Elementy projektowania inżynierskiego

Rozwiązanie problemu belki zginanej metodą MES

Uwagi:

- W środowisku Spyder generowane wykresy pokazują się na panelu **Plots**, można wymusić rysowanie wykresów w oknie za pomocą poleceń:

```
from IPython import get_ipython
get_ipython().run_line_magic("matplotlib", "qt")
```

- Ponowne włączenie panelu **Plots**:

```
from IPython import get_ipython
get_ipython().run_line_magic("matplotlib", "inline")
```

- Zmiana sposobu wyświetlania tablic:

```
np.set_printoptions(formatter={"float": lambda x: f"{x:10.4g}"}, suppress = True)
```

- Oprócz używanej we wcześniejszym ćwiczeniu biblioteki *NumPy* (dodającej obsługę dużych,46 wielowymiarowych tabel i macierzy) istnieją biblioteki wspomagające obliczenia numeryczne (*SciPy*; https://scipy.org) i symboliczne (*SymPy*; https://docs.sympy.org).
- Biblioteka SciPy to właściwie zbiór bibliotek obliczeniowych wysokiego poziomu oparty na bibliotece NumPy. Moduły w SciPy oferują specyficzne dla danej dziedziny metody obliczeniowe wysokiego poziomu: operacje algebry liniowej, metody optymalizacji, interpolacji, całkowania i wiele innych.
- Biblioteka SymPy zapewnia w pełni funkcjonalny system algebry komputerowej (CAS). która pozwala zapisywać wyrażenia zawierające symbole i wykonywać na nich operacje związane z analizą funkcji, algebrą wielomianów, rozwiązywaniem równań, kombinatoryką, statystyką i wiele innych.

Pochodne funkcji kształtu

- Import biblioteki.

```
import sympy as sp
```

- Definicja zmiennych symbolicznych.

```
x = sp.Symbol('x')
L = sp.Symbol('L')
```

- Definicja funkcji kształtu.

```
N1 = 1 - 3 * (x / L)**2 + 2 * (x / L)**3
N2 = x * (1 - 2 * (x / L) + (x / L)**2)
N3 = 3 * (x / L)**2 - 2 * (x / L)**3
N4 = x * (-x / L + (x / L)**2)
```

- Budowa macierzy funkcji kształtu.

```
N = sp.Matrix([[N1, N2, N3, N4]])
```

- Różniczkowanie macierzy funkcji kształtu.

```
d2N = sp.diff(sp.diff(N, x), x)
d3N = sp.diff(sp.diff(sp.diff(N, x), x), x)
```

Wymnożenie i wyświetlenie wyników.

```
N = sp.expand(N)
```

```
print("\n=== N ===")
sp.pprint(N)

print("\n=== d2N ===")
sp.pprint(d2N)

print("\n=== d3N ===")
sp.pprint(d3N)
```

- Wynik działania skryptu:

```
=== N ===
⎡       2      3         2    3     2      3     2    3⎤
⎢    3⋅x    2⋅x       2⋅x    x   3⋅x    2⋅x     x    x ⎥
⎢1 - ──── + ────  x - ──── + ── - ──── - ──── - ── + ──⎥
⎢      2      3        L      2    2      3     L     2⎥
⎣     L      L               L    L      L           L ⎦

=== d2N ===
⎡  6    12⋅x    4   6⋅x  6    12⋅x    2   6⋅x⎤
⎢- ── + ────  - ─ + ──  ── - ────  - ─ + ──⎥
⎢   2     3     L     2   2     3     L     2⎥
⎣  L     L           L   L     L           L ⎦

=== d3N ===
⎡12  6   -12   6 ⎤
⎢──  ──  ────  ──⎥
⎢ 3   2    3    2⎥
⎣L   L    L    L ⎦
```

- Utworzenie wyników w formie możliwej do bezpośredniego przeniesienia do kodu programu (zmiana wyników na łańcuchy znaków, dodanie nazw zmiennych i symbolu przypisania, zmiana ciągu **Matrix** na **np.array**, wydruk).

```
sN = str(f"\nN = {N}")
sd2N = str(f"\nd2N = {d2N}")
sd3N = str(f"\nd3N = {d3N}")

sN = sN.replace("Matrix", "np.array")
sd2N = sd2N.replace("Matrix", "np.array")
sd3N = sd3N.replace("Matrix", "np.array")

print(sN)
print(sd2N)
print(sd3N)
```

