Primeiro trabalho prático de elementos finitos -2d

Esse notebook desse trabalho vai estar no colab desse link .

Mas gosto do conceito de documentar tudo para melhor entendimento, então esse PDF vai ter a explicação das linhas de código.

Exemplo :

Importação de bibliotecas:

```
import numpy as np
import matplotlib.pyplot as plt
from sympy import Matrix, init_printing

init_printing()
```

1- Nossos valores do problema:

Nos foi dito para considerar:

Temperatura inicial do sistema: $T_\infty = 293,15^oK$

Largura de uma célula de combustível do reator nuclear: $L = 0,025m$

Condutividade térmica: $k = 35\ W/m^oK$

Energia interna devido a reação nuclear: $U = 67967200\ W/m^3$

Densidade de massa das células de combustível: $\rho = 8618,0\ Kg/m^3$

Coeficiente de transferência de calor entre a célula de combustível e o fluido: $\overline{h} = 0,25 * 10^5 W/^oKm^2$

Capacidade térmica: $c = 460,0\ J/Kg^oK$

Nosso código deve ter em algum momento um escopo de declaração desses valores.

```
#declarando respectivamente tudo que está acima
T_inf = 293.15
L = 0.025
k = 35
U = 67967200
rho = 8618
h_barra = 0.25 * (10**5)
c = 460
e = 2.71828182845904523536028 7  # nao foi definido no problema mas estou fazendo por
                                 # conta própria por conveniencia caso precise
```

2- O que nos é pedido?

1- Encontre a distribuição estacionária de temperatura na célula de combustível considerando condição de contorno prescrita e igual a T_∞ do lado direito. Compare com a solução analítica.

2- Encontre a distribuição estacionária de temperatura na célula de combustível considerando, agora, condição de contorno de fluxo. Repare que, neste caso, a condição de contorno do lado direito envolve a derivada e a temperatura em $x = L$, algo que não foi visto em aula. Compare com a solução analítica.

3- Como o problema de valor de contorno e inicial deve ser definido se considerar o domínio espacial como [0, 2L]?

O problema 1 e 2 são bastante parecidos, mas um é baseado em condição de contorno prescrita e o outro é condição de contorno de fluxo. Mas no próprio enunciado do problema, foi dito que nossa solução seria dada por :

$$T(x,t) = T_\infty + \frac{UL^2}{2k}\left[1 - \left(\frac{x}{L}\right)^2 + \frac{2}{B}\right]\left[1 - \exp\left(-\frac{3a}{L^2}\frac{B}{B+3}t\right)\right]$$

onde $B = \frac{\overline{h}L}{k}$ e $a = \frac{k}{\rho c}$.

Mas também nos foi dito no enunciado que quando temos uma condição de contorno prescrita igual a T_∞ no lado direito, faremos $\overline{h} \to \infty$, implicando consequentemente em $B \to \infty$ e fazendo com que nossa solução mude de cara e fique a seguinte forma.

$$T(x,t) = T_\infty + \frac{UL^2}{2k}\left[1 - \left(\frac{x}{L}\right)^2\right]\left[1 - \exp\left(-\frac{3a}{L^2}\right)t\right]$$

3- Implementação de elementos finitos

Primeiro fizemos a criação da malha:

```
coord = [
        [0,     0],
        [0.005, 0],
        [0.01,  0],
        [0.015, 0],
        [0.02,  0],
        [0.025, 0],

        [0,     0.05],
        [0.005, 0.05],
        [0.01,  0.05],
        [0.015, 0.05],
        [0.02,  0.05],
        [0.025, 0.05],
]
```

E também criaremos nossa matriz L_{global} e nosso vetor de equações:

```
LG = [                                  # L global
     [1 ,  2 ,  3 ,   4 ,  5 ],
     [2 ,  3 ,  4 ,   5 ,  6 ],
     [8 ,  9 , 10 ,  11 , 12 ],
     [7 ,  8 ,  9 ,  10 , 11 ]
     ]

EQ = [1, 2, 3, 4, 5, 0, 6, 7, 8, 9, 10, 0]  # vetor de equações

Neq = max(EQ)  # número de equações
Nel = len(LG)  # número de elementos
Nnos = len(EQ) # número de nós
```

