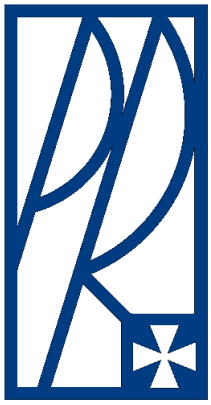


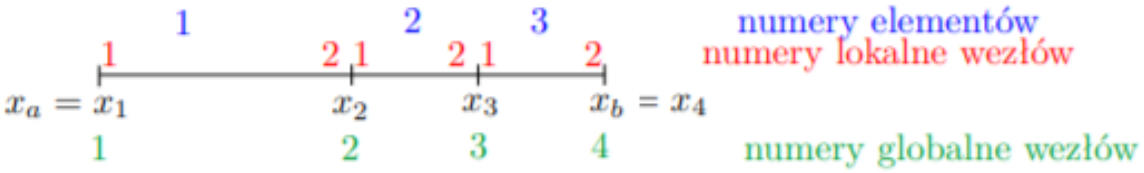
POLITECHNIKA
RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

Metody FEM w robotyce
Laboratoria

Kacper Żurawski
EA-DI
160867

1. Wstęp

Celem ćwiczenia jest zrealizowanie maksymalnie uproszczonego zagadnienia, którego geometria jest przedstawiona na rysunku 1. Jest to odcinek o długości *l* podzielony na *n* elementów o identycznej długości. Każdy z elementów składa się z dwóch węzłów po sobie następujących w kolejności od lewej do prawej. Na krańcowych węzłach występują warunki brzegowe Dirichleta.



Rysunek 1 – Przykładowa geometria wykorzystana w zagadnieniu

2. Spis skryptów Python

Projekt składa się z wielu plików, których zadaniem jest specyficzne zadanie - jest to wykonane w ten sposób w celu ułatwienia edytowania skryptów oraz modułowości projektu.

Main – główny plik wykonujący oraz wywołujący resztę funkcji
ManualGeometry – umożliwia zadeklarowanie geometrii przez użytkownika
AutomaticGeometry – automatyczne generowanie geometrii opartej o podane parametry
DrawGeometry – wizualizacja geometrii w formacie podobnym do tego z Rysunku 1
MemoryAllocation – alokacja pamięci do zmiennych globalnych
Aij – definicja funkcji podcałkowej
BaseFunctions - definicja funkcji bazowych
PlotResult – wizualizacja wyniku uproszczonego zagadnienia

3. Szczegółowy opis poszczególnych plików

3.1) ManualGeometry oraz AutomaticGeometry

Pierwszym krokiem w zrealizowaniu tego zagadnienia jest deklaracja geometrii, która zostanie przeanalizowana. W celu jej zdefiniowania potrzebne są dwie tablice – węzłów oraz elementów. Każdy z węzłów posiada własny indeks oraz współrzędną na osi X, a każdy z elementów zawiera własny indeks, globalny indeks początkowego węzła oraz globalny indeks końcowego węzła.
Poniższe tabele przedstawiają format danych oraz przykładowe ich wypełnienie.

Globalny indeks węzła	Współrzędna węzła na osi X
1	x_1
2	x_2
…..	…..
n	x_n

Tabela 1 – Format tabeli Węzły

Globalny indeks elementu	Globalny indeks początkowego węzła	Globalny indeks końcowego węzła
1	w_1	w_2
2	w_2	w_3
…..	….	…..
n-1	w_{n-1}	w_n

Tabela 2 – Format tabeli Elementy

Globalny indeks węzła	Współrzędna węzła na osi X
1	0
2	0.(3)
3	0.(6)
4	1

Tabela 3 – Przykładowe dane w tabeli Węzły

Globalny indeks elementu	Globalny indeks początkowego węzła	Globalny indeks końcowego węzła
1	1	2
2	2	3
3	3	4

Tabela 4 – Przykładowe dane w tabeli Elementy

W celu zdefiniowania geometrii wymagane są też dane o warunkach brzegowych, są one zapisywane w formacie słownika o poniższych właściwościach.

