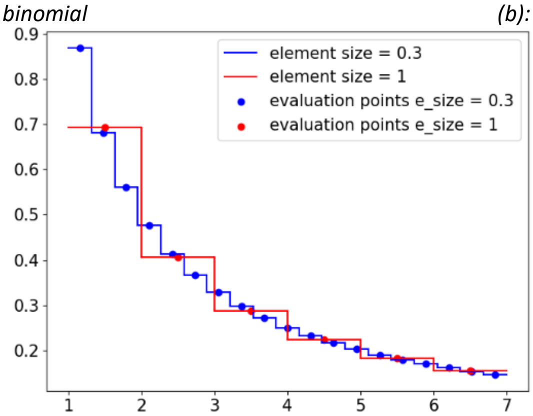
The deviation between approximated and real gradient values can be large at the edge of the "zigzags", especially when the gradient is changing in a non-linear way, as shown in Figure 3 (c). By decreasing the element size, this problem can be overcome, which is similar to the observation of approximating function values in task 4.1.



(a): binomial



(b): parabola



(c): natural logarithm

Figure 3: gradient and evaluation points of different functions.

Task 5.1:

*[code in exp2.py **class Integration()**: quadrature()]*

```
class Integration():
    def quadrature(self, x1, x2, n, d):
        '''
        Do quadrature integration.
        :param x1:   float,  starting point of integral
        :param x2:   float,  ending point of integral
        :param n:    int,    degree of Newton-Cotes formula
        :param d:    int,    degree of polynomial
        :return quad_integration:   float, result of quadrature integration
        '''
        quad_integration = 0

        para_zeta, para_lambda = self.NewtonCotes(x1, x2, n)    # get lambda & zeta
        f_zeta = func().f_polynomial_order(para_zeta, order=d)  # get f(zeta)

        for i in range(n):
            quad_integration += para_lambda[i] * f_zeta[i]      # do summation
        quad_integration = quad_integration * (x2 - x1)

        print(f'integral of n={n}, d={d}:  ', quad_integration)
        return quad_integration
```

Figure 4: Code of function quadrature().

Task 5.2:

*[code in exp2.py **class Integration()**: quadrature(), NewtonCotes(), intg_polynomial_order () and test.py **class Test()**: task5_2]*

Part of the testing results are shown in Figure 4. n indicates the degree of Newton-Cotes formula and d is the degree of polynomial.

The real areas are calculated analytically. Closed Newton-Cotes provides a precise estimation of the area, if n (degree of Newton-Cotes) is larger than or equal to d (order of polynomials).

```
=====real area of polynomial order d=1: 10.0 =====
integral of n=2, d=1:   10.0
integral of n=3, d=1:   10.0
integral of n=4, d=1:   10.0
=====real area of polynomial order d=2: 19.333333333333332 =====
integral of n=2, d=2:   22.0
integral of n=3, d=2:   19.333333333333332
integral of n=4, d=2:   19.333333333333336
=====real area of polynomial order d=3: 42.0 =====
integral of n=2, d=3:   58.0
integral of n=3, d=3:   42.0
integral of n=4, d=3:   42.0
=====real area of polynomial order d=4: 98.80000000000001 =====
integral of n=2, d=4:   166.0
integral of n=3, d=4:   99.33333333333333
integral of n=4, d=4:   99.03703703703705
=====real area of polynomial order d=5: 244.66666666666666 =====
integral of n=2, d=5:   490.0
integral of n=3, d=5:   249.99999999999997
integral of n=4, d=5:   247.03703703703707
```

Figure 5: Comparison between "real" and estimated area values.

