Katedra
Informatyki i Automatyki
Politechniki Rzeszowskiej

Metody FEM w robotyce

Sprawozdanie z laboratorium

Wykonujący: Jaromir Stawiarski

Rzeszów, 2021

Python (implementacja FEM):

Definicja parametrów sterujących:

Na początku należy zdefiniować parametry sterujące takie jak stała (c) oraz wymuszenie (f). Dla ułatwienia przyjmujemy te parametry jako zera.

```
#Parametry sterujące

c = 0

f = lambda x:0
```

Generowanie automatyczne geometrii:

W następnym kroku musimy wygenerować geometrię dla podanej przez użytkownika ilości węzłów (n), wartości warunku początkowego (x_a) oraz wartości warunku końcowego (x_b). Po wykonaniu obliczeń przez procesor program będzie zwracał węzły oraz elementy. Do zautomatyzowanego (wielokrotnego) wykorzystywania kodu utworzymy funkcję, w której to użytkowników będzie decydował wartościach wymienionych wyżej parametrów.

Kod funkcji generujTabliceGeometrii:

```
import numpy as np

def generujTabliceGeometrii(x_0, x_p, n):

    temp = (x_p - x_0) / (n - 1)

    matrix = np.array([1, 0 * temp + x_0])


    for i in range(1, n, 1):

        matrix = np.block([

            [matrix],

            [i + 1, i * temp + x_0],

        ])


    matrix2 = np.array([1, 1, 2])

    for i in range(1, n - 1, 1):

        matrix2 = np.block([

            [matrix2],

            [i + 1, i + 1, i + 2],

        ])


    return matrix, matrix2
```

Do zaimplementowania kodu potrzebujemy biblioteki **numpy**, która odpowiada za obliczenia na macierzach. Podana funkcja zwraca dwie macierze.

Przykładowe wywołanie funkcji:

```
from funkcje import generujTabliceGeometrii as GenTab

x_a = 1

x_b = 2

n = 5


wez, el = GenTab(x_a, x_b, n)

print('Tablica węzłów:\n',wez)

print('Tablica elementów:\n',el)
```

Do lepszej widoczności kodu funkcje będą znajdować się w oddzielnym pliku o nazwie funkcje.py, które będzie można importować poleceniem import, a także nadawać im krótsze aliasy. Po wywołaniu funkcji otrzymamy następujące wartości.

```
Tablica węzłów:

[[1.   1.  ]

[2.   1.25]

[3.   1.5 ]

[4.   1.75]

[5.   2.  ]]

Tablica elementów:

[[1 1 2]

[2 2 3]

[3 3 4]

[4 4 5]]
```

W tablicy węzłów pierwsza kolumna oznacza numer węzła, a druga jego wartość. Natomiast w tablicy elementów pierwsza kolumna to numer elementu, a kolejne dwie to numery połączeń węzłów (np. pierwszy element jest pomiędzy węzłem 1, a 2).

Rysowanie geometrii:

Do lepszej interpretacji wyników będzie potrzebne nam graficzne przedstawienie wygenerowania węzłów i elementów, aby tego dokonać należy utworzyć nową funkcję (rysuj_geometrie). Do rysowania wykresów musimy zaimportować nową bibliotekę o nazwie **matplotlib**, a dokładnie jej cześć o nazwie **pyplot**.

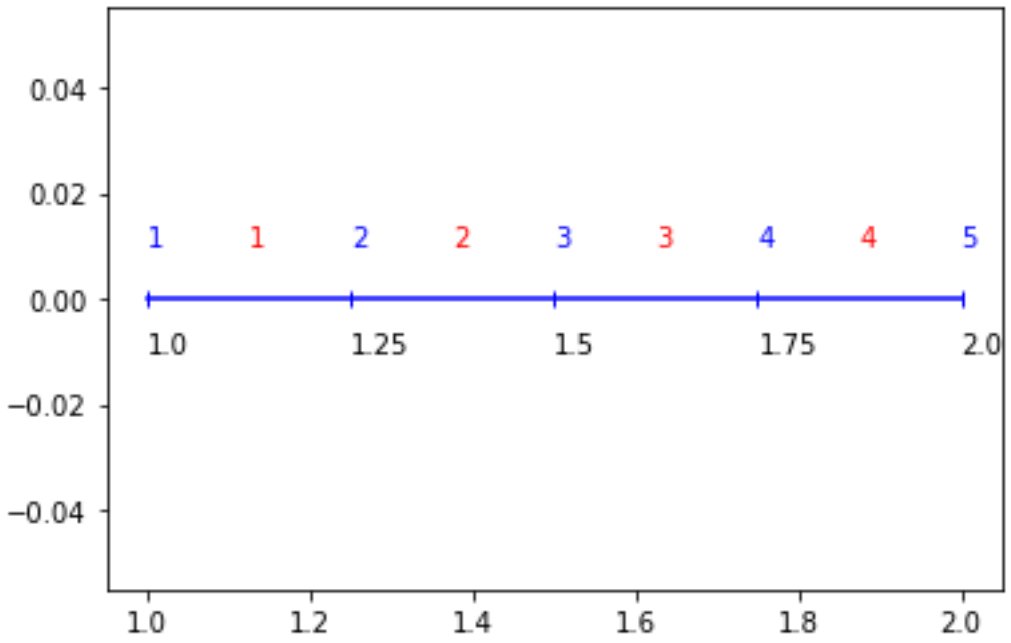
Kod funkcji rysuj_geometrie:

```
import matplotlib.pyplot as plt

def rysuj_geometrie(wez, el, WB):
    fh = plt.figure()


    plt.plot(wez[:,1], np.zeros( (np.shape(wez)[0], 1) ), '-b|' )


    nodeNo = np.shape(wez)[0]


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


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


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


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


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


    plt.show()
    return fh
```

