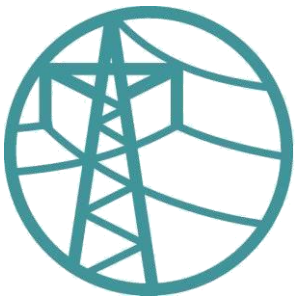


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Szymon Matura

Sprawozdanie

Metody FEM w robotyce

Rzeszów, 2021

Spis treści

1. FEM

FEM jest jedną z metod elementów skończonych, którą można określić ją jako zaawansowaną metodę rozwiązywania układów równań różniczkowych, opierającą się na podziale dziedziny na skończone elementy, dla których rozwiązanie jest przybliżone przez konkretne funkcje i przeprowadzenie obliczeń tylko dla węzłów tego przedziału.
Rodzaje sformułowania wariacyjnego:

- silne (OF)
- słabe (WF)
- odwrotne (IF)

Metoda ta może zostać wykorzystana w mechanice komputer do badania wytrzymałości konstrukcji, symulacji odkształceń, naprężenia, przemieszczenia lub przepływu ciepła.

2. Program

W tym rozdziale został opisany cały program wykorzystany do rozwiązania problemu z wyszczególnieniem i opisaniem funkcji wchodzących w jego strukturę.

2.1. Definicja geometrii

Funkcja GeometriaDefinicja jest jedną z metod implementacji do programu tablic z węzłami i elementami oraz warunkami brzegowymi. Tablice te stanową bazę do późniejszych obliczeń wykonywanych przez inne funkcje. Użytkownik może ręcznie wprowadzić dane aby przetestować oprogramowanie lub opisać niestandardowe obiekty. Funkcja nie pobiera żadnego parametru i zwraca tablice NODES, ELEMS oraz WB.

```
import numpy as np
def GeometriaDefinicja():

    NODES = np.array([[1, 0],
                      [2, 1],
                      [3, 0.5],
                      [4, 0.75]] )

    ELEMS = np.array( [[1, 1, 3],
                       [2, 4, 2],
                       [3, 3, 4]] )

    WB    = [{"ind": 1, "typ":'D', "wartosc":1},
             {"ind": 2, "typ":'D', "wartosc":2}]
    return NODES, ELEMS, WB
```

2.2. Automatyczny generator geometrii

Funkcja przyjmuje jako parametry a, b krańce przedziału oraz jako n ilość węzłów, a zwraca tablicę WEZLY oraz ELEMENTY. Wykorzystanie automatycznego generatora geometrii jest jedną z najszybszych oraz skutecznych metod poprawienie dokładności w obliczeniach, ponieważ program jest w stanie wygenerować coraz gęstsze, a co za tym idzie bardziej dokładne siatki.

```
import numpy as np

def AutomatycznyGeneratorGeometrii(a,b,n):

    lp = np.arange(1,n+1)
    x = np.linspace(a,b,n) ;
    WEZLY = (np.vstack( (lp.T, x.T) )).T #[lp.T, x.T]


    lp = np.arange(1,n)
    C1 = np.arange(1,n)
    C2 = np.arange(2,n+1)
    ELEMENTY = (np.block( [[lp], [C1], [C2] ] ) ).T

return WEZLY, ELEMENTY
```