Globalny indeks węzła	Typ warunku brzegowego	Wartość warunku brzegowego
w_1	D	y_1
w_n	D	y_2

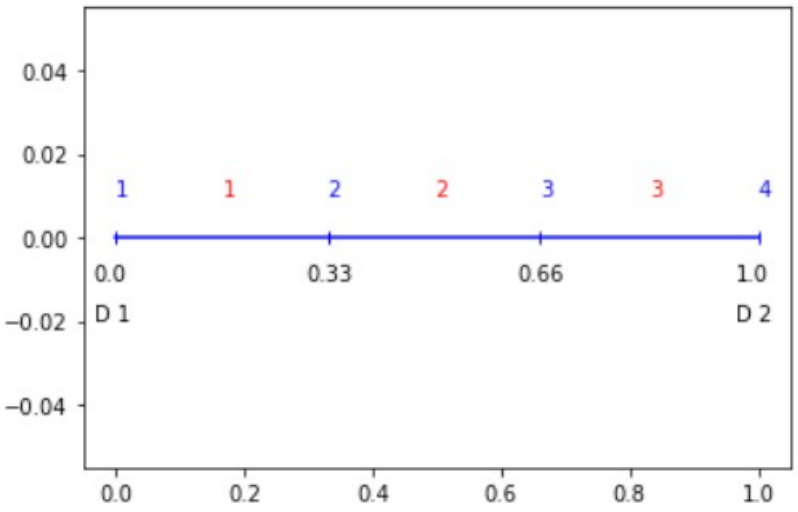
Tabela 5 – Format tabeli WarunkiBrzegowe

Zawiera ona dwa wiersze składające się z globalnego indeksu węzła, typu warunku brzegowego (Dirichleta) oraz wartości warunku brzegowego. Poniższa tabela ukazuje przykładowe wypełnienie danych warunków brzegowych.

Globalny indeks węzła	Typ warunku brzegowego	Wartość warunku brzegowego
4	D	1
1	D	2

Tabela 6 – Przykładowe dane w tabeli WarunkiBrzegowe

Dzięki powyższym danym możliwa jest wizualizacja geometrii (opisane w punkcie 3.2).



Rysunek 2 – Wizualizacja przykładowej geometrii

Występują dwie możliwości w generowaniu geometrii – podanie danych w tabelach przez użytkownika (ManualGeometry) lub automatyczna generacja (AutomaticGeometry).
Funkcja ManualGeometry zawiera definicję tabel 1,2 oraz 5 zawierających dane o węzłach, elementach oraz warunkach brzegowych. W celu zmiany zmiany geometrii wystarczy zmienić wartości w zdefiniowanych tabelach.

```
import numpy as np
def ManualGeometry():
    #array of nodes 1.index 2.position
    Nodes = np.array([[1, 0],
                      [2, 0.33],
                      [3, 0.66],
                      [4, 1]] )

    #array of elements 1.index 2.index of first nodes 3.index of second node
    Elements = np.array( [[1, 1, 2],
                          [2, 2, 3],
                          [3, 3, 4]] )

    #dictionary of edge conditions 1.1 index 1.2 type 1.3 value 2.1 index ...
    EdgeConditions    = [{"Index": 1, "Type":'D', "Value":1},
                         {"Index": 4, "Type":'D', "Value":2}]


    return Nodes, Elements, EdgeConditions
```

Tekst 1 – Funkcja ManualGeometry

W celu wykorzystania funkcji AutomaticGeometry wymagane są trzy parametry – początkowa współrzędna na osi X, końcowa współrzędna na osi X oraz ilość węzłów. Węzły rozmieszczane są w tej samej odległości od sąsiadujących węzłów poprzez wykorzystanie funkcji *linspace* do wygenerowania współrzędnych.
Indeksy węzłów oraz indeksy elementów generowane są z wykorzystaniem funkcji *arange*.
W celu utworzenia tablicy węzłów należy połączyć dwie tablice – indeksy węzłów (Indexes) oraz ich współrzędne (Xpositions) funkcją *vstack*.
Tworzenie tablicy elementów jest zrealizowane podobnie do tworzenia tablicy węzłów z jedną różnicą – wykorzystana jest funkcja *block* zamiast *vstack*.

```
import numpy as np

def AutomaticGeometry(XPosStart,XPosEnd,NumberOfNodes):
   #array of nodes indexes
   Indexes = np.arange(1,NumberOfNodes+1)
   #array of nodes positions
   XPositions = np.linspace(XPosStart,XPosEnd,NumberOfNodes) ;
   #array of nodes 1.index 2.position
   Nodes = (np.vstack( (Indexes.T, XPositions.T) )).T

   #array of first nodes indexes
   FirstNodesArray = np.arange(1,NumberOfNodes)
   #array of second nodes indexes
   SecondNodesArray = np.arange(2,NumberOfNodes+1)
   #array of elements 1.index 2.index of first nodes 3.index of second node
   Elements = (np.block( [[Indexes[0:len(Indexes)-1]], [FirstNodesArray],
[SecondNodesArray] ] ) ).T
   EdgeConditions  = [{"Index": 1, "Type":'D', "Value":1},
                     {"Index": NumberOfNodes, "Type":'D', "Value":2}]
   return Nodes, Elements,EdgeConditions
```