Wywołanie funkcji:

```
from funkcje import rysuj_geometrie as rGeo

WB    = [{"ind": 1, "typ":'D', "wartosc":1},

         {"ind": n, "typ":'D', "wartosc":2}]


rGeo(wez,el,WB)
```

Jako parametry funkcji komputer przyjmuje wartości policzone wcześniej przez funkcje **generujTabliceGeometrii** (w kodzie GenTab). Dodatkowo przyjmuje listę słowników (WB), w której zdefinowane są indeksy, typ oraz wartości. Litera D oznacza to, że jako warunki przyjmujemy kryterium Dirichleta. Rezultatem wywołania funkcji jest wykres przedstawiony na rysunku nr 1.



Rys. 1 Graficzna reprezentacja generowania geometrii

Z rysunku możemy zobaczyć, że węzły rozpoczynają się od 1, a kończą na 2, a liczba elementów wynosi w tym wypadku 4.

Alokacja pamięci na zmienne globalne:

Gdy mamy już zdefiniowane podstawowe funkcje to należy przejść do alokacji pamięci. Pamięć będzie alokowana w postaci dwóch macierzy wypełnionych zerami. W tym wypadku również należy utworzyć funkcje (**alokacja_pamieci_na_zmienne_globalne),** która będzie korzystała z biblioteki **numpy**. Do utworzenia macierz wypełnionych zerami należy skorzystać z funkcji **zeros()**, a następnie podać jej rozmiary. Alokacja pamięci będzie potrzebna w przypadku, gdy będą generowane rozwiązania (dane potrzebne do ich generowania). Jako parametr funkcja będzie przyjmowała liczbę węzłów (n).

Kod funkcji alokacja_pamieci_na_zmienne_globalne:

```
import numpy as np

def alokacja_pamieci_na_zmienne_globalne(n):

    A = np.zeros((n, n))

    b = np.zeros((n, 1))

    return A, b
```

Wywołanie funkcji alokacja_pamieci_na_zmienne_globalne:

```
from funkcje import alokacja_pamieci_na_zmienne_globalne as Alok

A,b = Alok(n)
```

Definiowanie funkcji bazowych:

W następnym kroku należy zdefiniować funkcje bazowe i ich pochodne, które następnie będą scałkowane. W implementacji należy zdefiniować funkcje zarówno rosnącą jak i malejącą. Dla przykładu zdefiniowano wielomian zerowego stopnia oraz pierwszego, dla wyższych stopni wielomianu nastąpi wywołanie wyjątku. Jako parametr funkcje przyjmują stopień wielomianu, a następnie definiowany jest x.