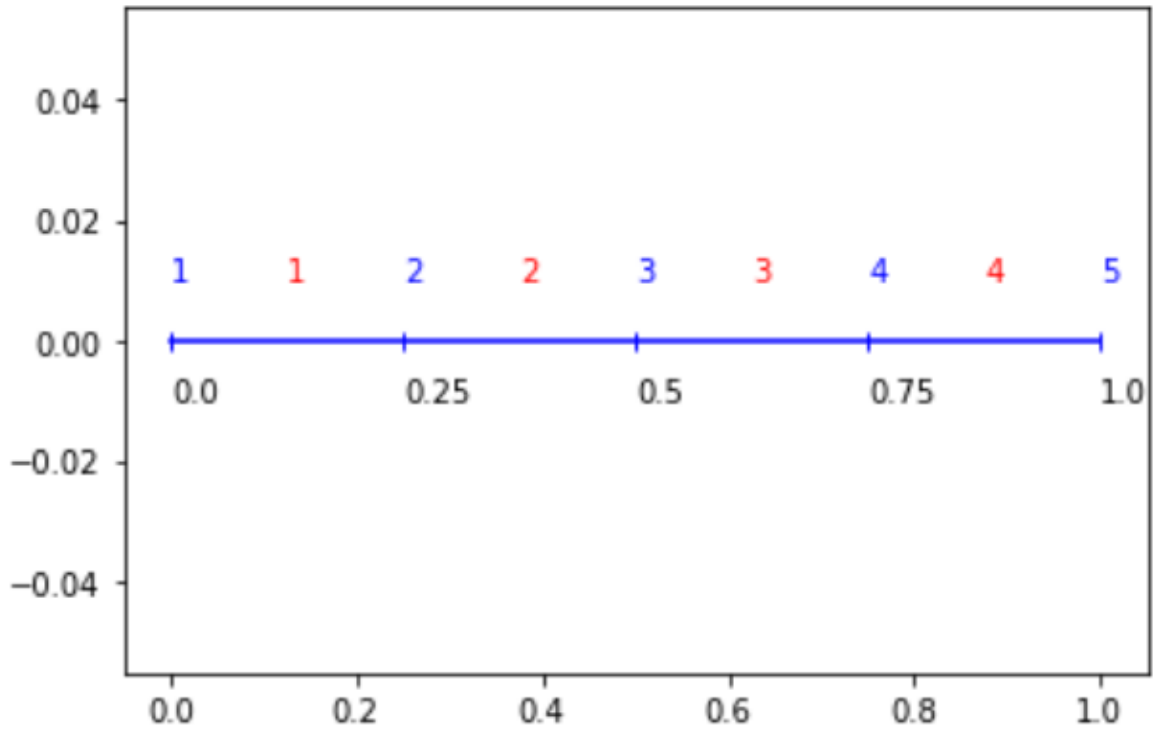
2.3. Rysowanie węzłów i elementów

Funkcja RysujGeometrie przyjmuje jako parametry: WEZLY są to informacje o krańcach przedziałów, EMEMS jest to liczba równo rozmieszczonych węzłów czyli elementy oraz WB czyli warunki brzegowe. Zadaniem tej funkcji jest przestawienie w sposób graficzny wcześniej stworzonej geometrii, tak aby użytkownik mógł w łatwy sposób zobaczyć czy została ona napisana poprawnie. Funkcja powinna po kolei przechodzić przez wiersze w tablicach i stopniowo uaktualniać stworzony wykres.

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

    fh = plt.figure()

    # plt.plot(NODES, np.zeros(np.size(NODES)) )
    # plt.plot(NODES, np.zeros(np.size(NODES)), marker="|",
color='b')
    plt.plot(NODES[:,1], np.zeros( (np.shape(NODES)[0], 1) ),
'-b|' )
```

```
#     indices = NODES[:,0]-1 #np.argsort(NODES)
#     print(indices)
#     text = (NODES[indices,0])
#     print("Indices: "); print(indices)
#     print(text)

    nodeNo = np.shape(NODES)[0]

    for ii in np.arange(0,nodeNo):

        ind = NODES[ii,0]
        x = NODES[ii,1]
        plt.text(x, 0.01, str( int(ind) ), c="b")
        plt.text(x, -0.01, str(x))

    elemNo = np.shape(ELEMS)[0]
    for ii in np.arange(0,elemNo):

        wp = ELEMS[ii,1]
        wk = ELEMS[ii,2]

        x = (NODES[wp-1,1] + NODES[wk-1,1] ) / 2
#         print(x)
        plt.text(x, 0.01, str(ii+1), c="r")

    plt.show()
    return fh
```

Przykładowy rysunek:



Rysunek 1liczba elementów n =5

2.4. Alokacja pamięci

Funkcja Alokacja pobiera jako parament ilość węzłów i zwraca tablice A wypełnioną zerami, gotową do zastąpienia oraz miejsce na wektor prawej strony b. Zwracane tablice mają rozmiar A – nxn oraz b – nx1. Zadaniem tej funkcji jest wydzielenie odpowiedniej ilości miejsca pod macierz i wektor.

