Aplikacja do statycznej i dynamicznej analizy kratownicy

Opracowali:
inż. Jakub Wilczek
inż. Artur Wilczek

Kraków 2021

SPIS TREŚCI

Strona

1 Wprowadzenie teoretyczne **5**
- 1.1 Metoda elementów skończonych . . . 5
- 1.2 Zagadnienia statyczne . . . 5
- 1.3 Zagadnienia dynamiczne . . . 6

2 Schemat kratownicy i jej parametry **7**
- 2.1 Schemat kratownicy . . . 7
- 2.2 Parametry kratownicy . . . 7

3 Wyznaczenie macierzy sztywności i bezwładności **8**
- 3.1 Macierze sztywności w lokalnym układzie współrzędnych . . . 8
- 3.2 Macierze sztywności w globalnym układzie współrzędnych . . . 8
- 3.3 Zagregowana macierz sztywności w globalnym układzie współrzędnych . . . 10
- 3.4 Macierze bezwładności w lokalnym układzie współrzędnych . . . 11
- 3.5 Macierze bezwładności w globalnym układzie współrzędnych . . . 11
- 3.6 Zagregowana macierz bezwładności w globalnym układzie współrzędnych . . . 13

4 Macierze po agregacji i nadaniu warunków brzegowych **14**
- 4.1 Macierz sztywności po agregacji i nadaniu warunków brzegowych . . . 14
- 4.2 Macierz bezwładności po agregacji i nadaniu warunków brzegowych . . . 14

5 Wektor obciążeń **15**

6 Wektor przemieszczeń węzłowych **16**

7 Macierze wektorów własnych i częstotliwości drgań własnych **17**

8 Analiza statyczna - program Python **19**

9 Analiza statyczna - Ansys **20**
- 9.1 Analiza w programie ANSYS . . . 20
- 9.2 Wyniki analizy . . . 20

10 Analiza dynamiczna **21**
- 10.1 Wartości częstotliwości drgań własnych otrzymane przy pomocy Ansysa oraz kodu Python 21
- 10.2 Porównanie postaci drgań własnych w programie ANSYS oraz Python . . . 21

11 Analiza uzyskanych wyników oraz wnioski **24**
- 11.1 Porównanie wyników otrzymanych w programie ANSYS i Python . . . 24
- 11.2 Analiza wyników i wnioski . . . 24

12 Załącznik nr 1 **25**

1. Wprowadzenie teoretyczne

1.1. Metoda elementów skończonych

Metoda elementów skończonych (Finite Element Method) jest obecnie jedną z najczęściej stosowanych metod obliczeniowych w nauce i technice. Jest ona w dzisiejszych czasach jedną z podstawowych metod do rozwiązywania problemów inżynierskich, brzegowych i początkowo-brzegowych. Określa się obszar rozwiązania, brzeg z warunkami brzegowymi, element objętości obszaru oraz element brzegu. Na brzegu mogą być zdefiniowane obciążenia, można uwzględniać siły masowe, które działają na cały obszar rozwiązania.

Algorytm metody elementów skończonych:

Require: problem w postaci równań różniczkowych cząstkowych
Require: warunki brzegowe
podział obszaru rozwiązania na elementy skończone
all for elementy obszaru **do**
wyznaczyć macierze opisujące właściwości elementów
agregować macierze elementów do macierzy globalnych
end for
for all elementy na brzegu **do**
odebrać odpowiednie stopnie swobody
nałożyć warunki Neumana
nałożyć warunki mieszane
end for
rozwiązać uzyskany układ równań algebraicznych
wizualizacja rozwiązania

1.2. Zagadnienia statyczne

Zagadnienia statyczne sprowadzająsię do rozwiązywania równań, w których nie występują pochodne wielkości po czasie. Równanie ma postać:

$$[k]\{u_i\} = \{P_i\}$$

gdzie:
$[k]$ - macierz sztywności
$\{u_i\}$ - macierz przemieszczeń węzłowych w lokalnym układzie współrzędnych
$\{P_i\}$ - macierz obciążeń węzłowych w lokalnym układzie współrzędnych

$$\{u_i\} = [DC]\{u_i^\circ\}$$

gdzie:
$\{u_i^\circ\}$ - macierz przemieszczeń węzłowych w globalnym układzie współrzędnych
$[DC]$ - macierz kosinusów kierunkowych równa:

$$[DC] = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \\ 0 & 0 & -\sin\theta & \cos\theta \end{bmatrix} \quad (1)$$

$$\cos\theta = \frac{x_2^\circ - x_1^\circ}{l}$$

$$\sin\theta = \frac{y_2^\circ - y_1^\circ}{l}$$

Macierz obciążeń w lokalnym układzie odniesienia:

$$\{P_i\} = [DC]\{P_i^\circ\}$$

gdzie:
$\{P_i^\circ\}$ - macierz obciążeń węzłowych w globalnym układzie współrzędnych
$\{P_i\}$ - macierz obciążeń węzłowych w lokalnym układzie współrzędnych
Skoro w lokalnym układzie odniesienia prawdziwa jest zależność

$$[k]\{u_i\} = \{P_i\}$$

to po dokonaniu podstawień i przekształceń możemy zauważyć że prawdziwa będzie zależność:

$$[k^\circ]\{u_i^\circ\} = \{P_i^\circ\}$$

gdzie $[k^\circ]$ nazywamy macierzą sztywności w globalnym układzie odniesienia.

$$[k^\circ] = [DC]^T[k][DC]$$

gdzie $[k]$

$$[k] = \frac{AE}{l}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2}$$

i podobną zależność możemy zapisać dla macierzy bezwładności:

$$[m^\circ] = [DC]^T[m][DC]$$

1.3. Zagadnienia dynamiczne

Rozwiązanie zagadnień dynamicznych w dużym stopniu opiera się na macierzach sztywności i bezwładności przygotowanych przy analizie statycznej. Ustalenie dynamiki układu sprowadza się do rozwiązania poniższych równań dynamicznych.