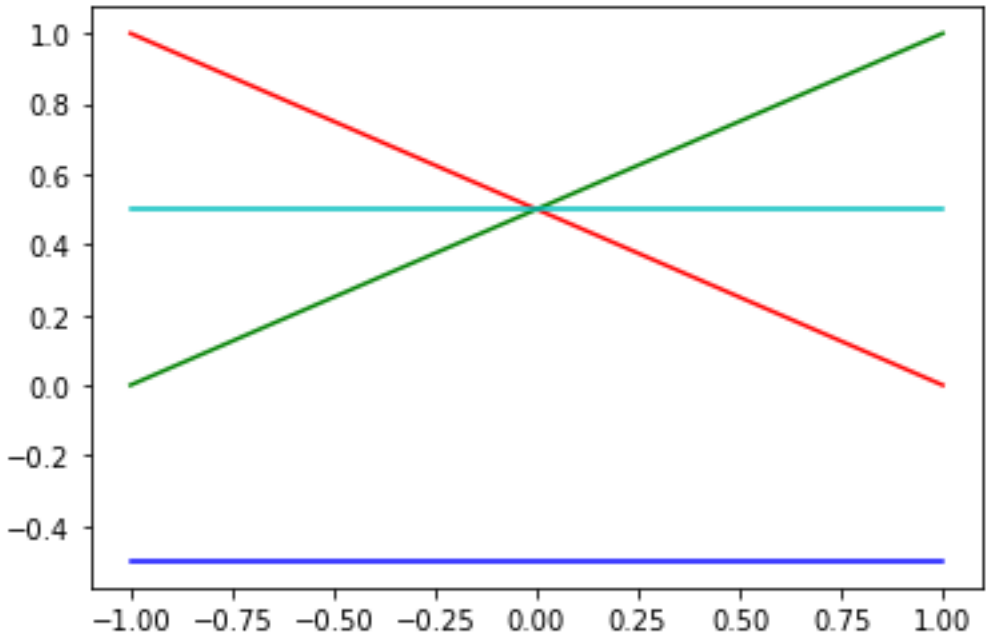
Kod funkcji funkcje_bazowe:

```
def funkcje_bazowe(n):
    if n==0:
        f = lambda x: x*0 + 1
        df = lambda x: x*0
    elif n==1:
        f = (lambda x: -1/2*x+1/2, lambda x: 1/2*x+1/2)
        df = (lambda x: -1/2+x*0, lambda x: 1/2+x*0)
    else:
        raise Exception("Nieobsułgiwany wielomian")
    return f,df
```

Przykładowe wywołanie funkcji:

```
SW = 1 #stopien wielomianu

phi,dphi = FBaz(SW)

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')
```

Po wywołaniu funkcji oraz przedstawienie ich na jednym wykresie otrzymujemy wykres taki jak na rysunku nr 2. Na wykresie kolorem czerwonym i zielonym przedstawiono wielomian 1. stopnia (czerwony malejący, zielony rosnący), a kolorami niebieskim i cyjanowym ich pochodne.



Rys. 2 Wywołanie funkcji bazowych

Agregacja macierzy:

W kolejnym kroku należy unormować element tak, aby granice całkowania były takie same niezależnie od elementu < -1,1 >. W celu przekształcenia przedziału do przedziału unormowanego należy zastosować jakobian. Dla uproszczenia złożoności kodu należy utworzyć kolejną funkcje, która będzie obliczała wartości pod całką (nie liczyła jej).

Kod funkcji elementy_macierzy:

```
def elementy_macierzy(dphi1,dphi2,c,phi1,phi2):

    Aij = lambda x: -dphi1(x)*dphi2(x)+ c *phi1(x)*phi2(x)

    return Aij
```

Mając już zdefiniowaną funkcję (**elementy_macierzy**) należy dodać nową bibliotekę o nazwie **scipy.integrate**, aby obliczyć całki. Na początku w kodzie należy zdefiniować indeks elementu równania (EIR), indeks elementu globalnego (EIG) oraz indeksy węzłów początkowego (EW1) i końcowego (EW2). Następnie należy zdefiniować tablicę dla globalnych indeksów węzłów (IGW) oraz wyznaczyć wartości przedziałów (x_a i x_b). W końcowych krokach należy obliczyć jakobian (J) oraz dokonać alokacji pamięci dla macierzy elementów (Ml), a następnie wykonać operację kwadratury Gaussa (funkcja quad).

```
import scipy.integrate as spint
from funkcje import elementy_macierzy as ElMat
liczbaElementow = np.shape(el)[0]
for ee in np.arange(0, liczbaElementow ):

    EIR = ee
    EIG = el[ee,0]
    EW1 = el[ee,1]
    EW2 = el[ee,2]
    IGW = np.array([EW1,EW2])

    x_a = wez[EW1-1,1]
    x_b = wez[EW2-1,1]
    J = (x_b-x_a)/2

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

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

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

print(A)
```

Po uruchomieniu skryptu otrzymamy następującą macierzy:

```
[[-0.0625  0.0625  0.      0.      0.    ]
[ 0.0625 -0.125   0.0625  0.      0.    ]
[ 0.      0.0625 -0.125   0.0625  0.    ]
[ 0.      0.      0.0625 -0.125   0.0625]
[ 0.      0.      0.      0.0625 -0.0625]]
```

Warunki brzegowe:

Po zakończeniu liczenia macierzy należy uwzględnić typ warunków brzegowych (Dirichleta lub Neumanna). Do zniwelowania wpływu innych równań można zastosować trik w postaci przemożenia przez bardzo dużą liczbę – wzmacniacz (wzm) przez to ułatwia to rozwiązanie układu równań.