- Wynik działania skryptu (część):

```
N = np.array([[1 - 3*x**2/L**2 + 2*x**3/L**3, ...
d2N = np.array([[-6/L**2 + 12*x/L**3, ...
d3N = np.array([[12/L**3, ...
```

- W przypadku zainstalowanego pakietu **LaTeX**, konsola **IPython** będzie mogła wyświetlać wyniki z jego pomocą:

In [2]: N
Out[2]:

$$\left[1 - \frac{3x^2}{L^2} + \frac{2x^3}{L^3} \quad x - \frac{2x^2}{L} + \frac{x^3}{L^2}\right]$$

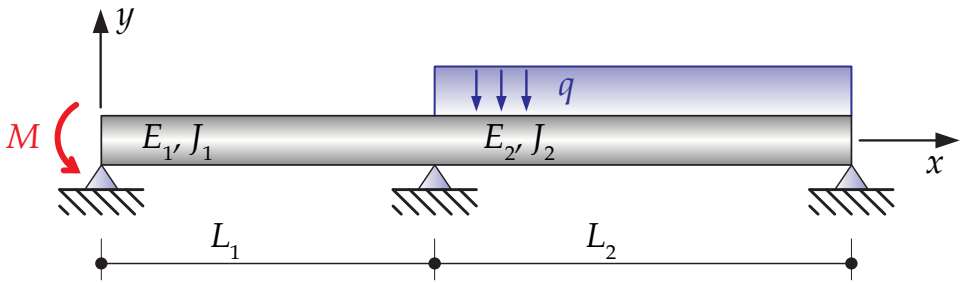
In [3]: d2N
Out[3]:

$$\left[-\frac{6}{L^2} + \frac{12x}{L^3} \quad -\frac{4}{L} + \frac{6x}{L^2}\right]$$

In [4]: d3N
Out[4]:

$$\left[\frac{12}{L^3} \quad \frac{6}{L^2}\right]$$

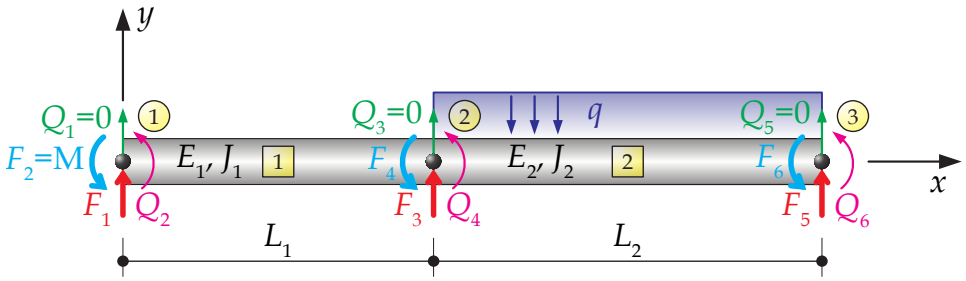
Belka zginana



- $L_1 = 6$ m
- $L_2 = 8$ m
- $J_1 = 5 \cdot 10^{-5}$ m^4
- $J_2 = 1 \cdot 10^{-4}$ m^4

- $E = 200$ GPa
- $q = -10$ kN/m
- $M = 20$ kNm
- $h_1 = 0{,}25$ m
- $h_2 = 0{,}32$ m

Dyskretyzacja belki



Import bibliotek i dodatkowe ustawienia

```
import numpy as np
import matplotlib as mpl
from IPython import get_ipython

get_ipython().run_line_magic("matplotlib", "qt")
np.set_printoptions(formatter={"float": lambda x: f"{x:10.4g}"}, suppress = True)
```

Definicja macierzy sztywności i wektora równoważników węzłowych obciążenia

- Są to zwykłe funkcje języka Python, które zwracają tablice typu **`array`** z biblioteki NumPy.

```
def Ke(EJ, L):
  return np.array([[ 12 * EJ / L**3,  6 * EJ / L**2, -12 * EJ / L**3,  6 * EJ / L**2 ],
                   [  6 * EJ / L**2,  4 * EJ / L,     -6 * EJ / L**2,  2 * EJ / L    ],
                   [-12 * EJ / L**3, -6 * EJ / L**2,  12 * EJ / L**3, -6 * EJ / L**2 ],
                   [  6 * EJ / L**2,  2 * EJ / L,     -6 * EJ / L**2,  4 * EJ / L    ]])

def Pe(q, L):
  return np.array([[ q *    L / 2  ],
                   [ q * L**2 / 12 ],
                   [ q *    L / 2  ],
                   [-q * L**2 / 12 ]])
```