$$[m]\{\ddot{u}_i\} + [c]\{\dot{u}_i\} + [k]\{u_i\} = \{P_i\}$$

gdzie kropkami nad zmienną oznaczono pochodną względem czasu:

$$\{\ddot{u}_i\} = \frac{d^2}{dt^2}\{u_i\}$$

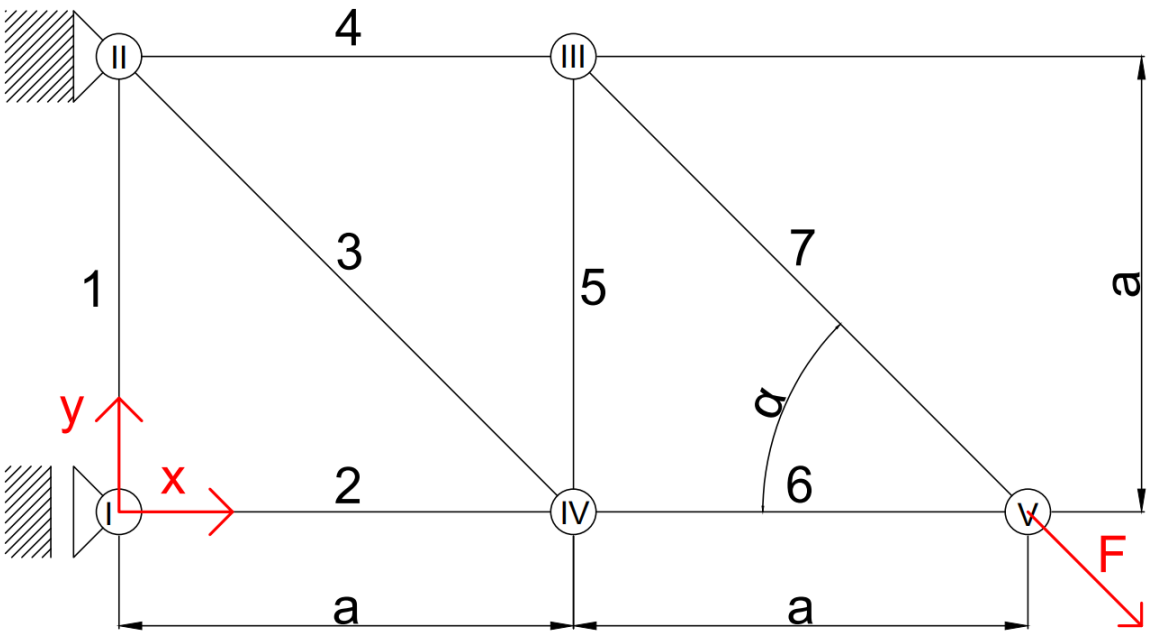
$$\{\dot{u}_i\} = \frac{d}{dt}\{u_i\}$$

Nowym względem analizy statycznej elementem jest macierz tłumienia postaci:

$$[c] = \int_\Omega [N]^T \mu [N] d\Omega$$

2. Schemat kratownicy i jej parametry

2.1. Schemat kratownicy



Rysunek 1: Schemat projektowanej kratownicy i umiejscowienie siły obciążającej F

2.2. Parametry kratownicy

Tabela 1: Parametry materiałowe oraz wartość siły

Dana	Wartość
a - długość pręta	1 [m]
b - długość pręta	1 [m]
E - moduł sprężystości	$2 \cdot 10^{11}$ [Pa]
d - średnica pręta	10 [mm]
F - obciążenie	141,4 [kN]
α - kąt działania siły	45°
A - pole przekroju pręta	$3.14 \cdot 10^{-4}$ $[m^2]$

Tabela 2: Zestawienie parametrów geometrycznych i materiałowych kratownicy

	Współrzędne			**Wartość**			
Element	**Węzeł 1**	**Węzeł 2**	**Długość elementu [m]**	**cos θ**	**sin θ**	**E [Pa]**	**A $[m^2]$**
I-II	(0,0)	(0,1)	1	0	1	$2 \cdot 10^{11}$	$3.14 \cdot 10^{-4}$
I-IV	(0,0)	(1,0)	1	1	0	$2 \cdot 10^{11}$	$3.14 \cdot 10^{-4}$
II-IV	(0,1)	(1,0)	1.41	0.707	-0.707	$2 \cdot 10^{11}$	$3.14 \cdot 10^{-4}$
II-III	(0,1)	(1,1)	1	1	0	$2 \cdot 10^{11}$	$3.14 \cdot 10^{-4}$
III-IV	(1,1)	(1,0)	1	0	-1	$2 \cdot 10^{11}$	$3.14 \cdot 10^{-4}$
III-V	(1,1)	(2,0)	1.41	0.707	-0.707	$2 \cdot 10^{11}$	$3.14 \cdot 10^{-4}$
IV-V	(1,0)	(2,0)	1	1	0	$2 \cdot 10^{11}$	$3.14 \cdot 10^{-4}$

3. Wyznaczenie macierzy sztywności i bezwładności

3.1. Macierze sztywności w lokalnym układzie współrzędnych

Poniżej przedstawione zostały wybrane macierze sztywności w lokalnym układzie współrzędnych dla prętów oznaczonych na rysunku nr 1 numerami "1,2,3" łączących kolejno węzły I-II, I-IV oraz II-IV.

Dla pręta numer 1:

$$[K_{1-2}] = 10^7 \cdot \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 6.28 & 0 & -6.28 \\ 0 & 0 & 0 & 0 \\ 0 & -6.28 & 0 & 6.28 \end{bmatrix} \tag{3}$$

Dla pręta numer 2:

$$[K_{1-4}] = 10^7 \cdot \begin{bmatrix} 6.28 & 0 & -6.28 & 0 \\ 0 & 0 & 0 & 0 \\ -6.28 & 0 & 6.28 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

Dla pręta numer 3:

$$[K_{2-4}] = 10^7 \cdot \begin{bmatrix} 2.2203 & -2.2203 & -2.2203 & 2.2203 \\ -2.2203 & 2.2203 & 2.2203 & -2.2203 \\ -2.2203 & 2.2203 & 2.2203 & -2.2203 \\ 2.2203 & -2.2203 & -2.2203 & 2.2203 \end{bmatrix} \tag{5}$$

Macierze sztywności dla pozostały prętów zostały wyznaczone analogicznie, możliwy jest ich podgląd przy pomocy dołączonego kodu programu Python.

W kolejnym korku, macierze wyznaczone dla wszystkich prętów zostały agregowane do jednej dużej macierzy znajdującej się w nowym, głównym układzie współrzędnych.

3.2. Macierze sztywności w globalnym układzie współrzędnych

Poniżej przedstawione zostały wybrane macierze sztywności w globalnym układzie współrzędnych dla prętów oznaczonych na rysunku nr 1 numerami "1,2" łączących kolejno węzły I-II oraz I-IV. Rozmiar tych macierzy jest równy dwukrotnej ilości prętów kratownicy. W omawianym przypadku będzie to $2 * 5 = 10$, otrzymamy finalnie macierze sztywności rozmiaru 10x10. Wszystkie miejsca macierzy pominięte poniżej wypełnione są zerami.