Antes de tentarmos montar a K e F local, precisamos primeiramente definir nossas funções de interpolação já vistas em sala de aula e suas respectivas derivadas , tanto em relação a η quanto em relação a ξ .

```
def Phi_1(xi, eta): return ((1 - xi)*(1 - eta)) / 4
def Phi_2(xi, eta): return ((1 + xi)*(1 - eta)) / 4
def Phi_3(xi, eta): return ((1 + xi)*(1 + eta)) / 4
def Phi_4(xi, eta): return ((1 - xi)*(1 + eta)) / 4

def dPhi1_dxi(xi, eta): return (-(1 - eta)) / 4
def dPhi1_deta(xi, eta): return (-(1 - xi)) / 4

def Phi2_dxi(xi, eta): return (1 - eta) / 4
def dPhi2_deta(xi, eta): return (-(1 + xi)) / 4

def dPhi3_dxi(xi, eta): return (1 + eta) / 4
def dPhi3_deta(xi, eta): return (1 + xi) / 4

def dPhi4_dxi(xi, eta): return (-(1 + eta)) / 4
def dPhi4_deta(xi, eta): return (1 - xi) / 4
```

E com nossas funções de interpolação, conseguimos facilmente obter o jacobiano:

```
# sabendo que jacobiano --> J = D*y
def jacobiano(xi, eta, coords):
    D = np.zeros((2,4))
    D = [
        [dPhi1_dxi(xi, eta) , dPhi2_dxi(xi, eta) , dPhi3_dxi(xi, eta) , dPhi4_dxi(xi, eta) ],
        [dPhi1_deta(xi, eta), dPhi2_deta(xi, eta), dPhi3_deta(xi, eta), dPhi4_deta(xi, eta)]
        ]
    J = np.dot(D,coords)
    return D, J
```

Como vimos em aula, devemos organizar nossas coordenadas globais com as coordenadas de cada elemento finito, e isso vamos fazer na função abaixo que cria uma matriz 4x2 que é similar à nossa global:

```
def coord_elemento(e):
    coord_local = np.zeros((4, 2))
    for i in range(4):
        coord_local[i, :] = coord[LG[i][e]-1];
    return coord_local
```

Agora que temos “tudo” de ferramenta necessária para construir a K local, faremos isso, e da K local nós podemos usar o conceito de elementos finitos que os relacionam com a K global pela LG, e depois de obter K local e global, a F local e global são obtidas também de forma imediata.

Mas vamos com uma coisa de cada vez.

K^e :

```
Q = np.matrix([[1, 0], [0, 1]])  # criando uma Q identidade

B = (h_barra*L)/k # não será usada no caso primário com temperatura preescrita
a = k/(rho*c)

Q =  Q*a

def K_local(e):
    PG = [-0.5773502691, 0.5773502691]  # pontos de Gauss
    w = [1, 1]                          # pesos
    N_int = len(PG)                     # numero de pontos em cada direcao
    coord_local = coord_elemento(e)

    K_e = np.zeros((4, 4), dtype = object)
    for a in range(N_int):
        for b in range(N_int):
            D, j = jacobiano(PG[a], PG[b], coord_local)
            det_J = np.linalg.det(j)
            if det_J <= 0:
                print("Determinante negativo!")
                exit()

            B = np.dot(np.linalg.pinv(j),D)
            x = np.dot(B.transpose(),Q)
            x = np.dot(x,B)
            K_e += x * det_J * w[a] * w[b]  # integral de (∇φᵀ)Q(∇φ) dΩ

    return K_e
```