```
import numpy as np

def Alokacja(n):

    A = np.zeros([n,n])
    b = np.zeros([n,1])

    return A,b
```

2.5. Funkcje bazowe

Funkcja FunkcjeBazowe przyjmuje jako parametr n stopień wymaganych funkcji kształtu oraz zwraca dwie zmienne, f jest to n+1 elementowa lista funkcji bazowych stopnia n oraz df czyli pochodną tych funkcji. Do obliczeń została wykorzystana funkcja lambda czyli jednolinijkowa funkcja wykorzystywana do obliczeń matematycznych.

```
import numpy as np

def FunkcjeBazowe(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 )

    # elif n == 2:
    #     f =  (lambda , lambda, lambda )
    #     df =

    else:
        raise Exception("Blad w funkcji FunkcjeBazowe().")

    return f,df
```

2.6. Funkcja Aij

Funkcja Aij przyjmuje jako parametry funkcje kształtu związane z i-tym oraz j-tym węzłem f_i, f_j oraz ich pochodne czyli df_i, df_j. Zadaniem tej funkcji jest obliczenie wartości funkcji podcałkowych.

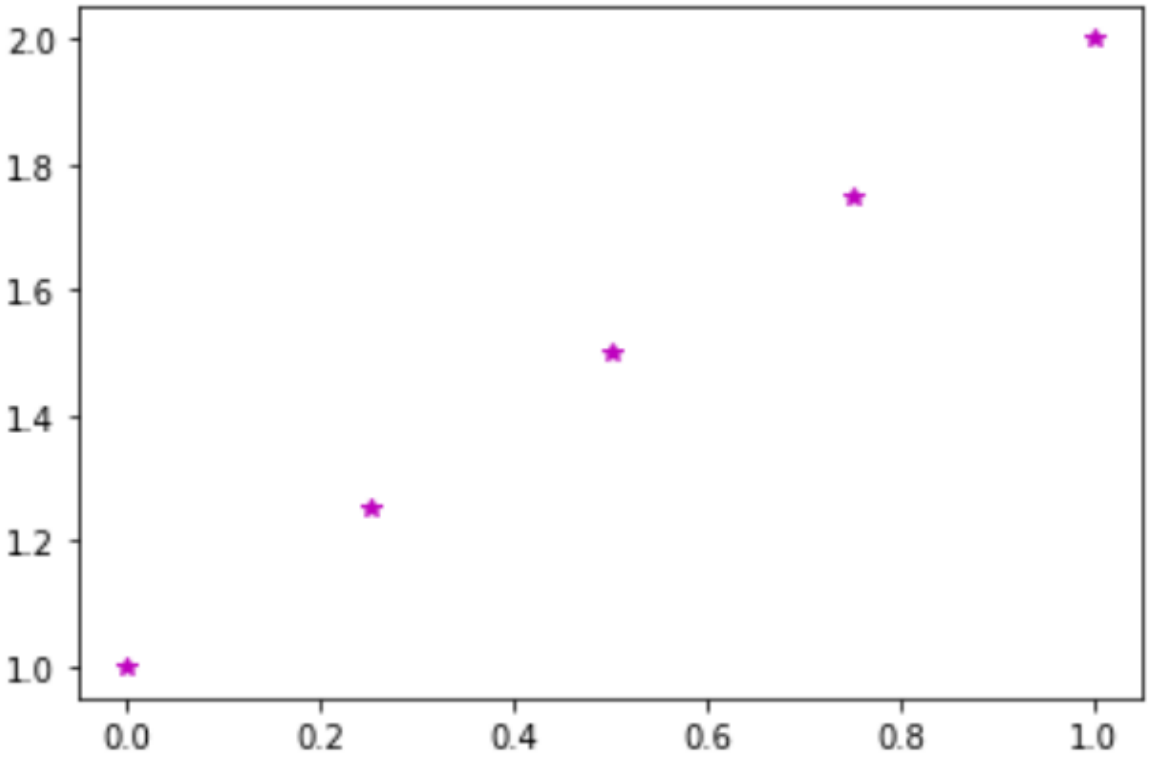
```
import numpy as np