Dla pręta numer 1:

$$[K^{\circ}_{A-B}] = 10^7 \cdot \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 6.28 & 0 & -6.28 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & -6.28 & 0 & 6.28 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{6}$$

Dla pręta numer 2:

$$[K^{\circ}_{A-D}] = 10^7 \cdot \begin{bmatrix} 6.28 & 0 & 0 & 0 & 0 & 0 & -6.28 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ -6.28 & 0 & 0 & 0 & 0 & 0 & 6.28 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & \end{bmatrix} \tag{7}$$

3.3. Zagregowana macierz sztywności w globalnym układzie współrzędnych

Poniżej przedstawiona została macierz sztywności w globalnym układzie współrzędnych po agregacji macierzy sztywności poszczególnych elementów kratownicy. Jej rozmiar jest analogiczny do reszty macierzy sztywności.

$$[K^{\circ}] = 10^7 \cdot \begin{bmatrix} 6.28 & 0 & 0 & 0 & 0 & 0 & -6.28 & 0 & 0 & 0 \\ 0 & 6.28 & 0 & -6.28 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8.5046 & -2.2203 & -6.28 & 0 & -2.2203 & 2.2203 & 0 & 0 \\ 0 & -6.28 & -2.2203 & 8.5046 & 0 & 0 & 2.2203 & -2.2203 & 0 & 0 \\ 0 & 0 & -6.28 & 0 & 8.5046 & -2.2203 & 0 & 0 & -2.2203 & 2.2203 \\ 0 & 0 & 0 & 0 & -2.2203 & 8.5046 & 0 & -6.28 & 2.2203 & -2.2203 \\ -6.28 & 0 & -2.2203 & 2.2203 & 0 & 0 & 1.4788 & -2.2203 & -6.28 & 0 \\ 0 & 0 & -2.2203 & -2.2203 & 0 & -6.28 & -2.2214 & 8.5046 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2.2203 & 2.2203 & -6.28 & 0 & 8.5046 & -2.2203 \\ 0 & 0 & 0 & 0 & 2.2203 & -2.2203 & 0 & 0 & -2.2203 & 2.2203 \end{bmatrix} \tag{8}$$

3.4. Macierze bezwładności w lokalnym układzie współrzędnych

Poniżej przedstawione zostały wybrane macierze bezwładności w lokalnym układzie współrzędnych dla prętów oznaczonych na rysunku nr 1 numerami "1,2,3" łączących kolejno węzły I-II, I-IV oraz II-IV.

Dla pręta numer 1:

$$[m_{1-2}] = \begin{bmatrix} 0.8227 & 0 & 0.4113 & 0 \\ 0 & 0.8227 & 0 & 0.4113 \\ 0.4113 & 0 & 0.8227 & 0 \\ 0 & 0.4113 & 0 & 0.8227 \end{bmatrix} \tag{9}$$

Dla pręta numer 2:

$$[m_{1-4}] = \begin{bmatrix} 0.8227 & 0 & 0.4113 & 0 \\ 0 & 0.8227 & 0 & 0.4113 \\ 0.4113 & 0 & 0.8227 & 0 \\ 0 & 0.4113 & 0 & 0.8227 \end{bmatrix} \tag{10}$$

Dla pręta numer 3:

$$[m_{2-4}] = \begin{bmatrix} 1.1634 & 0 & 0.5817 & 0 \\ 0 & 1.1634 & 0 & 0.5817 \\ 0.5817 & 0 & 1.1634 & 0 \\ 0 & 0.5817 & 0 & 1.1634 \end{bmatrix} \tag{11}$$

Macierze bezwładności dla pozostały prętów zostały wyznaczone analogicznie, możliwy jest ich podgląd przy pomocy dołączonego kodu programu Python.

W kolejnym korku, macierze wyznaczone dla wszystkich prętów zostały agregowane do jednej dużej macierzy znajdującej się w nowym, głównym układzie współrzędnych.

3.5. Macierze bezwładności w globalnym układzie współrzędnych

Poniżej przedstawione zostały wybrane macierze bezwładności w globalnym układzie współrzędnych dla prętów oznaczonych na rysunku nr 1 numerami "1,2" łączących kolejno węzły I-II oraz I-IV. Rozmiar tych macierzy jest równy dwukrotnej ilości węzłów kratownicy. W omawianym przypadku będzie to $2*5 = 10$, otrzymamy finalnie macierze bezwładności rozmiaru 10x10. Wszystkie miejsca macierzy pominięte poniżej wypełnione są zerami.

Dla pręta numer 1:

$$[m^{\circ}_{1-2}] = \begin{bmatrix} 0.8227 & 0 & 0.4113 & 0 & \dots & 0 \\ 0 & 0.8227 & 0 & 0.4113 & \dots & 0 \\ 0.4113 & 0 & 0.8227 & 0 & \dots & 0 \\ 0 & 0.4113 & 0 & 0.8227 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{12}$$

Dla pręta numer 2:

$$[m^{\circ}_{1-4}] = \begin{bmatrix} 0.8227 & 0 & 0 & 0 & 0 & 0 & 0.4113 & 0 & 0 & 0 \\ 0 & 0.8227 & 0 & 0 & 0 & 0 & 0 & 0.4113 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.4113 & 0 & 0 & 0 & 0 & 0 & 0.8227 & 0 & 0 & 0 \\ 0 & 0.4113 & 0 & 0 & 0 & 0 & 0 & 0.8227 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{13}$$

3.6. Zagregowana macierz bezwładności w globalnym układzie współrzędnych

Poniżej przedstawiona została macierz bezwładności w globalnym układzie współrzędnych po agregacji macierzy bezwładności poszczególnych elementów kratownicy. Jej rozmiar jest analogiczny do reszty macierzy bezwładności.

$$[m^{\circ}] = \begin{bmatrix} 1.6454 & 0 & 0.4113 & 0 & 0 & 0 & 0.4113 & 0 & 0 & 0 \\ 0 & 1.6454 & 0 & 0.4113 & 0 & 0 & 0 & 0.4113 & 0 & 0 \\ 0.4113 & 0 & 2.8088 & 0 & 0.4113 & 0 & 0.5817 & 0 & 0 & 0 \\ 0 & 0.4113 & 0 & 2.8088 & 0 & 0.4113 & 0 & 0.5817 & 0 & 0 \\ 0 & 0 & 0.4113 & 0 & 2.8088 & 0 & 0.4113 & 0 & 0.5817 & 0 \\ 0 & 0 & 0 & 0.4113 & 0 & 2.8088 & 0 & 0.4113 & 0 & 0.5817 \\ 0.4113 & 0 & 0.5817 & 0 & 0.4113 & 0 & 3.6315 & 0 & 0.4113 & 0 \\ 0 & 0.4113 & 0 & 0.5817 & 0 & 0.4113 & 0 & 3.6315 & 0 & 0.4113 \\ 0 & 0 & 0 & 0 & 0.5817 & 0 & 0.4113 & 0 & 1.9861 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5817 & 0 & 0.4113 & 0 & 1.9861 \end{bmatrix} \tag{14}$$

4. Macierze po agregacji i nadaniu warunków brzegowych

4.1. Macierz sztywności po agregacji i nadaniu warunków brzegowych

$$[K^{\circ}] = 10^7 \cdot \begin{bmatrix} 10^{-7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6.2800 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-7} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8.5003 & 2.2203 & 0 & 0 & -2.2203 & -2.2203 \\ 0 & 0 & 0 & 0 & 2.2203 & 8.5003 & 0 & -6.2800 & -2.2203 & -2.2203 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1.4780 & 2.2203 & -6.2800 & 0 \\ 0 & 0 & 0 & 0 & 0 & -6.2800 & 2.2203 & 8.5003 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2.2203 & -2.2203 & -6.2800 & 0 & 8.5003 & 2.2203 \\ 0 & 0 & 0 & 0 & -2.2203 & -2.2203 & 0 & 0 & 2.2203 & 2.2203 \end{bmatrix} \tag{15}$$