Dane geometryczne, materiałowe i obciążenie

```
L1 = 6
J1 = 5e-5
h1 = 0.25

L2 = 8
J2 = 1e-4
h2 = 0.32

q = -10e3

M = 20e3

E = 200e9
```

Liczba węzłów modelu, liczba węzłów elementu, liczba stopni swobody w węźle elementu

```
ln = 3
lw = 2
lq = 2
```

Budowa i wyświetlenie macierzy Boole'a

```
B1 = np.zeros((lw * lq, ln * lq))
B2 = np.zeros((lw * lq, ln * lq))

print(f"{B1} = B1", end="\n\n")
print(f"{B2} = B2", end="\n\n")

B1[[0, 1, 2, 3], [0, 1, 2, 3]] = 1
B2[[0, 1, 2, 3], [2, 3, 4, 5]] = 1

print(f"{B1} = B1", end="\n\n")
print(f"{B2} = B2", end="\n\n")
```

Obliczenie i wyświetlenie macierzy sztywności elementów

```
K1 = Ke(E * J1, L1)
K2 = Ke(E * J2, L2)

print(f"{K1} = K1", end="\n\n")
print(f"{K2} = K2", end="\n\n")
```

Obliczenie i wyświetlenie równoważników węzłowych obciążenia

```
P1 = Pe(0, L1)
P2 = Pe(q, L2)

print(f"{P1} = P1", end="\n\n")
print(f"{P2} = P2", end="\n\n")
```

Agregacja i wyświetlenie macierzy sztywności

```
K = B1.T.dot(K1).dot(B1) + B2.T.dot(K2).dot(B2)

print(f"{K} = K", end="\n\n")
```

Agregacja i wyświetlenie równoważników węzłowych obciążenia

```
P = B1.T.dot(P1) + B2.T.dot(P2)

print(f"{P} = P", end="\n\n")
```

Budowa i wyświetlenie wektora sił węzłowych

```
F = np.zeros((ln * lq, 1))
F[1] = M

print(f"{F} = F", end="\n\n")
```

Uwzględnienie warunków brzegowych, wyświetlanie macierzy

- Domyślnie przy przypisywaniu tablic, Python nie tworzy ich kopii tylko referencje, dlatego jawnie kopiujemy tablice do nowych zmiennych.

```
Kw = np.copy(K)
Fw = np.copy(F)
Pw = np.copy(P)

Kw[0, :] = 0
Kw[:, 0] = 0
Kw[0, 0] = 1
```

```
Fw[0] = 0
Pw[0] = 0

Kw[2, :] = 0
Kw[:, 2] = 0
Kw[2, 2] = 1

Fw[2] = 0
Pw[2] = 0

Kw[4, :] = 0
Kw[:, 4] = 0
Kw[4, 4] = 1

Fw[4] = 0
Pw[4] = 0

print(f"{Kw} = Kw", end="\n\n")
print(f"{Fw} = Fw", end="\n\n")
print(f"{Pw} = Pw", end="\n\n")
```

Rozwiązanie układu równań i wyświetlenie przemieszczeń

```
Q = np.linalg.solve(Kw, Pw + Fw)

print(f"{Q} = Q", end="\n\n")
```

Obliczenie i wyświetlenie reakcji

```
R = np.dot(K, Q) - P

print(f"{R} = R", end="\n\n")
```

Obliczenie i wyświetlenie wektorów przemieszczeń dla elementów

```
Q1 = np.dot(B1, Q)
Q2 = np.dot(B2, Q)

print(f"{Q1} = Q1", end="\n\n")
print(f"{Q2} = Q2", end="\n\n")
```

Definicja wektora funkcji kształtu

```
def N(x, L):
  return np.array([[ 1 - 3 * x**2 / L**2 + 2 * x**3 / L**3, x - 2 * x**2 / L + x**3 / L**2,
                       3 * x**2 / L**2 - 2 * x**3 / L** 3,      -x**2 / L + x**3 / L**2]])
```