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)
```

2.7. Rysowanie rozwiązania

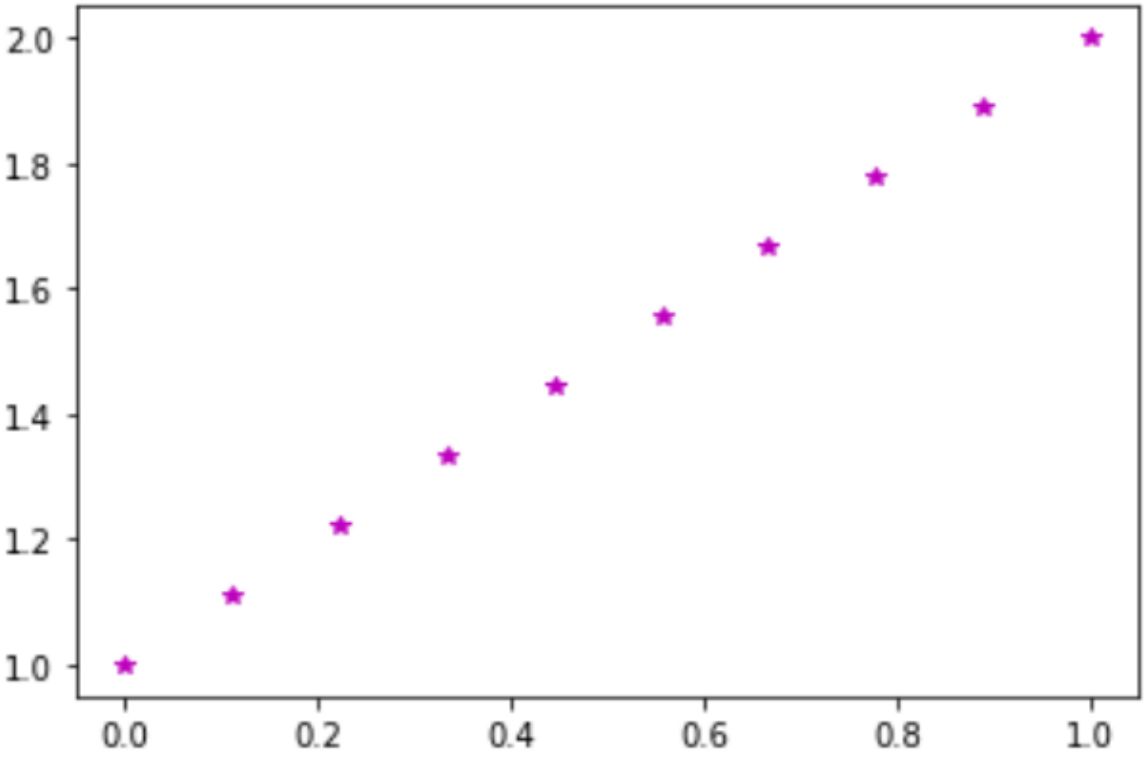
Funkcja RysujRozwiazanie pobiera jako parametry: NODES – krańce przedziału, ELEMS – liczba węzłów, WB – warunki brzegowe oraz u które jest rozwiązaniem macierzy A oraz wektor b obliczone przed wywołaniem funkcji. Ten fragment kodu ma za zadanie zwrócenie rysunku pobranego z funkcji RysujGeometrie z naniesionymi punktami, które stanową rozwiązanie problemu.