4.2. Macierz bezwładności po agregacji i nadaniu warunków brzegowych

$$[m^{\circ}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.6454 & 0 & 0 & 0 & 0 & 0 & 0.4113 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2.8088 & 0 & 0.4113 & 0 & 0.5817 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.8088 & 0 & 0.4113 & 0 & 0.5817 \\ 0 & 0 & 0 & 0 & 0.4113 & 0 & 3.6315 & 0 & 0.4113 & 0 \\ 0 & 0.4113 & 0 & 0 & & 0.4113 & 0 & 3.6315 & 0 & 0.4113 \\ 0 & 0 & 0 & 0 & 0.5817 & 0 & 0.4113 & 0 & 1.9861 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5817 & 0 & 0.4113 & 0 & 1.9861 \end{bmatrix} \tag{16}$$

5. Wektor obciążeń

Jako obciążenie przyjęto dwie siły składowe o wartości $100[kN]$ przyłożone do punktu V rozchodzące się na kierunkach X oraz Y. Dają one siłę wynikową o wartości $141,4[kN]$ skierowaną pod kątem 45° od poziomu. Poniżej przedstawiony został wektor sił którego długość odpowiada dwukrotnej ilości węzłów.

$$F = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 100000 \\ -100000 \end{bmatrix} [kN] \tag{17}$$

6. Wektor przemieszczeń węzłowych

Wektor przemieszczeń węzłowych będący wynikiem analizy statycznej kratownicy, został otrzymany przy użyciu opracowanego kodu programu Python. Przemieszczenia poszczególnych węzłów kształtują się następująco:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0.0016 \\ -0.0077 \\ -0.0016 \\ -0.0061 \\ -0.0016 \\ -0.0154 \end{bmatrix} [m] \tag{18}$$

7. Macierze wektorów własnych i częstotliwości drgań własnych

Jednym z wyników analizy dynamicznej kratownicy są wartości częstotliwości drgań własnych. Poniżej przedstawiony został wektor częstotliwości po uprzednim sortowaniu oraz odcięciu wartości wynikających z nałożonych warunków brzegowych.

$$[f] = \begin{bmatrix} 159.4846 \\ 496.4091 \\ 550.1881 \\ 920.5787 \\ 1011.4144 \\ 1253.8646 \\ 1516.6375 \end{bmatrix} [Hz] \tag{19}$$

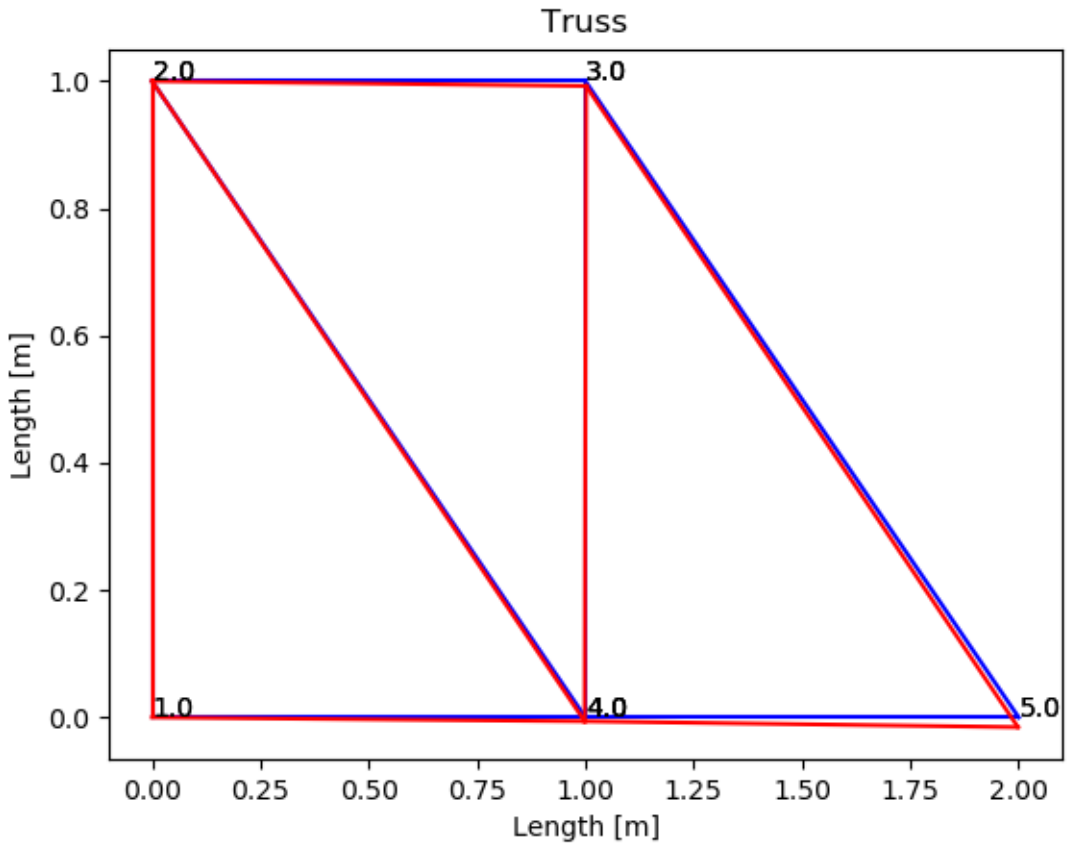
Macierz wektorów własnych przedstawia się następująco:

$$[u] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1,0000 & 0 & 0 \\ -0,0222 & 0,3014 & 0,8825 & 0,5734 & 0,0027 & -0,0322 & 0,0298 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1,0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1,0000 \\ 0,3392 & -0,1988 & 0,3085 & -0,5844 & -0,0298 & -0,1112 & 0,5514 & 0 & 0 & 0 \\ -0,301 & 0,6094 & -0,0079 & -0,316 & 0,4631 & -0,3919 & 0,0821 & 0 & 0 & 0 \\ 0,3466 & 0,4319 & -0,2323 & 0,2985 & 0,1391 & 0,2342 & 0,3568 & 0 & 0 & 0 \\ 0,0515 & -0,4642 & -0,1938 & 0,323 & 0,4017 & -0,3766 & 0,2614 & 0 & 0 & 0 \\ -0,7676 & -0,2052 & -0,0137 & 0,1139 & 0,1763 & 0,5262 & 0,4244 & 0 & 0 & 0 \\ 0,286 & -0,2329 & 0,1851 & -0,1533 & 0,7569 & 0,5996 & -0,5591 & 0 & 0 & 0 \end{bmatrix} \tag{20}$$

8. Analiza statyczna - program Python

Na rysunku nr 2 przedstawiona została kolorem niebieskim badana kratownica wraz z opisanymi węzłami. Kolorem czerwonym wyrysowana została kratownica po doznaniu przemieszczeń statycznych wywołanych przyłożoną siłą przedstawioną w rozdziale nr 6. Przedstawiony wykres został otrzymany przy pomocy kodu opracowanego w języku Python dodanego jako załącznik do sprawozdania.