Definicja wektorów pochodnych funkcji kształtu

- Wyrażenia policzone wcześniej z wykorzystaniem obliczeń symbolicznych.

```
def d2N(x, L):
  return np.array([[ -6 / L**2 + 12 * x / L**3, -4 / L + 6 * x / L**2,
                      6 / L**2 - 12 * x / L**3, -2 / L + 6 * x / L**2 ]])

def d3N(x, L):
  return np.array([[ 12 / L**3, 6 / L**2,
                    -12 / L**3, 6 / L**2 ]])
```

Definicja funkcji przemieszczeń dla elementu

- Wszystkie parametry funkcji są skalarami, ze względu na późniejszą wektoryzację funkcji.

```
def u(x, L, q1, q2, q3, q4):
  return np.dot(N(x, L), [[q1], [q2], [q3], [q4]])
```

Definicja funkcji momentu gnącego dla elementu

```
def M(x, L, q1, q2, q3, q4, E, J):
  return E * J * np.dot(d2N(x, L), [[q1], [q2], [q3], [q4]])
```

Definicja funkcji siły stycznej dla elementu

```
def V(x, L, q1, q2, q3, q4, E, J):
  return E * J * np.dot(d3N(x, L), [[q1], [q2], [q3], [q4]])
```

Definicja funkcji naprężeń normalnych w elemencie

```
def s(x, L, q1, q2, q3, q4, E, h):
  return E * h / 2 * np.dot(d2N(x, L), [[q1], [q2], [q3], [q4]])
```

Obliczenie i wyświetlenie wektorów momentu gnącego w elementach

```
M1N = np.vstack((M( 0, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, J1),
                 M(L1, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, J1)))
M2N = np.vstack((M( 0, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, J2),
                 M(L2, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, J2)))

print(f"{M1N} = M1N", end="\n\n")
print(f"{M2N} = M2N", end="\n\n")
```

Obliczenie i wyświetlenie wektorów siły stycznej w elementach

```
V1N = np.vstack((V( 0, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, J1),
                 V(L1, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, J1)))
V2N = np.vstack((V( 0, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, J2),
                 V(L2, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, J2)))

print(f"{V1N} = V1N", end="\n\n")
print(f"{V2N} = V2N", end="\n\n")
```

Obliczenie i wyświetlenie wektorów naprężeń normalnych w elementach

```
s1N = np.vstack((s( 0, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, h1),
                 s(L1, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, h1)))

s2N = np.vstack((s( 0, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, h1),
                 s(L2, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, h2)))

print(f"{s1N} = s1N", end="\n\n")
print(f"{s2N} = s2N", end="\n\n")
```

Wektoryzacja funkcji przemieszczeń, momentu gnącego, siły stycznej i naprężeń

```
vecu = np.vectorize(u)
vecM = np.vectorize(M)
vecV = np.vectorize(V)
vecs = np.vectorize(s)
```

Generowanie współrzędnych dla elementów

```
x1 = np.linspace(0, L1, 20)
x2 = np.linspace(0, L2, 20)
```

Obliczenie wartości przemieszczeń w elementach

```
u1 = vecu(x1, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0])
u2 = vecu(x2, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0])
```

Obliczenie wartości momentu gnącego w elementach

```
M1 = vecM(x1, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, J1)
M2 = vecM(x2, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, J2)
```

Obliczenie wartości siły stycznej w elementach

```
V1 = vecV(x1, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, J1)
V2 = vecV(x2, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, J2)
```