```
import numpy as np
import matplotlib.pyplot as plt
from RysujGeometrie import *

def RysujRozwiazanie(NODES, ELEMS, WB, u):

    RysujGeometrie(NODES,ELEMS,WB)

    x = NODES[:,1]

    plt.plot(x, u, 'm*')
```

Przykładowy rysunek:



Rysunek 2RysujRozwiazanie n =5



Rysunek 3RysujRozwiazanie n=10

2.8. Główny człon programu

Głównym zadaniem funkcji main jest wywoływanie kolejnych funkcji z odpowiednimi parametrami tak aby uzyskać poprawne rozwiązanie. Funkcje można podzielić na trzy etapy:

Pierwszym etapem jest przygotowanie programu czyli zaimportowanie odpowiednich bibliotek takich jak numpy, matplotlib czy scipy oraz popranie potrzebnych informacji z pozostałych funkcji w naszym przypadku pobieramy wszystko ponieważ są one stosunkowo proste i nie zawierają zbędnych linijek.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as spint

from GeometriaDefinicja import *
from AutomatycznyGeneratorGeometrii import *
from RysujGeometrie import *
from Alokacja import *
from FunkcjeBazowe import *
from Aij import *
from RysujRozwiazanie import *
```

Drugim etapem jest preprocessing. Ta część kodu odpowiada za przygotowanie właściwej geometrii, która będzie używana w dalej części do obliczeń. Dzięki funkcji Geometria definicja możemy ręczne zadeklarować geometrię lub używając funkcji AutomatycznyGeneratorGeometrii zrobi to za nas program. Przy testowaniu tego programu wyszło, że generowanie geometrii w sposób automatyczny jest dużo dokładniejsze dlatego taka forma została finalnie zastosowana. Dodatkowo w tej część wywoływana jest funkcja prezentująca tą geometrie w formie graficznej.

```
if __name__ == '__main__':

    # Preprocessing

    ## parametry sterujace
    c = 0
    f = lambda x: 0*x # wymuszenie


    ## 2. Geometria
    ### 2.1 Definicja
    # WEZLY, ELEMENTY, WB = GeometriaDefinicja()
    # n = np.shape(WEZLY)[0]

    ### lub Automatyczne wygenerowanie geometrii
    x_a =  0
    x_b =  1
    n = 5
    WEZLY, ELEMENTY =
AutomatycznyGeneratorGeometrii(x_a,x_b,n)
    # warunki brzegowe
    WB    = [{"ind": 1, "typ":'D', "wartosc":1},
             {"ind": n, "typ":'D', "wartosc":2}]


    RysujGeometrie(WEZLY, ELEMENTY, WB)

    # print(WEZLY)
    # print(ELEMENTY)

    A,b = Alokacja(n)

    # print(A)
    # print(b)

    stopien_fun_bazowych = 1
    phi, dphi = FunkcjeBazowe(stopien_fun_bazowych)

    # xx = np.linspace(-1,1, 101)
    # 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' )
```

Ostatnim etapem jest processing. W tej części wykonywane są wszystkie niezbędne obliczenia potrzebne do uzyskania prawidłowego rozwiązania.
Jako oddzielny punkt można również wydzielić postprocessing w którym jest zawarta funkcja RysujRozwiazanie, która prezentuje w sposób graficzny rozwiązanie problemu.