Warunek Dirichleta:

```
if WB[0]['typ'] == 'D':
    indw = WB[0]['ind']
    wwb = WB[0]['wartosc']

    iwp = indw - 1
    wzm = 10**14

    b[iwp] = A[iwp,iwp]*wzm*wwb
    A[iwp, iwp] = A[iwp,iwp]*wzm

if WB[1]['typ'] == 'D':
    indw = WB[1]['ind']
    wwb = WB[1]['wartosc']

    iwp = indw - 1
    wzm = 10**14

    b[iwp] = A[iwp,iwp]*wzm*wwb
    A[iwp, iwp] = A[iwp,iwp]*wzm
```

Warunek Neumanna:

Rozwiązanie układu równań:

Po otrzymaniu macierzy A i b zostało tylko etap rozwiązania układu równań i przedstawienia rozwiązania na wykresie. Do narysowania wykresu znowu należy utworzyć nową funkcję o nazwie **rysuj_rozwiazanie**. W funkcji wykorzystamy poprzednią funkcję rysuj_geometrie, żeby uwzględniła zapisane wcześniej elementy oraz węzły.

Kod funkcji rysuj_rozwiazanie:

```
def rysuj_rozwiazanie(wez, el, WB, u):


    from funkcje import rysuj_geometrie


    rysuj_geometrie(wez,el,WB)


    x = wez[:,1]


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

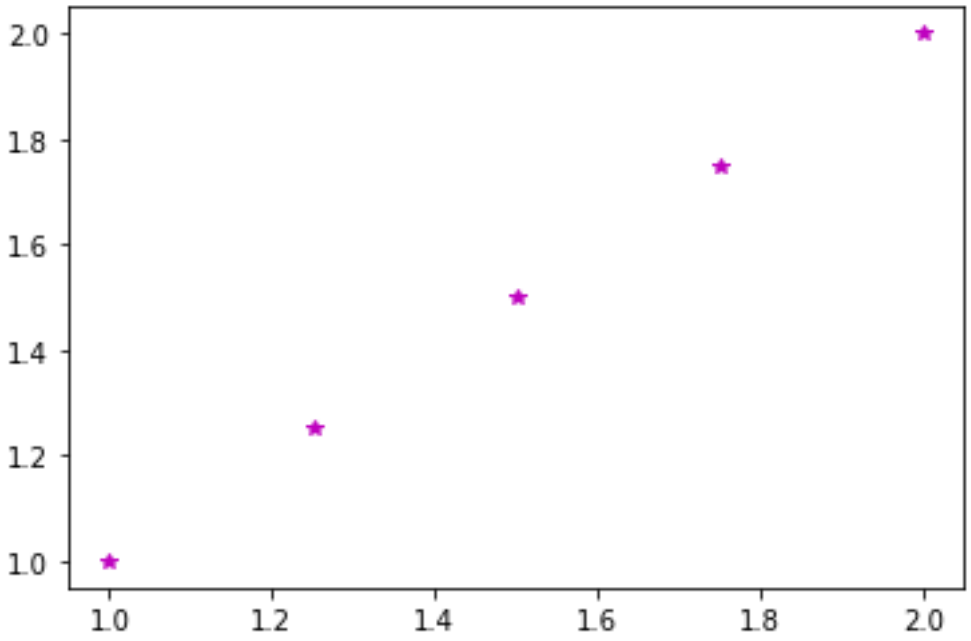
Kod rozwiązania układu równań oraz przedstawienia rozwiązania:

```
from funkcje import rysuj_rozwiazanie as Rezult
u = np.linalg.solve(A,b)


print(u)
Rezult(wez, el, WB, u)
```

Do rozwiązania układu równań użyto funkcji wbudowanej w bibliotekę numpy o nazwie linalog.solve(), który jako parametry przyjmuje dwie macierze A i b.

Rozwiązaniem naszego układu równań są współrzędne węzłów:

```
[[1.  ]
 [1.25]
 [1.5 ]
 [1.75]
 [2.  ]]
```



Rys.3 Rozwiązanie

Git i Github:

Komendy gita:

git config --global user.name Terrnox

git config --global user.email stawiarskijaromir@gmail.com

mkdir sprawozdanie

cd sprawozdanie

git init

git status

git add fem.py

git add funkcje.py

git add Sprawozdanie-Jaromir-Stawiarski-FEM.pdf

git status

git commit

git log