Task 5.3:

Handwritten solution:

- $N_1(x) = \frac{x - x_2}{x_1 - x_2} \qquad N_2(x) = \frac{x - x_1}{x_2 - x_1}$

$$\Rightarrow B(x) = \frac{d}{dx} N(x) = \begin{bmatrix} \frac{1}{x_1 - x_2} & \frac{1}{x_2 - x_1} \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{1}{l} & \frac{1}{l} \end{bmatrix}$$

$$K^e = \int_e \mathbf{B}^T EA \mathbf{B} \, dx$$

$$= EA \int_e \mathbf{B}^T \mathbf{B} \, dx$$

$$= EA \int_0^l \begin{bmatrix} -\frac{1}{l} \\ \frac{1}{l} \end{bmatrix} \begin{bmatrix} -\frac{1}{l} & \frac{1}{l} \end{bmatrix} dx$$

$$= EA \int_0^l \begin{bmatrix} \frac{1}{l^2} & -\frac{1}{l^2} \\ -\frac{1}{l^2} & \frac{1}{l^2} \end{bmatrix} dx$$

$$= EA \begin{bmatrix} \frac{1}{l^2} & -\frac{1}{l^2} \\ -\frac{1}{l^2} & \frac{1}{l^2} \end{bmatrix} l$$

$$= \frac{EA}{l} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

- Given $E = 210000\,N/mm^2$, $A = 25\,mm^2$, $l = 50\,mm$:

$$K^e = 105000 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 105000 & -105000 \\ -105000 & 105000 \end{bmatrix}$$

- $K^e \cdot \hat{u} = f^e$

$$\Leftrightarrow \begin{bmatrix} 105000 & -105000 \\ -105000 & 105000 \end{bmatrix} \cdot \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$$

$$\Rightarrow -105000\,\hat{u}_1 + 105000\,\hat{u}_2 = 5$$

$$\Rightarrow 105000\,\hat{u}_2 = 5$$

$$\Rightarrow \hat{u}_2 = \frac{1}{21000}\,mm \approx \underline{\underline{4.76 \times 10^{-5}\,mm}}$$

Solution on Python:

*[code in exp2.py **class LinearSys()**: assembleLinear(), solveLinear () and test.py **class Test()**: task5_3()]*

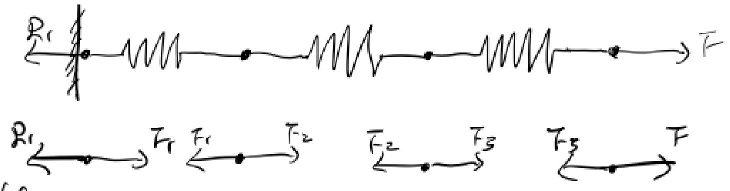
The result is identical to the handwritten one.

```
def task5_3(self):
    print('###### Testing Task 5.3 ######')
    ls = LinearSys()
    E = 210000
    A = 25
    Le = 50
    Ke = ls.element_stiffness_matrix(E, A, Le=Le)
    print("Element stiffness matrix:\n", Ke)

    K = ls.assembleLinear(Ke=Ke, N=1, u1 = 0.0, uend = 50.0, f_b = 0, f_s = 5, to_print=False)
    print("Global stiffness matrix:\n", K)

    d = ls.solveLinear(Ke=Ke, N=1, u1 = 0.0, uend = None, F = 5)
    print("Solved displacement vector:\n", d)
```

```
###### Testing Task 5.3 ######
B:
 [[-0.02  0.02]]
B.T:
 [[-0.02]
 [ 0.02]]
==============================
Testing: N = 1 , u1 = 0.0 , uend = None , F = 5 , Ke =
 [[ 105000. -105000.]
 [-105000.  105000.]]
Assembling global stiffness matrix with Ke=
 [[ 105000. -105000.]
 [-105000.  105000.]]
=== d ===
 [[0.00000000e+00]
 [4.76190476e-05]]
=== K ===
 [[ 105000. -105000.]
 [-105000.  105000.]]
=== d_dirichlet ===
 [[0.]
 [0.]]
=== f ===
 [[-5.]
 [ 5.]]
=== fr ===
 [[5.]]
=== Kr ===
 [[105000.]]
=== dr ===
 [[4.76190476e-05]]
Solved displacement vector:
 [[0.00000000e+00]
 [4.76190476e-05]]
```

Figure 6: Solver for one element problem.