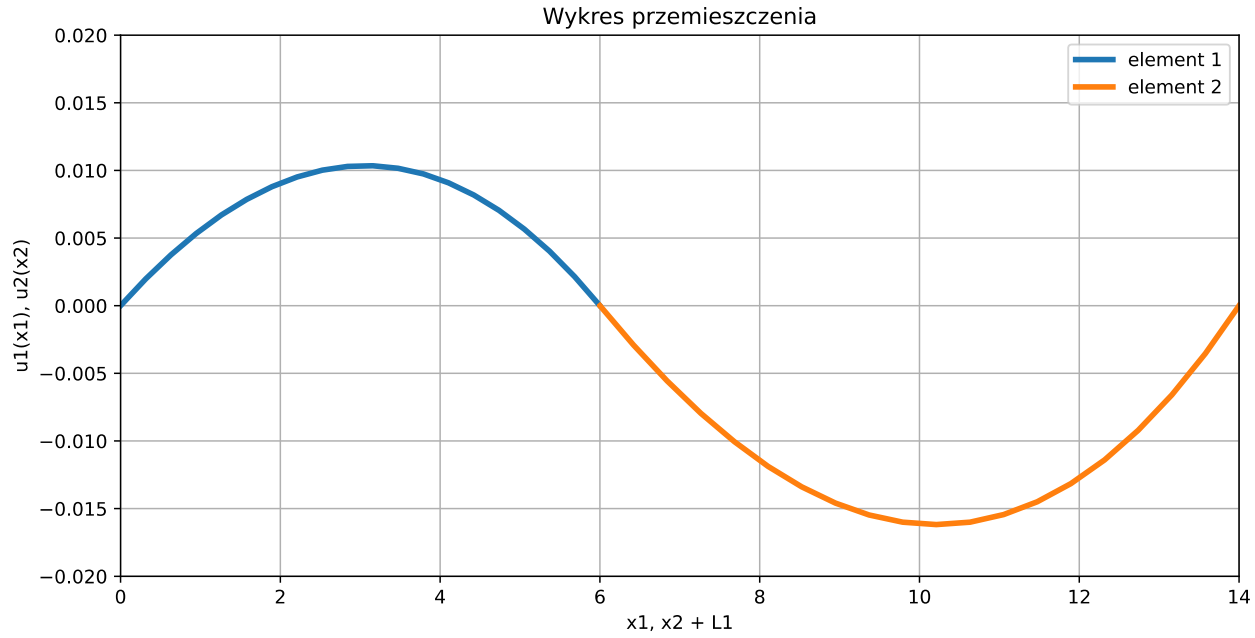
Obliczenie wartości naprężeń normalnych w elementach

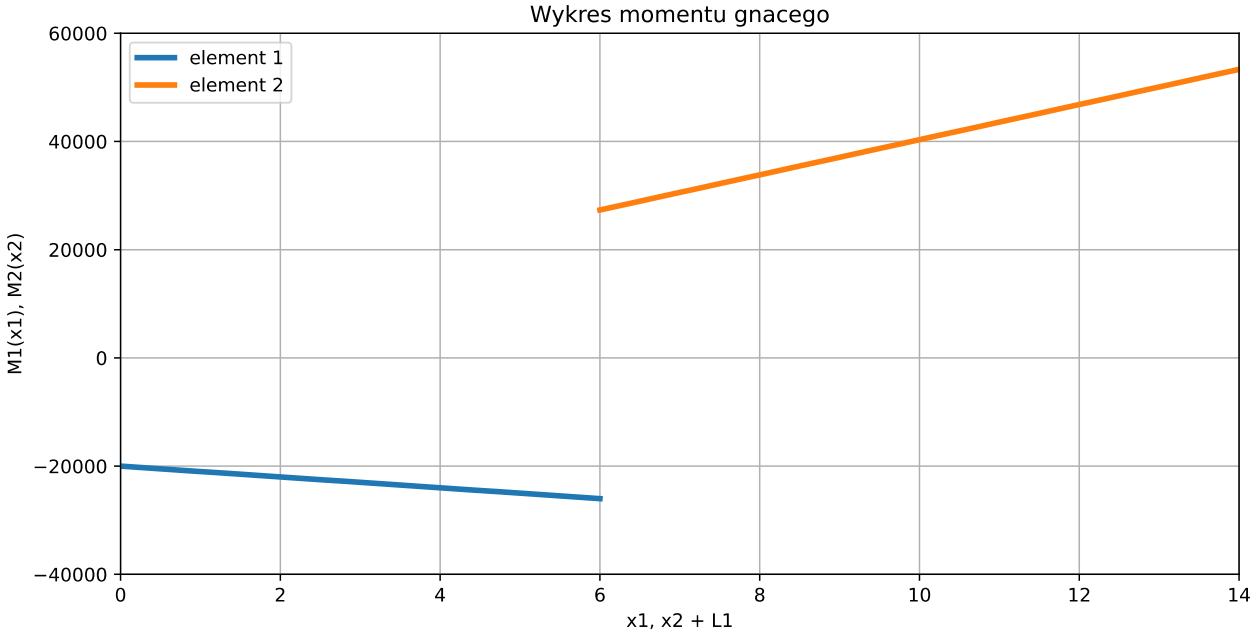
```
s1 = vecs(x1, L1, Q1[0, 0], Q1[1, 0], Q1[2, 0], Q1[3, 0], E, h1)
s2 = vecs(x2, L2, Q2[0, 0], Q2[1, 0], Q2[2, 0], Q2[3, 0], E, h2)
```

