

VEM alapjai 1. házi feladat

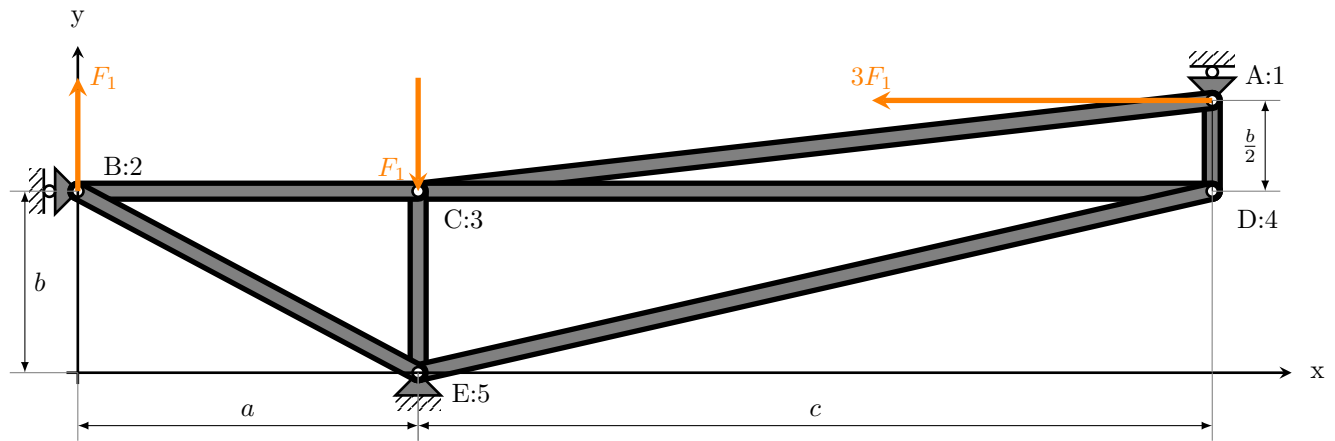
Tóth Barnabás
PMTHJV

2024. március 24.

Tartalomjegyzék

1. A szerkezet	2
2. VE-modell	2
3. Számolás	3
4. Deformált alak	4
5. Programkód	5

1. A szerkezet



1. ábra. A szerkezet méretarányos ábrája

$$\begin{array}{llll}
 a = 3 \text{ m} & c = 7 \text{ m} & F_1 = 90000 \text{ kN} & \nu = 0.3 \\
 b = 1.6 \text{ m} & d = 0.045 \text{ m} & E = 150 \text{ GPa} &
 \end{array}$$

A rudak cső keresztmetszetűek, ez alapján a keresztmetszet területe:

$$A = \frac{\left((1.3 \cdot d)^2 - d^2\right) \cdot \pi}{4}$$

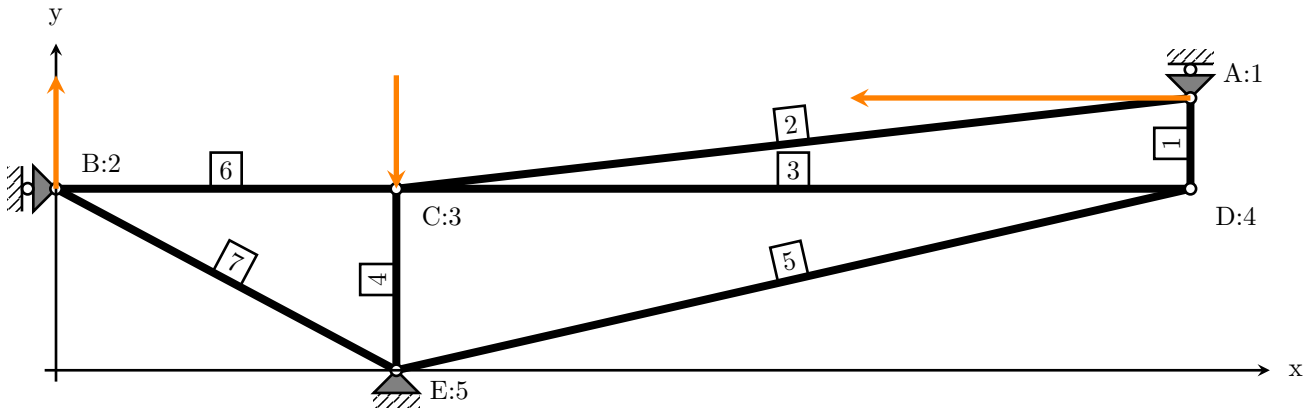
2. VE-modell

Két csomópontos rúdelemeket használva az egy elemhez tartozó elmozdulás- és erővektorok:

$$\mathbf{U}^{(e)} = \begin{bmatrix} u_1^{(e)} \\ v_1^{(e)} \\ u_2^{(e)} \\ v_2^{(e)} \end{bmatrix} \quad \mathbf{F}^{(e)} = \begin{bmatrix} F_{1,x}^{(e)} \\ F_{1,y}^{(e)} \\ F_{2,x}^{(e)} \\ F_{2,y}^{(e)} \end{bmatrix}$$

Ezekből előállítható a globális elmozdulás-és erővektor, a megfelelő kényszerekkel ellátva:

$$\mathbf{U} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \\ u_5 \\ v_5 \end{bmatrix} = \begin{bmatrix} u_1 \\ 0 \\ 0 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} F_{1,x} \\ F_{1,y} \\ F_{2,x} \\ F_{2,y} \\ F_{3,x} \\ F_{3,y} \\ F_{4,x} \\ F_{4,y} \\ F_{5,x} \\ F_{5,y} \end{bmatrix} = \begin{bmatrix} -3 \cdot F \\ F_{1,y} \\ F_{2,x} \\ F \\ 0 \\ -F \\ 0 \\ 0 \\ F_{5,x} \\ F_{5,y} \end{bmatrix}$$



2. ábra. A szerkezet végelem-modellje

3. Számolás

Merevségi mátrix

Az elkészített program végigmegy minden rúdelemen, elkészíti a lokális merevségi mátrixot az elem irányszögéből kiszámolt szögfüggvényekkel:

$$c := \cos(\alpha^{(e)})$$

$$s := \sin(\alpha^{(e)})$$

$$K^{(e)} = \frac{A^{(e)} \cdot E^{(e)}}{L^{(e)}} \begin{bmatrix} \overbrace{\begin{matrix} c^2 & c \cdot s \\ c \cdot s & s^2 \end{matrix}}^{n_1^{(e)}} & \overbrace{\begin{matrix} -c^2 & -c \cdot s \\ -c \cdot s & -s^2 \end{matrix}}^{n_2^{(e)}} \\ \overbrace{\begin{matrix} -c^2 & -c \cdot s \\ -c \cdot s & -s^2 \end{matrix}}^{n_1^{(e)}} & \overbrace{\begin{matrix} c^2 & c \cdot s \\ c \cdot s & s^2 \end{matrix}}^{n_2^{(e)}} \end{bmatrix} \begin{matrix} n_1^{(e)} \\ n_2^{(e)} \end{matrix}$$

Az elemi merevségi mátrix négy "sarkát", azaz a a csomópont-párosok által alkotott 2×2 -es almatrrixokat a \mathbf{K} megfelelő csomóponti sor-oszlop helyekhez hozzáadja, ezzel elkészül a globális merevségi mátrix:

$$\mathbf{K} = \begin{bmatrix} 2.3062e+7 & 2.6357e+6 & 0 & 0 & -2.3062e+7 & -2.6357e+6 & -7.7148e-25 & 1.2599e-8 & 0 & 0 \\ 2.6357e+6 & 2.0606e+8 & 0 & 0 & -2.6357e+6 & -3.0122e+5 & 1.2599e-8 & -2.0576e+8 & 0 & 0 \\ 0 & 0 & 9.2563e+7 & -2.0103e+7 & -5.4870e+7 & 0 & 0 & 0 & -3.7693e+7 & 2.0103e+7 \\ 0 & 0 & -2.0103e+7 & 1.0722e+7 & 0 & 0 & 0 & 0 & 2.0103e+7 & -1.0722e+7 \\ -2.3062e+7 & -2.6357e+6 & -5.4870e+7 & 0 & 1.0145e+8 & 2.6357e+6 & -2.3516e+7 & 0 & -3.8574e-25 & 6.2996e-9 \\ -2.6357e+6 & -3.0122e+5 & 0 & 0 & 2.6357e+6 & 1.0318e+8 & 0 & 0 & 6.2996e-9 & -1.0288e+8 \\ -7.7148e-25 & 1.2599e-8 & 0 & 0 & -2.3516e+7 & 0 & 4.5302e+7 & 4.9797e+6 & -2.1786e+7 & -4.9797e+6 \\ 1.2599e-8 & -2.0576e+8 & 0 & 0 & 0 & 0 & 4.9797e+6 & 2.0690e+8 & -4.9797e+6 & -1.1382e+6 \\ 0 & 0 & -3.7693e+7 & 2.0103e+7 & -3.8574e-25 & 6.2996e-9 & -2.1786e+7 & -4.9797e+6 & 5.9479e+7 & -1.5123e+7 \\ 0 & 0 & 2.0103e+7 & -1.0722e+7 & 6.2996e-9 & -1.0288e+8 & -4.9797e+6 & -1.1382e+6 & -1.5123e+7 & 1.1474e+8 \end{bmatrix} \frac{\text{N}}{\text{m}}$$