Tekst 2 – Funkcja ManualGeometry

W celu wykorzystania tych funkcji wystarczy je poprawnie wywołać po zaimportowaniu wymaganych plików.

```
#MANUAL GEOMETRY
Nodes, Elements, EdgeConditions = ManualGeometry()
NumberOfNodes = np.shape(Nodes)[0]


#AUTOMATIC GEOMETRY
#XPosStart =  0
#XPosEnd =  1
#NumberOfNodes = 4
#Nodes, Elements, EdgeConditions = AutomaticGeometry(XPosStart,XPosEnd,NumberOfNodes);
```

Tekst 3 – Przykładowe wykorzystanie funkcji ManualGeometry oraz AutomaticGeometry

3.2) DrawGeometry

Funkcja ta odpowiada za utworzenie wizualizacji zdefiniowanej geometrii z wykorzystaniem biblioteki *matplotlib*, jej parametrami są tablice węzłów i elementów oraz słownik zawierający dane z warunkach brzegowych.
Pierwszym krokiem jest utworzenie nowego okienka wizualizacji – jest to zrealizowane funkcją *plt.figure()*.
Następnie tworzona jest linia łącząca kolejne węzły – funkcja *plt.plot()*, której argumentami są wartości na osi X oraz Y, w naszym przypadku będą to współrzędne węzłów oraz wartości 0.
Kolejnym krokiem jest wyświetlenie globalnych indeksów węzłów oraz ich współrzędnych – jest to zrealizowane w pętli *for*, której indeks jest wykorzystywany w celu indeksowania poszczególnych węzłów, których dane są wyświetlane za pomocą funkcji *plt.text()*, której argumentami są współrzędne na osiach X oraz Y, a także tekst który powinien zostać wyświetlony.
W podobny sposób dodane zostają informacje o elementach oraz warunkach brzegowych.

```
import numpy as np
import matplotlib.pyplot as plt

def DrawGeometry(Nodes, Elements, EdgeConditions):
    plt.figure()
    plt.plot(Nodes[:,1], np.zeros( (np.shape(Nodes)[0], 1) ), '-b|' )

    NumberOfNodes = np.shape(Nodes)[0]
    NumberOfElements = np.shape(Elements)[0]

    for CurrentNode in np.arange(0,NumberOfNodes):
        Index = Nodes[CurrentNode,0]
        XPos = Nodes[CurrentNode,1]
        plt.text(XPos, 0.01, str( int(Index) ), c="b")
        plt.text(XPos-0.035, -0.01, str(round(XPos,3)))

    for CurrentElement in np.arange(0,NumberOfElements):
        FirstNode = Elements[CurrentElement,1]
        SecondNode = Elements[CurrentElement,2]
        XPos = (Nodes[FirstNode-1,1] + Nodes[SecondNode-1,1] ) / 2
        plt.text(XPos, 0.01, str(CurrentElement+1), c="r")

    plt.text(Nodes[0,1]-0.035, -0.02, str(EdgeConditions[0]['Type']) + " " + str(EdgeConditions[0]
['Value']))
    plt.text(Nodes[NumberOfNodes-1,1]-0.035, -0.02, str(EdgeConditions[1]['Type']) + " " +
str(EdgeConditions[1]['Value']))

    plt.show()
```