Po napisaniu kodu w języku Python, wyrysowana została badana kratownica. Kratownica w kolorze niebieskim oznacza kratownice przed odkształceniem, a kolor czerwony pokazuje kratownice po odkształceniu.



Rysunek 2: Statyczne odkształcenie kratownicy w Pythonie

9. Analiza statyczna - Ansys

9.1. Analiza w programie ANSYS

Przy pomocy programu Ansys została przeprowadzona analiza, która pozwoli na weryfikacje wyników otrzymanych drogą programowania. Niezbędnym do poprawnej analizy krokiem było wybranie rodzaju badanych elementów jako elementy prętowe, tak samo jak zostały one potraktowane w kodzie programu Python. Wybór elementu został zrealizowany przy pomocy poniższej komendy:

```
*get,myarea,SECP,matid,PROP,AREA
ET,matid,180
SECTYPE,matid,LINK
SECDATA,myarea
```

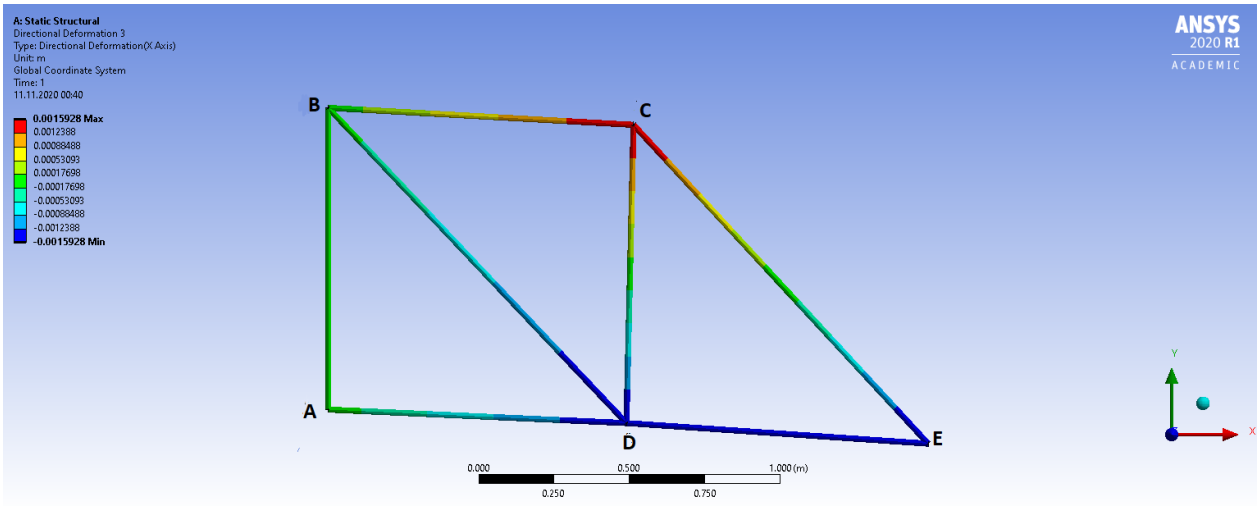
9.2. Wyniki analizy

Wartości otrzymanych przemieszczeń zostały zebrane w tabeli nr 3. Rysunek nr 3 przedstawia wykres przemieszczeń węzłowych kratownicy.

Otrzymane przemieszczenia zebrano w poniższej tabeli, a schemat odkształceń przedstawiono na rysunku.

Tabela 3: Przemieszczenia węzłów na podstawie programu ANSYS

Węzeł	**Przemieszczenie na osi X [m]**	**Przemieszczenie na osi Y [m]**
A	0	0
B	0	0
C	$1.59 \cdot 10^{-3}$	$-7.7 \cdot 10^{-3}$
D	$-1.59 \cdot 10^{-3}$	$-6.1 \cdot 10^{-3}$
E	$-1.59 \cdot 10^{-3}$	$-1.53 \cdot 10^{-2}$



Rysunek 3: Schemat projektowanej kratownicy

10. Analiza dynamiczna

10.1. Wartości częstotliwości drgań własnych otrzymane przy pomocy Ansysa oraz kodu Python

Tabela 4: Porównanie częstotliwości drgań własnych w programie ANSYS i Python

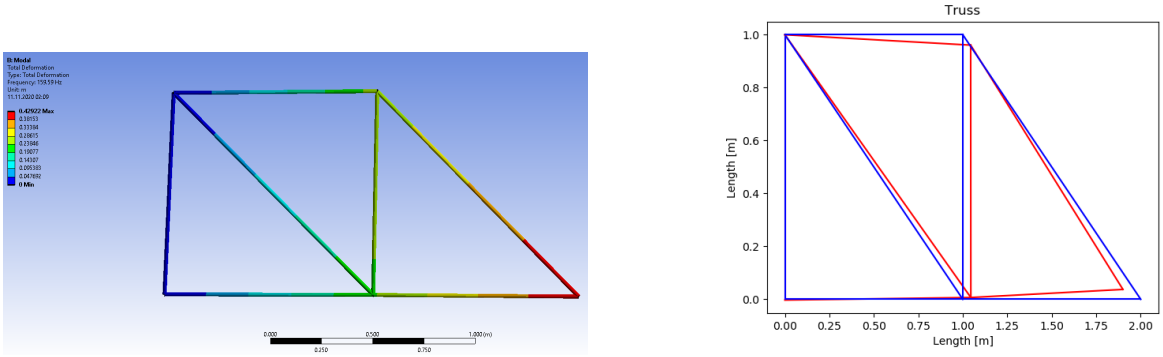
Numer modalnej	ANSYS [Hz]	Python [Hz]	Błąd [%]
4	159.59	159.4846	nieistotny
5	496.73	496.4091	nieistotny
6	550.54	550.1881	nieistotny
7	921.16	920.5787	nieistotny
8	1012.1	1011.4144	nieistotny
9	1254.7	1253.8646	nieistotny
10	1517.6	1516.6375	nieistotny

Analizując tabelę nr 4 można stwierdzić, że otrzymane wartości są niezwykle do siebie zbliżone, a różnice pomiędzy nimi należy uznać za nieistotne. Z powodzeniem można więc stwierdzić że analizy zostały wykonane poprawnie.

10.2. Porównanie postaci drgań własnych w programie ANSYS oraz Python

Wartości przemieszczeń wynikające z drgań własnych kratownicy możemy znaleźć w kolejnych kolumnach macierzy u przedstawionej w rozdziale nr 7.