Kondenzáció

Az elmozdulásvektor nemzérus elemeit tartalmazó egyenletek alapján ezeket az elemeket kiszámíthatjuk, ehhez a zérus elemekre vonatkozó részeket törölni kell. A $\mathbf{K} \cdot \mathbf{U} = \mathbf{F}$ egyenlet(rendszer) $u_i, v_i = 0$ sorait illetve az ezeknek megfelelő oszlopait a \mathbf{K} mátrixnak töröljük, így a $\tilde{\mathbf{K}} \cdot \tilde{\mathbf{U}} = \tilde{\mathbf{F}}$ egyenletet kapjuk:

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ 0 \\ 0 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -3 \cdot F \\ F_{1,y} \\ F_{2,x} \\ F \\ 0 \\ -F \\ 0 \\ 0 \\ F_{5,x} \\ F_{5,y} \end{bmatrix}$$

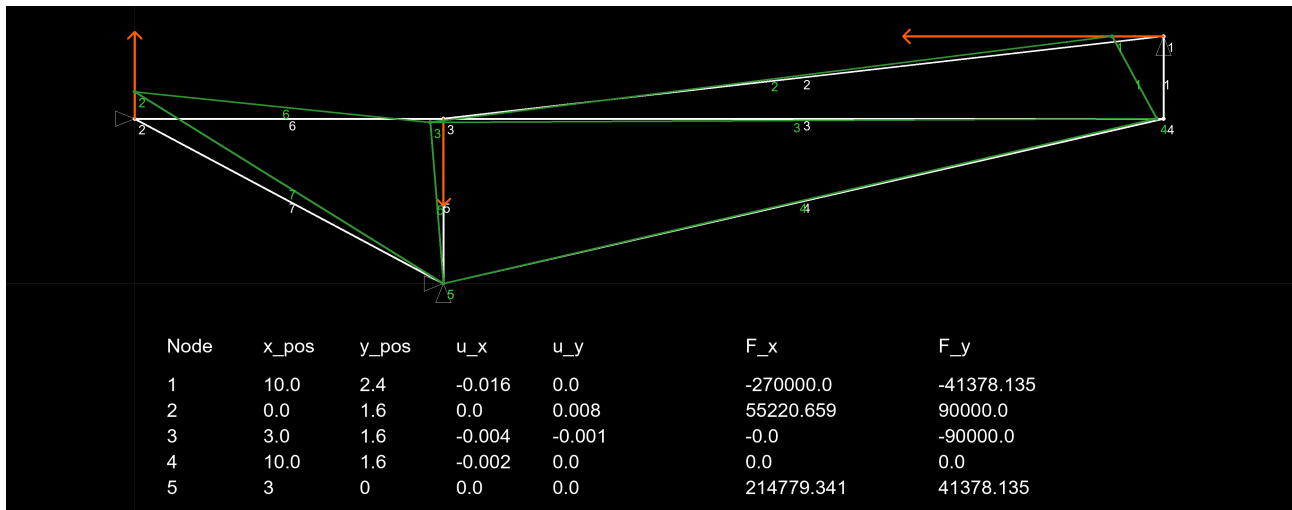
Mivel egy csomópont egyik irányú komponensében vagy a terhelő erőt ismerjük, vagy azt, hogy nincs elmozdulás (így reakcióerőt várunk) a nemzérus elmozdulásokhoz csak ismert erők tartoznak, így a $\tilde{\mathbf{U}} = \tilde{\mathbf{K}}^{-1} \cdot \tilde{\mathbf{F}}$ egyenlettel meghatározhatjuk a kondenzált elmozdulásvektort. A kiszámolt elmozduláskomponenseket a megfelelő helyekre visszaírva megkapjuk a globális elmozdulásvektort. Ezzel (jobbról) megszorozva a merevségi mátrixot megkapjuk a szerkezetre ható csomóponti erők vektorát.