Tekst 4 – Funkcja DrawGeometry

3.3) MemoryAllocation

Funkcja ta odpowiada za zarezerwowanie pamięci dla zmiennych A i b, które przechowują wyniki z agregacji macierzy lokalnej w macierzy globalnej oraz z obliczania elementów wektora podczas obliczania wyników analizy podanej geometrii.

```
import numpy as np

def MemoryAllocation(NumberOfNodes):
    A = np.zeros([NumberOfNodes,NumberOfNodes])
    b = np.zeros([NumberOfNodes,1])

    return A,b
```

Tekst 5 – Funkcja MemoryAllocation

3.4) BaseFunctions

Funkcja ta odpowiada za realizację funkcji bazowych występujących w uproszczonym zagadnieniu. W naszym przypadku nad każdym węzłem rozważane są dwie funkcje – malejąca(Funkcja 1) oraz rosnąca(Funkcja 2), których pochodnymi są funkcje Funkcja 3 oraz Funkcja 4.

$$Funkcja\,1\,(malejąca) - f_1 = -\frac{1}{2}x + \frac{1}{2}$$

$$Funkcja\,2\,(rosnąca) - f_2 = \frac{1}{2}x + \frac{1}{2}$$

$$Funkcja\,3\,(malejąca - pochodna) - f_3 = -\frac{1}{2}$$

$$Funkcja\,4\,(rosnąca - pochodna) - f_4 = \frac{1}{2}$$

```
def BaseFunctions(n):

    if n==0:
        f  = (lambda x: 1 + 0*x )
        df = (lambda x: 0*x )

    elif n == 1:

        f = (lambda x: -1/2*x + 1/2, lambda x: 0.5*x+0.5 )
        df= (lambda x: -1/2 + 0*x ,   lambda x: 0.5  + 0*x )
    else:
        raise Exception("Error")

    return f,df
```

Tekst 6 – Funkcja BaseFunctions

3.5) Aij

Funkcja ta realizuje funkcję podcałkową z Funkcji 5 – jest to iloczyn pochodnych funkcji kształtu.

$$A_{ji} = \int_{x_a}^{x_b} -\phi_i' \phi_j' \mathrm{d}x + \left(\phi_i(x_{\Gamma_u}) \frac{\mathrm{d}\phi_j(x_{\Gamma_u})}{\mathrm{d}n} + \frac{\mathrm{d}\phi_i(x_{\Gamma_u})}{\mathrm{d}n} \phi_j(x_{\Gamma_u}) \right)$$

Funkcja 5 – Funkcja Aij

```
def Aij(df_i, df_j, c, f_i, f_j):
    return lambda x: -df_i(x)*df_j(x) + c*f_i(x)*f_j(x)
```

Tekst 7 – Funkcja Aij

3.6) PlotResult

Zadaniem tej funkcji jest wizualizacja wyniku uproszczonej analizy zadanej geometrii. Podawanymi argumentami jest tablica węzłów oraz tablica wartości rozwiązania.

Pierwszym krokiem jest uzyskanie tablicy zawierającej współrzędne poszczególnych węzłów. Następnie należy utworzyć nowe okno wizualizacji korzystając z funkcji *plt.figure()* oraz wykorzystać funkcję *plt.plot()* w celu dodania wyniku analizy geometrii do wykresu.

```
import matplotlib.pyplot as plt

def PlotResult(Nodes, u):

    NodeXPositions = Nodes[:,1]

    plt.figure()
    plt.plot(NodeXPositions, u, '*-')
```

Tekst 8 – Funkcja PlotResult

3.7) Main

Pierwszym krokiem jest zaimportowanie wymaganych funkcji oraz bibliotek.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as spint
#Custom Funcions
from ManualGeometry import ManualGeometry
from AutomaticGeometry import AutomaticGeometry
from DrawGeometry import DrawGeometry
from MemoryAllocation import MemoryAllocation
from BaseFunctions import BaseFunctions
from Aij import Aij
from PlotResult import PlotResult
```

Tekst 9 – Importowanie wymaganych bibliotek oraz niezbędnych funkcji

Następnie wymagana jest definicja oraz wizualizacja geometrii, jest to zrealizowane funkcjami opisanymi w punkcie 3.1 oraz 3.2.
W celu zmiany definicji geometrii z podanej przez użytkownika do automatycznej należy usunąć trzy pierwsze linijki oraz odkomentować odpowiednie linie w celu wykorzystania funkcji AutomaticGeometry.