Task 6.1:



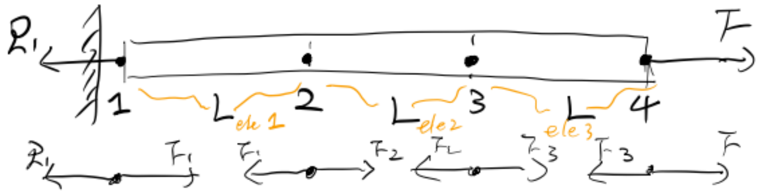
$$\left\{\begin{array}{l}\begin{cases} F_1 = (u_2-u_1)c \\ -R_1+F_1 = 0 \end{cases} \\ \begin{cases} F_2 = (u_3-u_2)c \\ -F_1+F_2 = 0 \end{cases} \\ \begin{cases} F_3 = (u_4-u_3)c \\ -F_3+F = 0 \end{cases}\end{array}\right. \Rightarrow \begin{cases} c(u_1-u_2) = -R_1 \\ c(u_3-2u_2+u_1) = 0 \\ c(u_4-2u_3+u_2) = 0 \\ c(u_3-u_4) = -F \end{cases} \Rightarrow c\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} -R_1 \\ 0 \\ 0 \\ -F \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ 0 \\ -F \end{bmatrix} - \begin{bmatrix} R_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

proved.

Task 6.2:

*[code in exp2.py **LinearSys().assembleLinear()**]*



c replaced by $\frac{AE}{L}$

Since $\sigma_i = E\xi \Rightarrow \frac{F_i}{A} = E\frac{\Delta u}{L} \Rightarrow \frac{F_i}{A} = \frac{E}{L}(u_{i+1} - u_i)$

$\Rightarrow u_{i+1} - u_i = \frac{F_i L}{AE}$

$\Rightarrow \underline{F_i = \frac{AE}{L}(u_{i+1} - u_i)}$

(direction left → right)

$N_1: \frac{AE}{L}(u_1 - u_2) = 0 - R_1$

$N_2: F_1 + F_2 = 0$

$\Rightarrow \frac{AE}{L}(u_2 - u_1 + u_2 - u_3) = 0$

$N_3: F_2 + F_3 = 0$

$\Rightarrow \frac{AE}{L}(u_3 - u_2 + u_4 - u_3) = 0$

$N_4: F_3 = -F$

$\Rightarrow \frac{AE}{L}(u_4 - u_3) = -F$

Element stiffness matrix:

$$\frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Copied block-wise into the global stiffness matrix in diagonal

$$\frac{AE}{L}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \frac{AE}{L}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1+1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \frac{AE}{L}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1+1 & -1 & 0 \\ 0 & -1 & 1+1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

adding element 1 adding element 2 adding element 3

Similarly, assemble displacement vector element by element

$$\begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \hat{u}_3 \\ \hat{u}_4 \end{bmatrix}$$

Last step: apply BCs

$$\Rightarrow \frac{AE}{L}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -F \end{bmatrix} - \begin{bmatrix} R_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Task 6.3:

*[code in test.py **Test().task6_2_3()**]*

6.3
$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -F \end{bmatrix} - \begin{bmatrix} R_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

→ properties of K:

Sparsely occupied, linear, 3-band, symmetric on diagonal

on diagonal (except first & last = 1) = 2
others = −1

→ Because BC is not yet given (force, Dirichlet BC)

F: External force applied at the last node.

R: Reaction force at the first node. In a stationary system, it has same value with F, while pointing to the opposite direction.