$$\mathbf{U} = \begin{bmatrix} -1.5923e-2 \\ 0 \\ 0 \\ 8.3943e-3 \\ -4.0819e-3 \\ -1.1747e-3 \\ -2.1245e-3 \\ 5.1132e-5 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{m} = \begin{bmatrix} -15.92 \\ 0 \\ 0 \\ 8.39 \\ -4.08 \\ -1.17 \\ -2.12 \\ 0.05 \\ 0 \\ 0 \end{bmatrix} \quad \text{mm}$$

$$\mathbf{F} = \begin{bmatrix} -2.7e+5 \\ -4.1378e+4 \\ 5.5221e+4 \\ 9e+4 \\ 0 \\ -9e+4 \\ 0 \\ 0 \\ 2.1478e+5 \\ 4.1378e+4 \end{bmatrix} \quad \mathbf{N} = \begin{bmatrix} -270 \\ -41.38 \\ 55.22 \\ 90 \\ 0 \\ -90 \\ 0 \\ 0 \\ 214.8 \\ 41.38 \end{bmatrix} \quad \text{kN}$$

4. Deformált alak

A megírt program segítségével kirajzoltatom a deformált alakot, az eredeti alak mellett, illetve a csomóponti elmozdulás-és erőkomponenseket (SI mértékegységekben megadva):



3. ábra. A számítás eredményei

Megjegyzés: Az Ansys számolásában a csomópontok számozása eltér a programkódos számozástól, ez az eredményeket tartalmazó excelben javításra került, a képernyőképen viszont továbbra is a szoftver által generált sorrend látható.

5. Programkód

```

import numpy as np
from PIL import Image, ImageDraw, ImageFont
from tabulate import tabulate
import cv2

target_loc = r"C://saját/docs/python/VEM_hf1.png"
target_loc = r"C://Users/ZenBook/OneDrive - Budapesti Műszaki és Gazdaságtudományi
↳ Egyetem/6_felev/VEM/HF1/hf1.png"
class Node:
    def __init__(self, node_num, loc_x, loc_y, ux = None, uy = None, fx = None, fy = None):
        self.node_num = node_num
        self.pos = np.array([loc_x, loc_y])
        self.displacement = np.array([ux, uy])
        self.force = np.array([fx, fy])
        self.elements = np.array([])

    def add_element(self, elem_num):
        self.elements = np.append(self.elements, elem_num)

    def __repr__(self):
        return f"{self.node_num}, x: {self.pos[0]}, y: {self.pos[1]}\n"

class Element:
    def __init__(self, elem_num, node_1, node_2, E=None, A=None):
        self.elem_num = elem_num
        self.nodes = np.array([node_1, node_2])
        self.E = E
        self.A = A

    def set_length(self, L):
        self.L = L

    def set_angle(self, a):
        self.angle = a

    def __repr__(self):
        return f"{self.elem_num}, n1: {self.nodes[0]}, n2: {self.nodes[1]}\n"

class Model:
    def __init__(self):
        self.elements = np.array([])
        self.nodes = np.array([])

    def addNode(self, node_num, loc_x, loc_y, ux = None, uy = None, fx = None, fy = None):
        self.nodes = np.append(self.nodes, Node(node_num, loc_x, loc_y, ux, uy, fx, fy))

    def addElement(self, elem_num, node_1, node_2, E=None, A=None):
        self.elements = np.append(self.elements, Element(elem_num, node_1, node_2, E, A))
        for i in range(len(self.nodes)):
            if self.nodes[i].node_num == node_1 or self.nodes[i].node_num == node_2:
                self.nodes[i].elements = np.append(self.nodes[i].elements, elem_num)

    def calc_elems(self):
        for elem in self.elements:
            n1 = self.find_node(elem.nodes[0])
            n2 = self.find_node(elem.nodes[1])
            elem.set_angle(np.arctan2((self.nodes[n2].pos[1] - self.nodes[n1].pos[1]),
↳ (self.nodes[n2].pos[0] - self.nodes[n1].pos[0])))