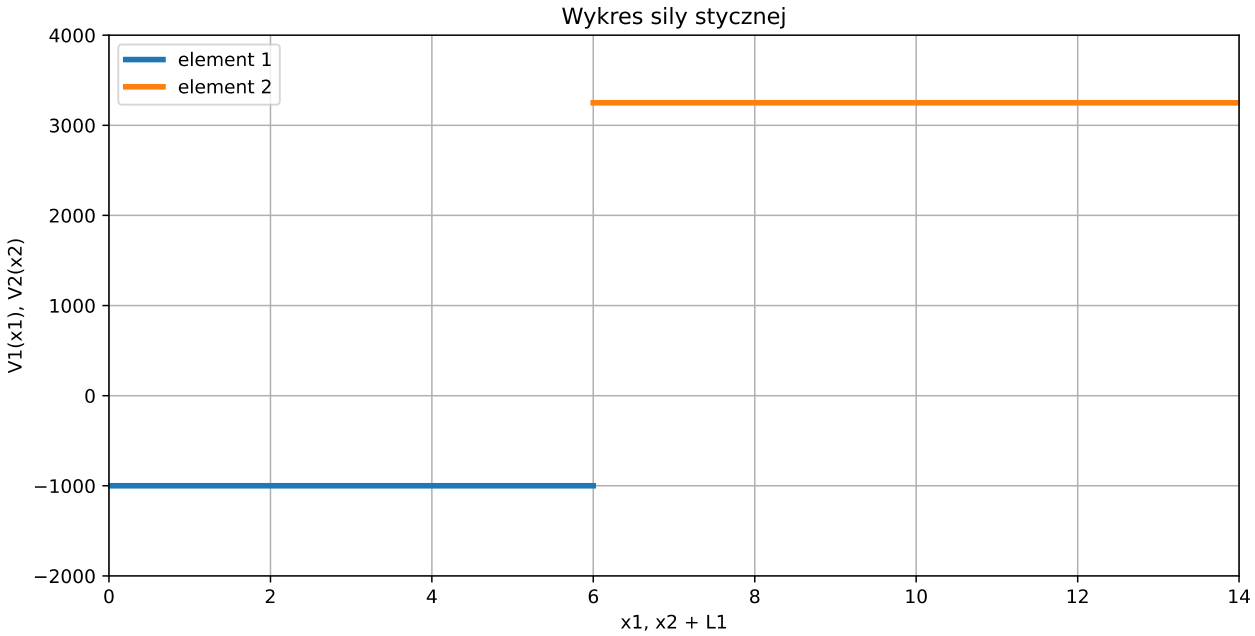
Wykresy

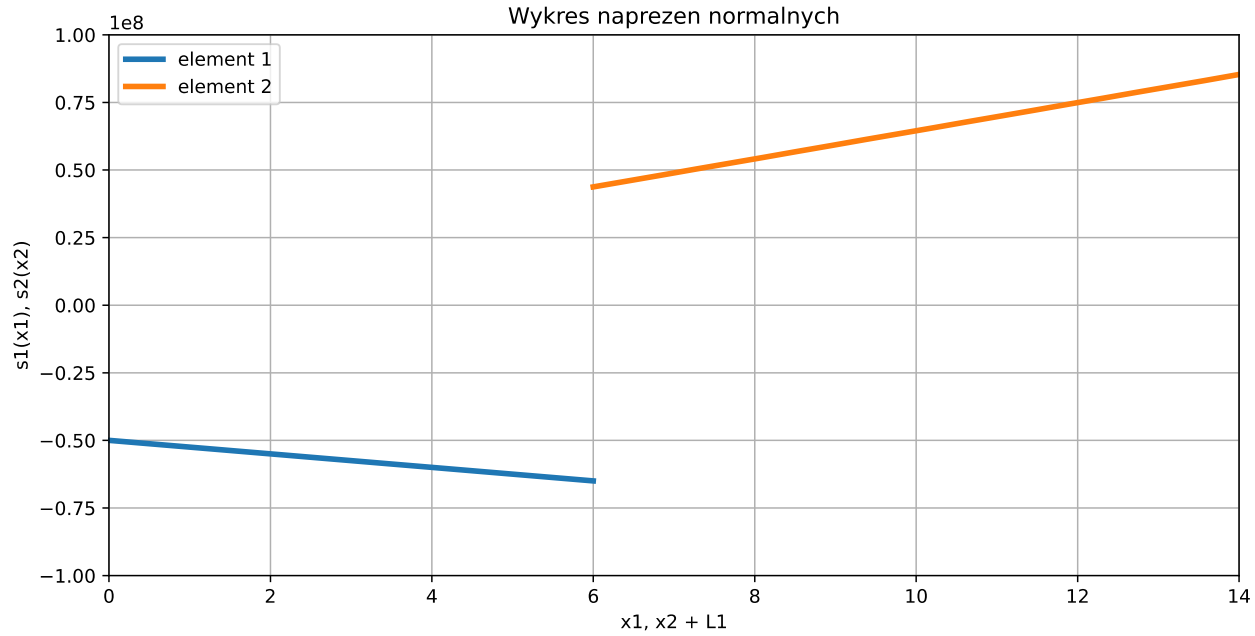
```
mpl.pyplot.figure(1)
mpl.pyplot.clf()

mpl.pyplot.subplot(2, 1, 1)
mpl.pyplot.plot(x1, u1, label = "element 1", linewidth = 3)
mpl.pyplot.plot(x2 + L1, u2, label = "element 2", linewidth = 3)
mpl.pyplot.title("Wykres przemieszczenia")
mpl.pyplot.legend()
mpl.pyplot.xlabel("x1, x2 + L1")
mpl.pyplot.ylabel("u1(x1), u2(x2)")
mpl.pyplot.xlim(0, L1 + L2)
mpl.pyplot.ylim(-0.02, 0.02)
mpl.pyplot.grid()

mpl.pyplot.subplot(2, 1, 2)
mpl.pyplot.plot(x1, M1, label = "element 1", linewidth = 3)
mpl.pyplot.plot(x2 + L1, M2, label = "element 2", linewidth = 3)
mpl.pyplot.title("Wykres momentu gnacego")
mpl.pyplot.legend()
mpl.pyplot.xlabel("x1, x2 + L1")
mpl.pyplot.ylabel("M1(x1), M2(x2)")
mpl.pyplot.xlim(0, L1 + L2)
mpl.pyplot.ylim(-40e3, 60e3)
mpl.pyplot.grid()

mpl.pyplot.tight_layout()

mpl.pyplot.figure(2)
mpl.pyplot.clf()

mpl.pyplot.subplot(2, 1, 1)
mpl.pyplot.plot(x1, V1, label = "element 1", linewidth = 3)
mpl.pyplot.plot(x2 + L1, V2, label = "element 2", linewidth = 3)
```

```
mpl.pyplot.title("Wykres sily stycznej")