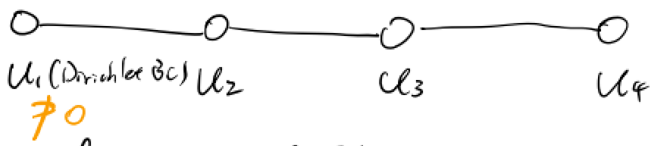
In reality, according to Hooke's Law:

$F = k \cdot x$

The elongation/compression x can't be solved if F is unknown, since the number of unknowns is larger than the number of equations.

Similarly, linear equations for first & last elements can't be solved neither.

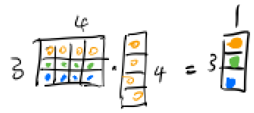
Task 7.1:



According to eq. (57)

$$\underbrace{c\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} 0 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}}_{\check{d}} = \underbrace{\begin{bmatrix} -R \\ 0 \\ 0 \\ -F \end{bmatrix}}_{f-R}$$

$$d_r = P \cdot \begin{bmatrix} 0 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \longrightarrow$$



$$\underbrace{\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 0 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}}_{\check{d}} = \begin{bmatrix} bu_2 + cu_3 + du_4 \\ fu_2 + gu_3 + hu_4 \\ ju_2 + ku_3 + lu_4 \end{bmatrix} = \underbrace{\begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix}}_{d_r}$$

$\Rightarrow b = 1$, $g = 1$, $l = 1$

a, e, i can be arbitrary real number

the rest $= 0$

P can be e.g.:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

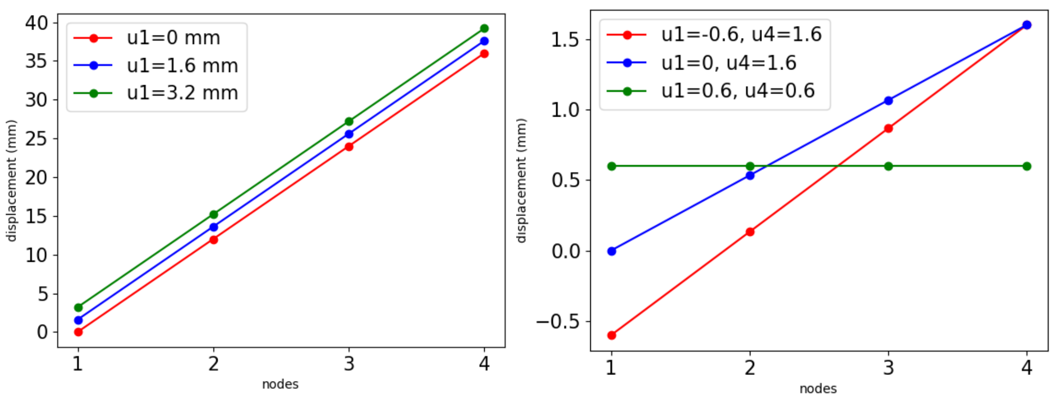
Task 7.2:

*[code in exp2.py **class solveLinear ()**: oneElement() and test.py **class Test()**: task7_2()]*

The permutation matrix method was applied to solve the system of linear equations. Other conditions for each system are same: F = 12N, c = 1. Figure 7 shows how the displacement results look like under different Dirichlet boundary conditions. Since we have a stable linear homogeneous system, the displacement value is related with the location of this node (displacement gradient is constant). This results in parallel straight lines, as shown in Figure 7 (a).

When Dirichlet BC is applied only on one side, the result is always linear and the change on displacement is identical to the change of Dirichlet BC. Table 1 provides a convincing proof on it.

The result by Dirichlet BC on both sides also appears to be straight lines. The slope only depends on the boundary conditions, as shown in Figure 7 (b). The displacement on all nodes is same if a boundary condition of u1 = u4 is given.



(a): spring system with Dirichlet BC on the first node   *(b): spring system with Dirichlet BC at both sides*

Figure 7: displacement results under different Dirichlet boundary conditions.

Table 1: displacement results in digits for Figure 6 (a).