```

```

        elem.set_length(np.sqrt((self.nodes[n2].pos[1] - self.nodes[n1].pos[1])**2 +
        ↪ (self.nodes[n2].pos[0] - self.nodes[n1].pos[0])**2))

def find_node(self, n):
    for i in range(len(self.nodes)):
        if self.nodes[i].node_num == n:
            return i
    return None

def draw(self, deform=False):
    siz=4000
    coordmult=siz/2000
    img = Image.new("RGB", (siz, siz))
    def_scale = 10000
    F_scale = 0.003
    draw = ImageDraw.Draw(img)
    fontsize = coordmult*20
    font_num = ImageFont.truetype("arial.ttf", fontsize)
    font_table = ImageFont.truetype("arial.ttf", coordmult*30)
    x_max, y_max = 0, 0
    x_min, y_min = 1000, 1000
    for n in self.nodes:
        if(n.pos[0] > x_max): x_max = n.pos[0]
        if(n.pos[1] > y_max): y_max = n.pos[1]
        if(n.pos[1] < y_min): y_min = n.pos[1]
        if(n.pos[0] < x_min): x_min = n.pos[0]

    dx = (x_max - x_min)
    dy = (y_max - y_min)

    scale = coordmult*1600 / dx if dx > dy else coordmult*1600 / dy

    #xshift = coordmult*1000 - (x_max + x_min) / 2 * scale
    #yshift = coordmult*1000 + (y_max + y_min) / 2 * scale

    maxY = img.height
    minY = 0

    if deform:
        for node in self.nodes:

            ux = def_scale * self.U_result[2 * (node.node_num - 1)]
            uy = def_scale * self.U_result[2 * (node.node_num - 1) + 1]

            nx = scale * node.pos[0] + ux
            ny = - scale * node.pos[1] - uy

            if ny <= maxY:
                maxY = ny
            if ny >= minY:
                minY = ny

            F = node.force

            if F[0] == None:
                F[0] = 0
            if F[1] == None:
                F[1] = 0

```

```

        if np.linalg.norm(F) != 0:
            F_draw = F * F_scale
            F_end = (int(nx + F_draw[0]), int(ny - F_draw[1]))

        if F_end[1] <= maxY:
            maxY = F_end[1]
        if F_end[1] >= minY:
            minY = F_end[1]

yshift = img.height - (len(self.nodes) + 4) * coordmult*40 - minY
xshift = coordmult*1000 - (x_max + x_min) / 2 * scale
#yshift = coordmult*1000 + (y_max + y_min) / 2 * scale
maxY = img.height
minY = 0

draw.line(xy=(xshift, 0, xshift, coordmult*2000), fill=(50,50,50))
draw.line(xy=(0, yshift, coordmult*2000, yshift), fill=(50,50,50))

for i in range(len(self.elements)):
    elem=self.elements[i]
    n1=self.nodes[self.find_node(elem.nodes[0])]
    n2=self.nodes[self.find_node(elem.nodes[1])]
    n1x = xshift + scale*n1.pos[0]
    n1y = -(-yshift + scale*n1.pos[1])
    n2x = xshift + scale*n2.pos[0]
    n2y = -(-yshift + scale*n2.pos[1])
    draw.line(xy = (n1x, n1y, n2x, n2y), width=int(coordmult*3))
    draw.text(((n1x+n2x)/2, (n1y+n2y)/2),f"{elem.elem_num}",font=font_num)

for node in self.nodes:
    size=coordmult*6
    nx = xshift + scale * node.pos[0]
    ny = yshift - scale * node.pos[1]

    if node.displacement[1] == 0:
        draw.polygon([(nx,ny), (nx + 2*size, ny + 5*size), (nx - 2*size, ny +
        ↪ 5*size)], outline = "white")
    if node.displacement[0] == 0:
        draw.polygon([(nx,ny), (nx - 5*size, ny + 2*size), (nx - 5*size, ny -
        ↪ 2*size)], outline = "white")

    draw.ellipse([(nx-size/2,ny-size/2,nx+size//2,ny+size//2)], fill="white")
    draw.text((nx+size,ny+size),f"{node.node_num}",font=font_num)

F = node.force

if F[0] == None:
    F[0] = 0
if F[1] == None:
    F[1] = 0

if np.linalg.norm(F) != 0:
    F_draw = F * F_scale
    F_end = (int(nx + F_draw[0]), int(ny - F_draw[1]))

    img_array = np.array(img)
    img_array = cv2.arrowedLine(img_array, (int(nx), int(ny)), F_end, (255, 94,
    ↪ 5), int(coordmult*3), tipLength=30/np.linalg.norm(F_draw))