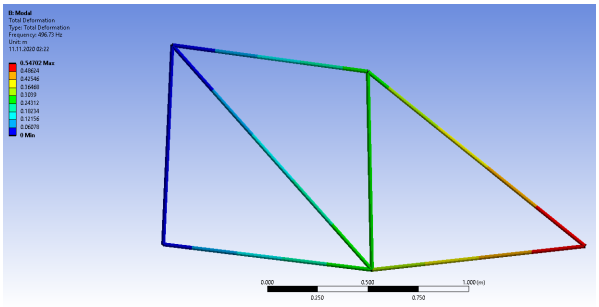
Modalna numer 4:



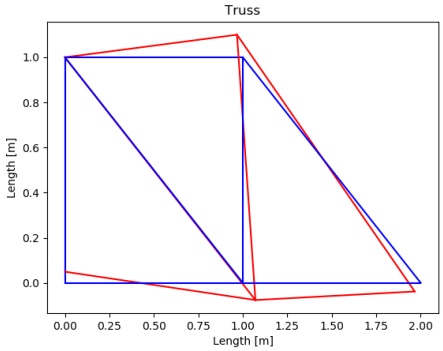
(a) ANSYS (b) Python

Rysunek 4: Porównanie postaci drgań własnych dla modalnej numer 4

Modalna numer 5:



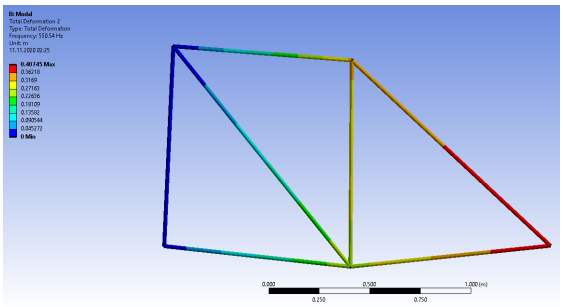
(a) ANSYS



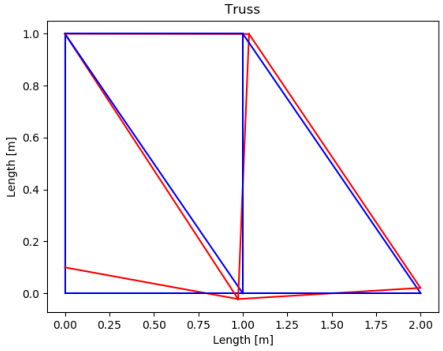
(b) Python

Rysunek 5: Porównanie postaci drgań własnych dla modalnej numer 5

Modalna numer 6:



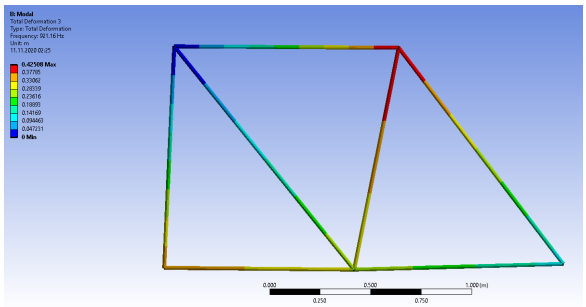
(a) ANSYS



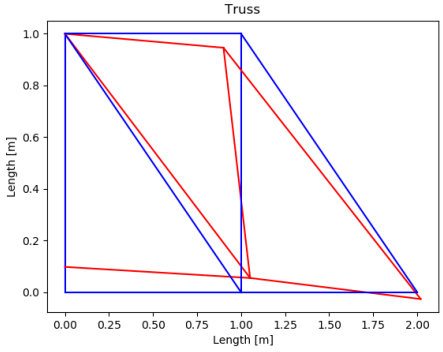
(b) Python

Rysunek 6: Porównanie postaci drgań własnych dla modalnej numer 6

Modalna numer 7:



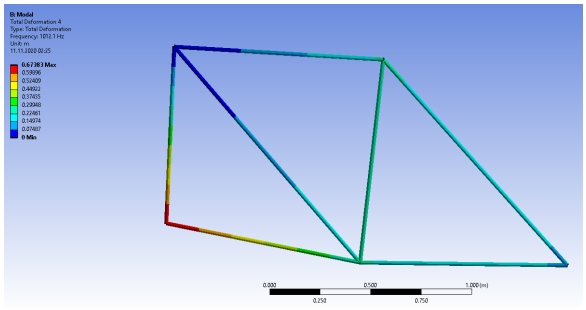
(a) ANSYS



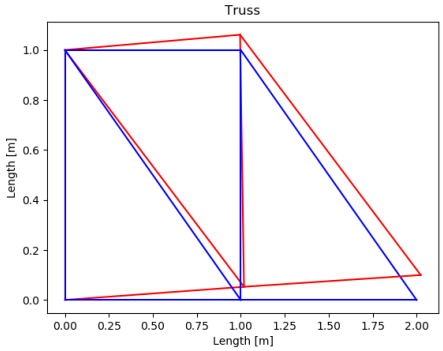
(b) Python

Rysunek 7: Porównanie postaci drgań własnych dla modalnej numer 7

Modalna numer 8:



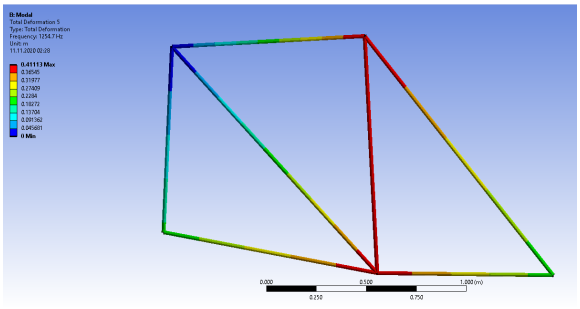
(a) ANSYS



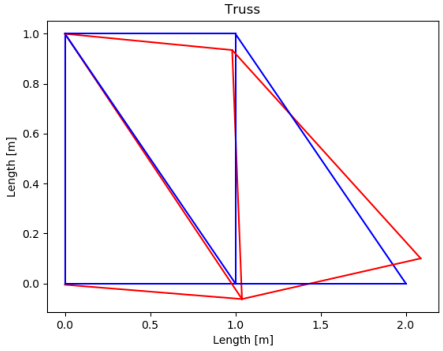
(b) Python

Rysunek 8: Porównanie postaci drgań własnych dla modalnej numer 8

Modalna numer 9:



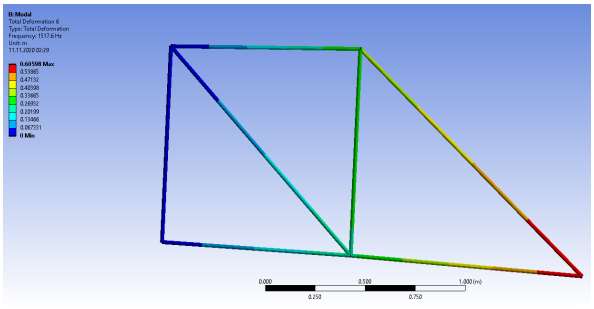
(a) ANSYS



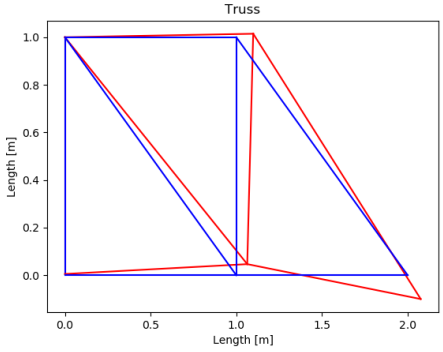
(b) Python

Rysunek 9: Porównanie postaci drgań własnych dla modalnej numer 9

Modalna numer 10:



(a) ANSYS



(b) Python

Rysunek 10: Porównanie postaci drgań własnych dla modalnej numer 10

11. Analiza uzyskanych wyników oraz wnioski

11.1. Porównanie wyników otrzymanych w programie ANSYS i Python

Tabela nr 5 zawiera zestawienie wyników przemieszczeń węzłowych badanej kratownicy otrzymanych przy pomocy programu ANSYS i Python.

Tabela 5: Porównanie wartości przemieszczeń z programów ANSYS i Python

	Przemieszczenie na osi X [m]		**Przemieszczenie na osi Y [m]**	
Węzeł	**ANSYS**	**Python**	**ANSYS**	**Python**
I	0	0	0	0
II	0	0	0	0
III	$1.59 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$	$-7.7 \cdot 10^{-3}$	$-7.7 \cdot 10^{-3}$
IV	$-1.59 \cdot 10^{-3}$	$-1.6 \cdot 10^{-3}$	$-6.1 \cdot 10^{-3}$	$-6.1 \cdot 10^{-3}$
V	$-1.59 \cdot 10^{-3}$	$-1.6 \cdot 10^{-3}$	$-1.53 \cdot 10^{-2}$	$-1.54 \cdot 10^{-2}$

Porównując powyższe wyniki możemy jednoznacznie stwierdzić, że różnice między nimi są niezwykle małe i wynikające z błędów przybliżeń. Podobieństwo wyników pozwala traktować obie analizy jako analogiczne i poprawnie przeprowadzone.

11.2. Analiza wyników i wnioski

Celem zadania projektowego było opracowanie programu przy pomocy dowolnego narzędzia pozwalającego na przeprowadzenie analizy statycznej oraz dynamicznej dowolnej wybranej kratownicy.

Do jego rozwiązania zostało wybrane narzędzie jakim jest język programowania Python. Użytkownik opisuje wybraną kratownicę podając współrzędne węzłów oraz zależności pomiędzy nimi w pliku zewnętrznym. Kolejne dwa pliki .txt przeznaczone są kolejno do podania wektora obciążenia oraz wprowadzenia warunków brzegowych czyli rodzaju utwierdzeń kratownicy. Ponadto użytkownik może także określić gęstość materiału, moduł Younga oraz pole przekroju poprzecznego pręta. Wszystkie pręty kratownicy traktowane sa jako elementy prętowe co stanowi istotne założenie analizy.

Potwierdzeniem poprawności opracowania programu miało być wykonanie analogicznej analizy, na tej samej kratownicy przy pomocy oprogramowania Ansys. Otrzymane wyniki dla analizy statycznej jak i dynamicznej okazały się niezwykle zbliżone. Różnice które występują mogą wynikać z błędów zaokrągleń i należy je uznać za pomijalne. Pewne różnice na porównaniu wykresów znajdujące się w rozdziale 10.2 mogą płynąć z faktu, iż w Ansysie mamy do czynienia z analizą 3D, gdy w Pythonie zostało to ograniczone do analizy 2D wzdłuż osi XY.

Wykorzystana do powyższej analizy metoda elementów skończonych może być z powodzeniem implementowana do środowiska programistycznego i dawać niezwykle satysfakcjonujące wyniki dotyczące rozwiązywania problemów inżynierskich. Jej możliwości są znacznie większe niż krótkie wprowadzenie teoretyczne przedstawione w rozdziale nr 1, może być ona z powodzeniem wykorzystywana także np do analiz przepływowych.