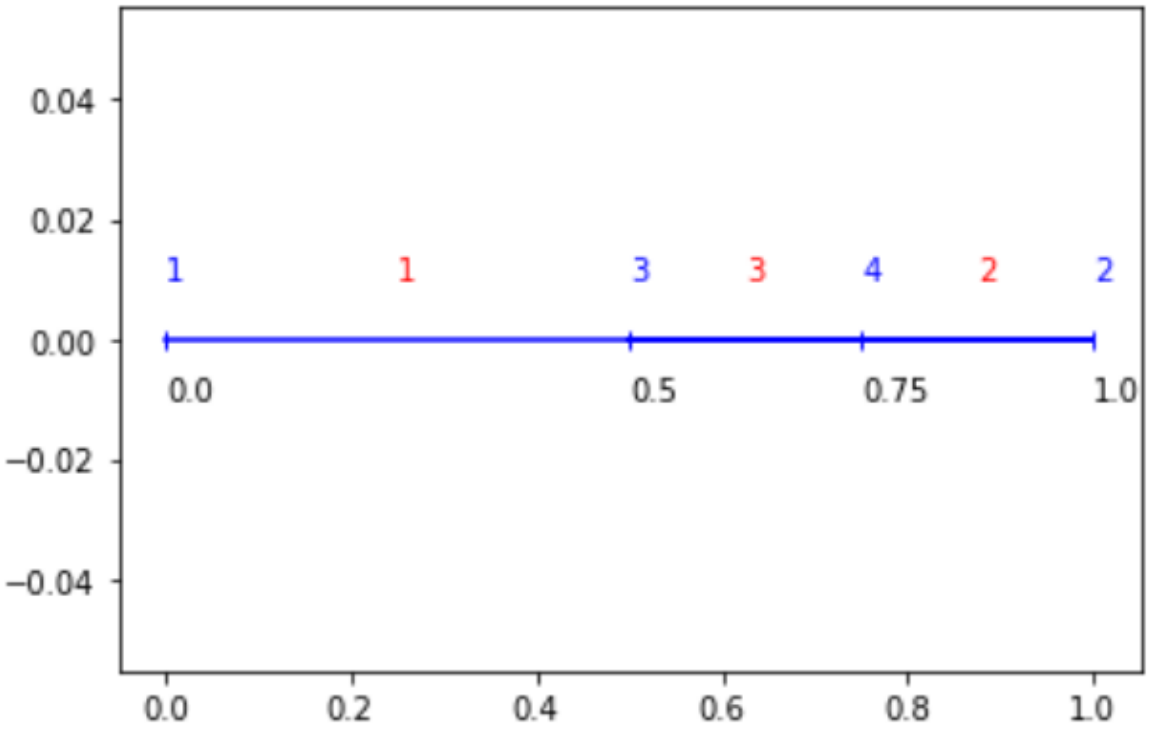
```
    #PROCESSING

    liczbaElementow = np.shape(ELEMENTY)[0]

    for ee in np.arange(0, liczbaElementow ):

        elemRowInd = ee
        elemGlobalInd = ELEMENTY[ee,0]
        elemWezel1 = ELEMENTY[ee,1]     # indeks wezla
poczatkowego elemntu ee
        elemWezel2 = ELEMENTY[ee,2]     # indeks wezla
koncowego elemntu ee
        indGlobalneWezlow = np.array([elemWezel1, elemWezel2
])

        x_a = WEZLY[ elemWezel1-1 ,1]
        x_b = WEZLY[ elemWezel2-1 ,1]


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

        J = (x_b-x_a)/2

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

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

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

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

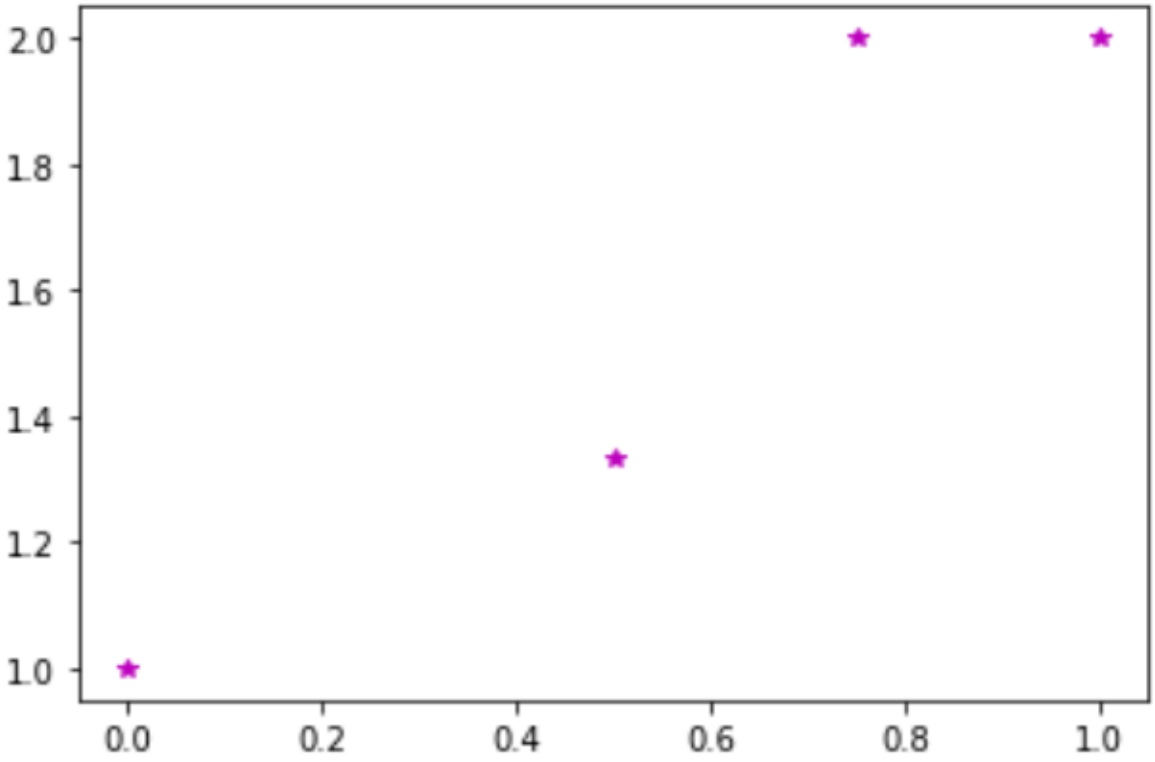
```
        A[np.ix_(indGlobalneWezlow-1, indGlobalneWezlow-1  )
] =  \
             A[np.ix_(indGlobalneWezlow-1, indGlobalneWezlow-1
) ] + Ml