```

```

    img = Image.fromarray(img_array)
    draw = ImageDraw.Draw(img)
    if F_end[1] <= maxY:
        maxY = F_end[1]
    if F_end[1] >= minY:
        minY = F_end[1]

if deform:
    num_nodes = len(self.nodes)

    draw.text((coordmult*250, img.height - (num_nodes + 2) * coordmult*40),
        ↪ "Node",font=font_table, fill="white")
    draw.text((coordmult*400, img.height - (num_nodes + 2) * coordmult*40),
        ↪ "x_pos",font=font_table, fill="white")
    draw.text((coordmult*550, img.height - (num_nodes + 2) * coordmult*40),
        ↪ "y_pos",font=font_table, fill="white")
    draw.text((coordmult*700, img.height - (num_nodes + 2) * coordmult*40),
        ↪ "u_x",font=font_table, fill="white")
    draw.text((coordmult*850, img.height - (num_nodes + 2) * coordmult*40),
        ↪ "u_y",font=font_table, fill="white")
    draw.text((coordmult*1150, img.height - (num_nodes + 2) * coordmult*40),
        ↪ "F_x",font=font_table, fill="white")
    draw.text((coordmult*1450, img.height - (num_nodes + 2) * coordmult*40),
        ↪ "F_y",font=font_table, fill="white")

for i in range(len(self.elements)):
    elem=self.elements[i]
    n1=self.nodes[self.find_node(elem.nodes[0])]
    n2=self.nodes[self.find_node(elem.nodes[1])]
    n1_ux = def_scale * self.U_result[2 * (elem.nodes[0] - 1)]
    n1_uy = def_scale * self.U_result[2 * (elem.nodes[0] - 1) + 1]
    n2_ux = def_scale * self.U_result[2 * (elem.nodes[1] - 1)]
    n2_uy = def_scale * self.U_result[2 * (elem.nodes[1] - 1) + 1]

    n1x = xshift + scale*n1.pos[0] + n1_ux
    n1y = -(-yshift + scale*n1.pos[1] + n1_uy)
    n2x = xshift + scale*n2.pos[0] + n2_ux
    n2y = -(-yshift + scale*n2.pos[1] + n2_uy)
    draw.line(xy = (n1x, n1y, n2x, n2y), width=int(coordmult*3),
        ↪ fill=(50,150,50))
    draw.text(((n1x+n2x)/2, (n1y+n2y)/2),f"{elem.elem_num}",font=font_num,
        ↪ fill=(50,210,50))

for node in self.nodes:
    size=coordmult*6

    ux = def_scale * self.U_result[2 * (node.node_num - 1)]
    uy = def_scale * self.U_result[2 * (node.node_num - 1) + 1]

    nx = xshift + scale * node.pos[0] + ux
    ny = yshift - scale * node.pos[1] - uy

    if ny <= maxY:
        maxY = ny
    if ny >= minY:
        minY = ny

    draw.ellipse([nx-size/2,ny-size/2,nx+size//2,ny+size//2]),
        ↪ fill=(50,150,50))