```
#MANUAL GEOMETRY
Nodes, Elements, EdgeConditions = ManualGeometry()
NumberOfNodes = np.shape(Nodes)[0]


#AUTOMATIC GEOMETRY
#XPosStart =  0
#XPosEnd =  1
#NumberOfNodes = 4
#Nodes, Elements, EdgeConditions = AutomaticGeometry(XPosStart,XPosEnd,NumberOfNodes);

DrawGeometry(Nodes, Elements, EdgeConditions)

A,b = MemoryAllocation(NumberOfNodes)
```

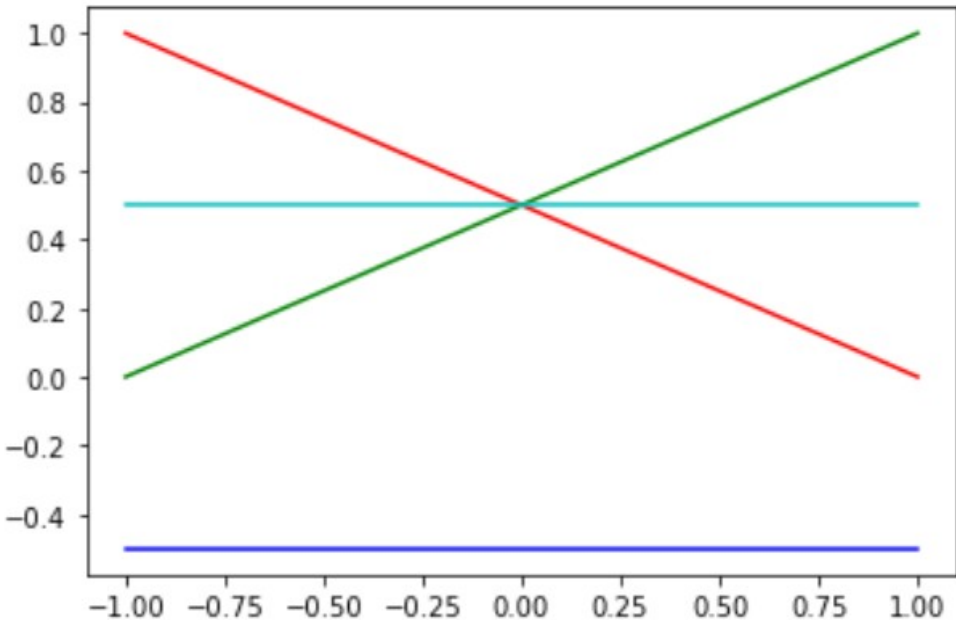
Tekst 10 – Definicja oraz wizualizacja geometrii

Następnie wymagana jest definicja funkcji bazowych opisanych w punkcie 3.4 oraz wizualizacja ich w celu sprawdzenia ich poprawności.

```
BaseFunctionDegree = 1
phi, dphi = BaseFunctions(BaseFunctionDegree)