        # print(Ml)
        # print(A)
        # print('\n')


    print(WB)


    # UWZGLEDNIENIE WARUNKOW BRZEGOWYCH
    if WB[0]['typ'] == 'D':
        ind_wezla = WB[0]['ind']
        wart_war_brzeg = WB[0]['wartosc']

        iwp = ind_wezla - 1

        WZMACNIACZ = 10**14

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


    if WB[1]['typ'] == 'D':
        ind_wezla = WB[1]['ind']
        wart_war_brzeg = WB[1]['wartosc']

        iwp = ind_wezla - 1

        WZMACNIACZ = 10**14

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


    if WB[0]['typ'] == 'N':
        print('Nie zaimplementowano jeszcze. Zad.dom')


    if WB[1]['typ'] == 'N':
        print('Nie zaimplementowano jeszcze. Zad.dom')



    # Rozwiazanie ukl row lin
    u = np.linalg.solve(A,b)

    RysujRozwiazanie(WEZLY, ELEMENTY, WB, u)
```

3. Podsumowanie

Po przeprowadzeniu kilku testów w tym programie można zauważyć, że generowanie geometrii w sposób automatyczny jest dużo dokładniejsze. Geometria powinna składać się z równo rozmieszczonych elementów natomiast przy generowaniu jej w sposób ręczny nie udało się tego uzyskać.

Przykładowe rozwiązanie ręcznej geometrii dla n=4:





Automatyczny sposób generowania geometrii wykonuje to zadanie poprawnie co można zobaczy na rysunkach w podrozdziałach 2.3 i 2.7.