```



```

draw.text((nx+size,ny+size),f"{node.node_num}",font=font_num,
↳ fill=(50,210,50))

draw.text((coordmult*250, img.height + (node.node_num - num_nodes - 1.5) *
↳ coordmult*40), f"{node.node_num}",font=font_table, fill="white")
draw.text((coordmult*400, img.height + (node.node_num - num_nodes - 1.5) *
↳ coordmult*40), f"{np.round(node.pos[0],2)}",font=font_table,
↳ fill="white")
draw.text((coordmult*550, img.height + (node.node_num - num_nodes - 1.5) *
↳ coordmult*40), f"{np.round(node.pos[1],2)}",font=font_table,
↳ fill="white")
draw.text((coordmult*700, img.height + (node.node_num - num_nodes - 1.5) *
↳ coordmult*40), f"{np.round(self.U_result[2 * (node.node_num -
↳ 1)],3)}",font=font_table, fill="white")
draw.text((coordmult*850, img.height + (node.node_num - num_nodes - 1.5) *
↳ coordmult*40), f"{np.round(self.U_result[2 * (node.node_num - 1) +
↳ 1],3)}",font=font_table, fill="white")
draw.text((coordmult*1150, img.height + (node.node_num - num_nodes - 1.5) *
↳ coordmult*40), f"{np.round(self.F_result[2 * (node.node_num -
↳ 1)],3)}",font=font_table, fill="white")
draw.text((coordmult*1450, img.height + (node.node_num - num_nodes - 1.5) *
↳ coordmult*40), f"{np.round(self.F_result[2 * (node.node_num - 1) +
↳ 1],3)}",font=font_table, fill="white")


w,h = img.size
img = img.crop((0,maxY-coordmult*40,w,h))
img.show()
img.save(target_loc)

def calculate(self):
    self.calc_elems()
    nodes_num = len(self.nodes)
    U = np.array([])
    F = np.array([])

    K = np.zeros((2 * nodes_num, 2 * nodes_num))

    for elem in self.elements:
        a = elem.angle
        c = np.cos(a)
        s = np.sin(a)
        n1 = elem.nodes[0]      #node number, NOT index
        n2 = elem.nodes[1]

        K_elem = (elem.A * elem.E / elem.L) * np.array([[c**2, c*s, -c**2, -c*s],
        [c*s, s**2, -c*s, -s**2],
        [-c**2, -c*s, c**2, c*s],
        [-c*s, -s**2, c*s, s**2]])

        n = [n1, n2]
        for na in [0, 1]:
            for nb in [0, 1]:
                for i in [0, 1]:
                    for j in [0, 1]:
                        K[2 * (n[na] - 1) + i, 2 * (n[nb] - 1) + j] += K_elem[2 * na +
                        ↳ i, 2 * nb + j]

    for n in self.nodes:

```

```

        for i in [0, 1]:
            if (n.displacement[i] == None and n.force[i] == None):
                U=np.append(U, None)
                F=np.append(F, 0)
            else:
                U=np.append(U, n.displacement[i])
                F=np.append(F, n.force[i])

    Uh = np.array([])
    Fh = np.array([])
    Kh = np.array([])

    count = 0
    for i in range(2 * nodes_num):
        if U[i] != 0:
            count += 1
            Uh=np.append(Uh, U[i])
            Fh=np.append(Fh, F[i])
            K_row = np.array([])
            for j in range(2 * nodes_num):
                if U[j] != 0:
                    K_row = np.append(K_row, K[i, j])
            Kh = np.append(Kh, K_row)

    print(K)

    U_result = np.matmul(np.linalg.inv(np.reshape(Kh, (count, count))), Fh)
    self.U_result = np.array([])
    count = 0
    for i in range(2 * nodes_num):
        if U[i] == 0:
            self.U_result = np.append(self.U_result, 0)
        else:
            self.U_result = np.append(self.U_result, U_result[count])
            count += 1

    print(self.U_result)

    self.F_result = np.matmul(K, self.U_result)
    print(self.F_result)

    def print_result(self):
        np.set_printoptions(precision=4)
        print("\tu_x[mm]\tu_y[mm]\t\tfx[N]\t\tfy[N]")
        for i in range(len(self.nodes)):
            ↪ print(f"{i+1}.\t{np.round(self.U_result[2*i]*1000,2)}\t\t{np.round(self.U_result[2*i+1]

```

```

mod = Model()

a=3
b=1.6
c=7
d=45*10**(-3)
F=90000
E=150
A = ((1.3 * d)**2 - d**2) * np.pi / 4

```

```
mod.addNode(1, a+c, 1.5*b, uy=0, fx=-3*F)
mod.addNode(2, 0, b, ux=0, fy=F)
mod.addNode(3, a, b, fy=-F)
mod.addNode(4, a+c, b)
mod.addNode(5, a, 0, ux=0, uy=0)

mod.addElement(1, 1, 4, E*10**9, A)
mod.addElement(2, 1, 3, E*10**9, A)
mod.addElement(3, 3, 4, E*10**9, A)
mod.addElement(4, 5, 4, E*10**9, A)
mod.addElement(5, 3, 5, E*10**9, A)
mod.addElement(6, 2, 3, E*10**9, A)
mod.addElement(7, 2, 5, E*10**9, A)

np.set_printoptions(precision=4)
mod.calculate()
mod.draw(deform=True)
mod.print_result()
```