#Checking if base functions are properly defined

plt.figure()
xx = np.linspace(-1,1, 2)
plt.plot(xx, phi[0](xx), 'r' )
plt.plot(xx, phi[1](xx), 'g' )
plt.plot(xx, dphi[0](xx), 'b' )
plt.plot(xx, dphi[1](xx), 'c' )
```

Tekst 11 – Definicja oraz wizualizacja funkcji bazowych

Jak widać na poniższym rysunku funkcje te zostały poprawnie zadeklarowane, są to funkcje : malejąca, rosnąca oraz dwie stałe (Funkcje 1-4 z punktu 3.4).



Rysunek 3 – Wizualizacja funkcji bazowych

Kolejnym korkiem jest obliczenie całek poszczególnych funkcji bazowych w każdym z elementów. Jest to realizowane poprzez wykorzystanie funkcji *Aji* oraz *spint.quad()*. Wyniki obliczeń zapisywane są w globalnej tablicy A.

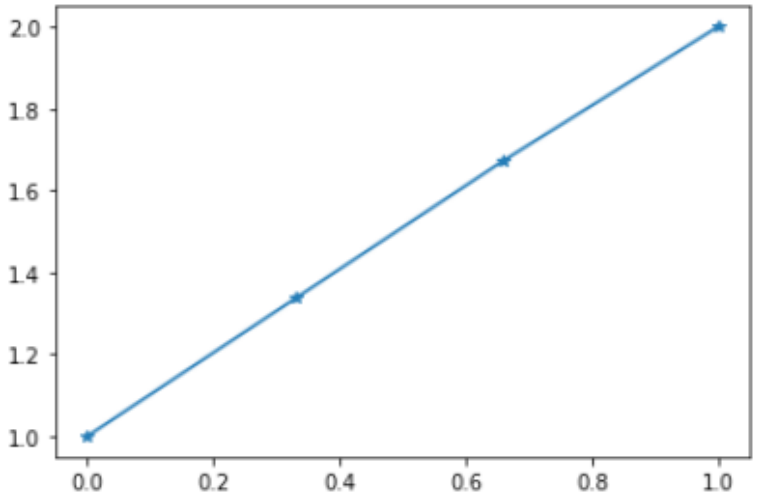
```
for CurrentElement in np.arange(0, NumberOfElements ):

    FirstNode = Elements[CurrentElement,1]     # indeks wezla poczatkowego elemntu ee
    SecondNode = Elements[CurrentElement,2]     # indeks wezla koncowego elemntu ee
    NodeGlobalIndexes = np.array([FirstNode, SecondNode ])

    XPosStart = Nodes[ FirstNode-1 ,1]
    XPosEnd = Nodes[ SecondNode-1 ,1]


    Ml = np.zeros( [BaseFunctionDegree+1, BaseFunctionDegree+1] )

    J = (XPosEnd-XPosStart)/2

    m = 0; n = 0 ;
    Ml[m,n] = J * spint.quad( Aij(dphi[m], dphi[n], c, phi[m],phi[n]), 0, 1)[0]

    m = 0; n = 1 ;
    Ml[m,n] = J * spint.quad( Aij(dphi[m], dphi[n], c, phi[m],phi[n]), 0, 1)[0]

    m = 1; n = 0 ;
    Ml[m,n] = J * spint.quad( Aij(dphi[m], dphi[n], c, phi[m],phi[n]), 0, 1)[0]

    m = 1; n = 1 ;
    Ml[m,n] = J * spint.quad( Aij(dphi[m], dphi[n], c, phi[m],phi[n]), 0, 1)[0]


    A[np.ix_(NodeGlobalIndexes-1, NodeGlobalIndexes-1  ) ] =  \
       A[np.ix_(NodeGlobalIndexes-1, NodeGlobalIndexes-1  ) ] + Ml
```

Tekst 12 – Obliczenia całek funkcji bazowych

Ostatnim krokiem jest wzięcie pod uwagę warunków brzegowych do obliczenia poprawnych wartości tablic A i b, obliczenie końcowego wyniku wykorzystując funkcję *np.linalg*, której parametrami są zmienne A i b oraz wizualizacja wyniku.

```
# Edge Conditions
if EdgeConditions[0]['Type'] == 'D':
    ind_wezla = EdgeConditions[0]['Index']
    wart_war_brzeg = EdgeConditions[0]['Value']

    iwp = ind_wezla - 1

    WZMACNIACZ = 10**14

    b[iwp] = A[iwp,iwp]*WZMACNIACZ*wart_war_brzeg
    A[iwp, iwp] = A[iwp,iwp]*WZMACNIACZ


if EdgeConditions[1]['Type'] == 'D':
    ind_wezla = EdgeConditions[1]['Index']
    wart_war_brzeg = EdgeConditions[1]['Value']

    iwp = ind_wezla - 1

    WZMACNIACZ = 10**14

    b[iwp] = A[iwp,iwp]*WZMACNIACZ*wart_war_brzeg
    A[iwp, iwp] = A[iwp,iwp]*WZMACNIACZ

u = np.linalg.solve(A,b)

PlotResult(Nodes, u)
```

Tekst 13 – Obliczenie wyniku analizy geometrii



Rysunek 4 – Wizualizacja wyniku analizy geometrii