K global :

```
def K_global(K, K_e, e):
    for a in range(4):
        for b in range(4):
            # vamos relacionar a coordenadas (i,j) com o vetor de equações via LG
            i = EQ[LG[a][e]-1]
            j = EQ[LG[b][e]-1]
            # as que não forem 0, vão receber o valor da matriz K local do El. Finito
            if (i != 0) and (j != 0):   # iremos pular
                K[i-1][j-1] += K_e[a][b]
    return K
```

Agora com as duas matrizes acima, podemos obter também a F local e global, mas primeiro tratamentos das condições de contorno:

```
P = np.zeros(Nnos)
P[5] = T_inf
P[11] = T_inf
p = np.zeros(4);

def F_prescrito(K_e, e):
  p = P[LG[0:4, e]-1]
  F_p = np.dot(K_e, p)
  return F_p
```

E agora fazemos a F local :

```
f = lambda x : U / (rho*c) # definindo nossa função como uma lambda

def F_local(K_e, e):
    PG = [-0.5773502691, 0.5773502691]  # pontos de Gauss
    w = [1, 1]                          # pesos
    N_int = len(PG)                     # numero de pontos em cada direcao
    coord_local = coord_elemento(e)
    M = np.zeros((4,4), dtype=object)
    for a in range (N_int):
        for b in range(N_int):
            do_nothing , J = jacobiano(PG[a], PG[b], coord_local)
            det_J = np.linalg.det(J)
            if det_J <= 0:
                print("Determinante negativo!")
                exit()

            Phi = [Phi_1(PG[a], PG[b]), Phi_2(PG[a], PG[b]), Phi_3(PG[a], PG[b]), Phi_4(PG[a], PG[b])]
            W = w[a]*w[b]*det_J
            for m in range(4):
                for n in range(4):
                    M[m][n] += Phi[m]*Phi[n]*W

    f_e = np.zeros(4)
    for a in range(4):
      for j in range(2):
        f_e[a] = f(coord_local[a][j])

    f_e = np.dot(M,f_e)

    return f_e - F_prescrito(K_e, e)
```

e agora K global :

```
def F_global(F, F_e, e):
    for a in range(4):
        i = EQ[LG[a][e]-1]
```

```
        if i != 0:
            F[i-1] = F[i-1] + F_e[a]

    return F
```

Tendo tudo isso feito, teremos no fim que fazer fazer as locais e globais para cada elemento finito que temos, que é $\text{número de colunas de LG} = N_{elementos}$ que no nosso caso é 5.

```
for e in range(Nel):
  K_e = K_local(e)
  K_global(K, K_e, e)
  F_e = F_local(K_e, e)
  F_global(F, F_e, e)
```

E ai nossa K e F definidas lá encima agora foram alimentadas com informações dos nossos 5 elementos finitos que nós tiramos da malha que escolhemos do problema.

Disso, agora basta resolver o sistema linear, trivial com a linha de comando:

```
solucao_numerica = np.linalg.solve(K,F)
```

E depois resolver a solução analítica também:

```
solucao_analitica = []
xs = np.arange(0, 0.02, 0.005)
t = lambda x : T_inf + (U*L**2)/(2*k)*(1 - (x/L)**2)
for x in xs:
  solucao_analitica.append(t(x))
```

E comparar ambos os resultados e conferir, meu output foi o seguinte:

```
[263] print("\nA solução analítica é:\n")
      print(solucao_analitica)
      print("\nA solução numérica é:\n")
      print(solucao_numerica)
```

```
A solução analítica é:

[900.0000000000001, 875.7260000000001, 802.9040000000002, 681.5340000000001]

A solução numérica é:

[900.    875.726 802.904 681.534 511.616 900.    875.726 802.904 681.534
 511.616]
```

O que valida nosso método, estamos encima do resultado analítico, o que é excelente… 😉

Autor : Matheus Oliveira Silva
DRE: 119180151
E-mail: matheusflups8@gmail.com / matheusolivsilv.job@gmail.com