12. Załącznik nr 1

```
import numpy as np
import math
import matplotlib.pyplot as plt
import scipy.linalg as la

#Material parameters
global E, A, ro
A = 3.14e-4
E = 2 * (10 ** 11)
ro = 7860

#Files
pointsfile = 'points.txt'
forcefile = 'force.txt'
dirichletsfile = 'xyz.txt'

#Set print options

#np.set_printoptions(formatter={'float': "{0:0.0e}".format})
np.set_printoptions(precision=4)

#Import from file

elements = np.loadtxt(pointsfile)
elements_base = elements
#Looking for points number

maxvalue = 0
for element in elements:
if maxvalue < element[5]:
maxvalue = element[5]

elif maxvalue < element[6]:
maxvalue = element[6]

#Looking for elements number

lelements = len(elements)

#Cos, sin calculation

def ssin(x):
l = math.sqrt((x[3]-x[1])**2+(x[4]-x[2])**2)
ssin = (x[4]-x[2])/l
return ssin

def ccos(x):
```

```
l = math.sqrt((x[3] - x[1]) ** 2 + (x[4] - x[2]) ** 2)
ccos = (x[3]-x[1])/l
return ccos

#Cosinus direction matrix

def DC(ccos, ssin):
DC = np.array([[ccos, ssin, 0, 0], [(-ssin), ccos, 0, 0],
[0, 0, ccos, ssin], [0, 0, (-ssin), ccos]])
return DC

#Drawing the main truss

for i in range(lelements):
blank = []
plt.plot([elements[i][1], elements[i][3]], [elements[i][2], elements[i][4]], 'b')
plt.annotate(elements[i][5], (elements[i][1], elements[i][2]))
plt.annotate(elements[i][6], (elements[i][3], elements[i][4]))
i = i+1

#Stiffness matrix

#Statics

def matrixk():
k = np.array([[1, 0, -1, 0],
[0, 0, 0, 0],
[-1, 0, 1, 0],
[0, 0, 0, 0]])
return k


def matrixkprim(DC, k, x):
l = math.sqrt((x[3]-x[1])**2+(x[4]-x[2])**2)
matrixkprim = ((A*E)/l)*(np.dot(np.dot(DC.T, k), DC))
return matrixkprim

k1 = matrixk()
#Dynamics

def matrixm():
m = np.array([[2, 0, 1, 0],
[0, 2, 0, 1],
[1, 0, 2, 0],
[0, 1, 0, 2]])
return m

def matrixmprim(DC, m, x):
l = math.sqrt((x[3]-x[1])**2+(x[4]-x[2])**2)
matrixmprim = ((ro*A*l)/6)*(np.dot(np.dot(DC.T, m), DC))
```

```
return matrixmprim

m1 = matrixm()

#Stiffness matrix verification

def spr(x, ssin, ccos):
l = math.sqrt((x[3]-x[1])**2+(x[4]-x[2])**2)
spr = ((A*E)/l)*np.array([[ccos*ccos, ssin*ccos, -ccos*ccos, -ssin*ccos],
[ssin*ccos, ssin*ssin, -ssin*ccos, -ssin*ssin],
[-ccos*ccos, -ssin*ccos, ccos*ccos, ssin*ccos],
[-ssin*ccos, -ssin*ssin, ssin*ccos, ssin*ssin]])
return spr

#Inertia matrix after agregation
#Z1 - inertia matrix - statics
#M_ini - inertia matrix - dynamics

Z1 = np.zeros((2*int(maxvalue), 2*int(maxvalue)))
M_ini = np.zeros((2*int(maxvalue), 2*int(maxvalue)))

for element in elements:
Z = np.zeros((2*int(maxvalue), 2*int(maxvalue)))
M = matrixkprim((DC(ccos(element), ssin(element))), k1, element)


m = int(element[5])
n = int(element[6])
#print(m, n)

m = 2 * m - 1
n = 2 * n - 1
Z[m - 1][m - 1] = M[0][0]
Z[m - 1][m] = M[1][0]
Z[m][m - 1] = M[0][1]
Z[m][m] = M[1][1]

Z[n - 1][n - 1] = M[2][2]
Z[n - 1][n] = M[3][2]
Z[n][n - 1] = M[2][3]
Z[n][n] = M[3][3]

Z[m - 1][n - 1] = M[0][2]
Z[m - 1][n] = M[1][2]
Z[m][n - 1] = M[0][3]
Z[m][n] = M[1][3]

Z[n - 1][m - 1] = M[2][0]
Z[n - 1][m] = M[3][0]
Z[n][m - 1] = M[2][1]
```

```
Z[n][m] = M[3][1]
Z1 = Z1 + Z

# Dynamics ----------------------------------------

Z_P2 = np.zeros((2 * int(maxvalue), 2 * int(maxvalue)))
M_P2 = matrixmprim((DC(ccos(element), ssin(element))), m1, element)

Z_P2[m - 1][m - 1] = M_P2[0][0]
Z_P2[m - 1][m] = M_P2[1][0]
Z_P2[m][m - 1] = M_P2[0][1]
Z_P2[m][m] = M_P2[1][1]

Z_P2[n - 1][n - 1] = M_P2[2][2]
Z_P2[n - 1][n] = M_P2[3][2]
Z_P2[n][n - 1] = M_P2[2][3]
Z_P2[n][n] = M_P2[3][3]

Z_P2[m - 1][n - 1] = M_P2[0][2]
Z_P2[m - 1][n] = M_P2[1][2]
Z_P2[m][n - 1] = M_P2[0][3]
Z_P2[m][n] = M_P2[1][3]

Z_P2[n - 1][m - 1] = M_P2[2][0]
Z_P2[n - 1][m] = M_P2[3][0]
Z_P2[n][m - 1] = M_P2[2][1]
Z_P2[n][m] = M_P2[3][1]
M_ini = M_ini + Z_P2


M_ini = np.where(M_ini < (1*10**(-15)), 0, M_ini)
#Force vector - length have to be 2*number of points

#Force manually
'''
F = np.zeros((10, 1))
F[8][0] = 1000000
F[9][0] = -1000000
'''


#Import force from file

F = np.loadtxt(forcefile)
length_F = len(F)
F = F.reshape(length_F, 1)

#Dirichlet's boundary conditions from file


P = np.loadtxt(dirichletsfile, dtype='str')
```

```
i = 0
L = []

for i in range(len(P)):
#print(P[i][0])
if P[i][1] == 'n':
L.append(2*int(P[i][0])-2)
L.append(2*int(P[i][0])-1)
elif P[i][1] == 'p' and P[i][2] == 'x':
L.append(2 * int(P[i][0]) - 2)
elif P[i][1] == 'p' and P[i][2] == 'y':
L.append(2 * int(P[i][0]) - 1)


i = 0
f = 0

for i in range(len(Z1)):
for f in L:
Z1[:][f] = 0
Z1[i][f] = 0
Z1[f][f] = 1

#Dynamics

M_ini[:][f] = 0
M_ini[i][f] = 0
M_ini[f][f] = 1




#Calculating points displacements

U = np.dot(np.linalg.inv(Z1), F)

#Creating new points after displacements
i = 0
for element in elements:
j = int(element[5])
j = 2*j -1
elements[i][1] = elements[i][1] + U[j - 1][0]
#print(elements[i][1], U[j - 1][0])
elements[i][2] = elements[i][2] + U[j][0]
k = int(element[6])
k = 2*k -1
elements[i][3] = elements[i][3] + U[k - 1][0]
elements[i][4] = elements[i][4] + U[k][0]
```

```
i = i + 1



#Drawing truss after displacements

i = 0
for i in range(lelements):
plt.plot([elements[i][1], elements[i][3]],
[elements[i][2], elements[i][4]], 'r')
i = i+1

plt.title('Truss')
plt.xlabel('Length [m]')
plt.ylabel('Length [m]')
#plt.show()

#Creating eigenvector and eigenmatrix

#w, u = la.eig(Z1, M_ini)
w, u = la.eig(np.dot(np.linalg.inv(M_ini),Z1))



#Deleting values connected with dirichlet's conditions
w = np.real(w)
w = np.delete(w, np.where(w == 1))
#Vibration frequency

f_hz = 1/(2*np.pi)*np.sqrt(w)

f_hz = np.sort(f_hz)

b = []
u = u.T
for column in u:
max_value = max(np.abs(column))
column = column/max_value
b.append(column)


b = np.array(b)
b = b.T
u = b*0.1


elements = np.loadtxt(pointsfile)
z = 0
j = 0
k = 0
```

```
for z in range(len(u)-3):
x = u[:len(u), z:z+1]
i = 0
elements = np.loadtxt(pointsfile)
elements2 = np.loadtxt(pointsfile)
for element in elements:

j = int(element[5])
j = 2*j -1
elements[i][1] = elements[i][1] + x[j - 1][0]
#print(elements[i][1], U[j - 1][0])
elements[i][2] = elements[i][2] + x[j][0]
k = int(element[6])
k = 2*k -1
elements[i][3] = elements[i][3] + x[k - 1][0]
elements[i][4] = elements[i][4] + x[k][0]
i = i +1

h = 0
for h in range(lelements):
plt.plot([elements[h][1], elements[h][3]],
[elements[h][2], elements[h][4]], 'r')
plt.plot([elements2[h][1], elements2[h][3]],
[elements2[h][2], elements2[h][4]], 'b')

plt.title('Truss')
plt.xlabel('Length [m]')
plt.ylabel('Length [m]')
plt.show()
```

Literatura

[1] Czajka I., Gołaś A., *Inżynierskie metody analizy numerycznej i planowanie eksperymentu*, Wydawnictwa AGH, Kraków 2017