mpl.pyplot.legend()
mpl.pyplot.xlabel("x1, x2 + L1")
mpl.pyplot.ylabel("V1(x1), V2(x2)")
mpl.pyplot.xlim(0, L1 + L2)
mpl.pyplot.ylim(-2e3, 4e3)
mpl.pyplot.grid()

mpl.pyplot.subplot(2, 1, 2)
mpl.pyplot.plot(x1, s1, label = "element 1", linewidth = 3)
mpl.pyplot.plot(x2 + L1, s2, label = "element 2", linewidth = 3)
mpl.pyplot.title("Wykres naprezen normalnych")
mpl.pyplot.legend()
mpl.pyplot.xlabel("x1, x2 + L1")
mpl.pyplot.ylabel("s1(x1), s2(x2)")
mpl.pyplot.xlim(0, L1 + L2)
mpl.pyplot.ylim(-100e6, 100e6)
mpl.pyplot.grid()

mpl.pyplot.tight_layout()
```


Wykres przemieszczenia
element 1
element 2
u1(x1), u2(x2)
x1, x2 + L1
0.020
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.020
0
2
4
6
8
10
12
14


Wykres momentu gnacego
element 1
element 2
M1(x1), M2(x2)
x1, x2 + L1
60000
40000
20000
0
−20000
−40000
0
2
4
6
8
10
12
14


Wykres sily stycznej
element 1
element 2
V1(x1), V2(x2)
x1, x2 + L1
4000
3000
2000
1000
0
−1000
−2000
0
2
4
6
8
10
12
14


Wykres naprezen normalnych
1e8
element 1
element 2
s1(x1), s2(x2)
x1, x2 + L1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14