Dirichlet boundary condition	Displacement result (mm) u1	u2	u3	u4	change on displacement (mm)
u1 = 0 mm	0	12	24	36	0
u1 = 1.6 mm	1.6	13.6	25.6	37.6	1.6
u1 = 3.2 mm	3.2	15.2	27.2	39.2	3.2

Task 8.1:

*[code in exp2.py **class solveLinear ()**: postProcess () and test.py **class Test()**: task8_1()]*

The permutation matrix method was applied to solve the system of linear equations. In our one-dimension linear system, the stress field is a 1-D vector. In Figure 8, some representative results of stress field under different boundary conditions are shown. Condition: sectional area A =25 mm^2, Young's modulus E = 210000 N/mm^2, element length L=50mm.

They look analogous to the displacement gradient vector — all of them are constant over the entire system. The reason is: The gradient vector $\mathbf{B}(\mathbf{x})$ has same values in one system. The displacement vector represents local displacement in one element. Since the adjacent nodes are equidistant, the stress values of all elements are same. Unlike global displacement, the Dirichlet boundary condition might not affect the result. When only one Dirichlet BC is applied on a side, with no constraint on another side, the result of stress field will be same. The value of this Dirichlet BC doesn't have influence, see Figure 8 blue and orange lines.

When Dirichlet BC is applied on both sides, the result depends on the difference between these two values. In Figure 8 (green line), the stress is zero everywhere since the same displacement values on node 1 and node 4 results in no difference in displacement between nodes. By creating difference between boundary values, the "plain" can move upwards or downwards, which can be observed in Figure 8 (red line).

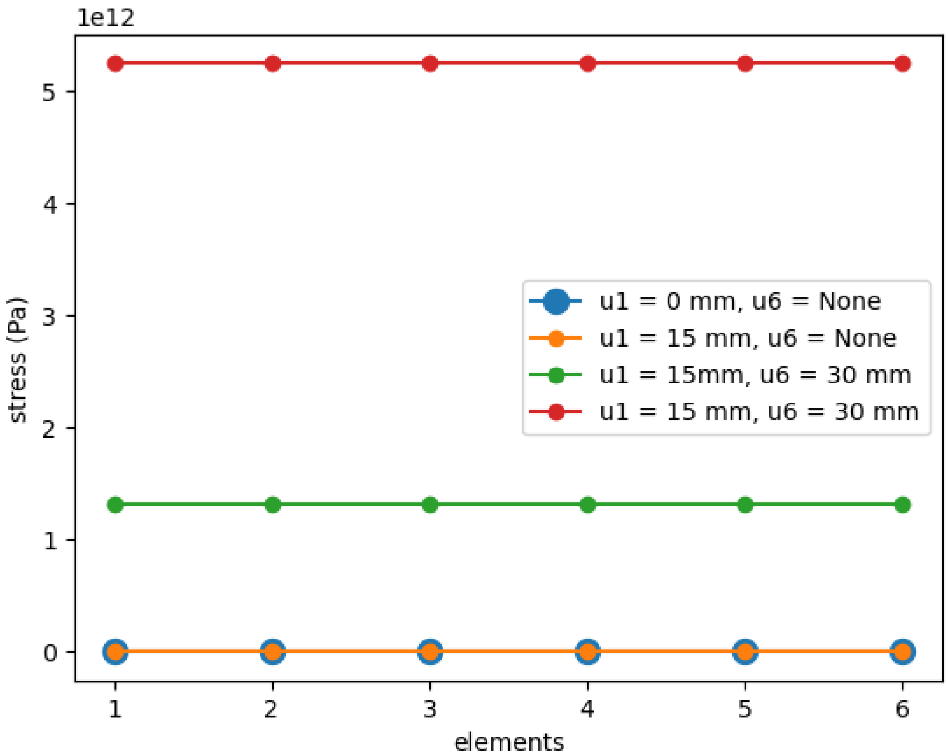


Figure 8: stress field results under different Dirichlet boundary conditions.