Trabalho4

August 8, 2020

1 Método dos Elementos Finitos - Trabalho 4

Universidade Federal Fluminense

Disciplina ministrada pelo Prof. Marco Ferro

marcoferro@id.uff.br

Aluno Noé de Lima

noe_lima@id.uff.br

Este trabalho visa aplicar o MEF ao Estado Plano de Tensões (EPT).

Primeiro semestre de 2020

A célula a seguir configura o Jupyter-Notebook para exibir as equações matemáticas no formato do ambiente LaTeX e importa as bibliotecas necessárias.

[1]:
```
import numpy as np
import sympy as sp
import pandas as pd
from sympy.vector import CoordSys3D, divergence
sp.init_printing(use_latex='mathjax',latex_mode='equation*')
!uname -a
```

```
Linux DESKTOP-CR708A2 4.19.104-microsoft-standard #1 SMP Wed Feb 19 06:37:35 UTC
2020 x86_64 x86_64 x86_64 GNU/Linux
```

Contents

2 Introdução

O Estado Plano de Tenção - EPT ou o Estado Plano de Deformação - EPD, caracterizam-se por serem estruturas planares (no plano xy), com carregaemntos, reações de apoio, restrições e deformações igualmente planares. Tais estruturas têm, na direção z, espessura fixa constante t. As condições que definem e diferenciam o EPT e o EPD estão relacionadas às restrições na direção z.

Temos, na dereção z, das faces do plano, as seguintes variáveis de Tensão $\times$ Deformação: σ_z e ϵ_z.

Portanto, segue que:

No EPT: $\sigma_z = 0$;

No EPD: $\epsilon_z = 0$.

O EPT tem, portanto, suas faces livres, sem tensões de reação e liberdade de deformação. Tal é a situação em chapas, onde uma estrutura plana recebe carregamentos no mesmo plano da face, e não recebe carregamentos transversais (como nas placas ou lages). Nessa configuração, a espessura t da chapa pode variar, pois não há tensão de reação na direção z.

Já o EPD é caracterizado por uma restrição de deformação, já que há reação de tensão na direção z que restringe a deformação. Nessa configuração, portanto, a espessura t não varia. Este é o caso onde o plano compõe uma seção transversão de uma estrutura maior, como um muro de arrimo ou uma barragem de represa. Neste caso, a espessura é fixa e definida como $t = 1$, sendo as tensões definidas em termos de comprimento de muro ou barragem.

As variáveis simbólicas definidas abaixo serão utilizadas na análise subsequente.

[2]:
```python
E,nu = sp.symbols('E,nu')
G = E/(2*(1+nu))
```

Sendo,

- E o Módulo de Elasticidade Longitudinal;
- ν a constante de Poisson do material; elastica; e
- G o Módulo de Elasticidade Transversal.

[3]:
```python
sigma_x,sigma_y,sigma_z = sp.symbols('sigma_x,sigma_y,sigma_z')
tau_xy,tau_yz,tau_zx = sp.symbols('tau_xy,tau_yz,tau_zx')
epsilon_x,epsilon_y,epsilon_z = sp.symbols('epsilon_x,epsilon_y,epsilon_z')
gamma_xy,gamma_yz,gamma_zx = sp.symbols('gamma_xy,gamma_yz,gamma_zx')
```

Onde,

- σ_x, σ_y e σ_z são, respectivamente, as tensões normais nas direções x, y e z;
- τ_{xy}, τ_{yz} e τ_{zx} são, respectivamente, as tensões cisalhantes nas direções xy, yz e zx;
- ε_x, ε_y e ε_z são, respectivamente, as deformações lineares em x e y e z;
- γ_{xy}, γ_{yz} e γ_{zx} são, respectivamente, as deformações angulares nas direções xy, yz e zx.

A partir das restrições do sistema bidimensional, temos o seguinte Vetor de Tensões $\vec{\sigma}$:

[4]:
```
sigma = sp.Matrix([sigma_x,sigma_y,tau_xy])
display(sigma)
```

$$\begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix}$$

Bem como o Vetor de Deformações $\vec{\varepsilon}$:

[5]:
```
epsilon = sp.Matrix([epsilon_x,epsilon_y,gamma_xy])
display(epsilon)
```

$$\begin{bmatrix}\epsilon_x\\ \epsilon_y\\ \gamma_{xy}\end{bmatrix}$$

Temos também a matriz **D**, em função de E_{var} e ν_{var}:

[6]:
```
E_var,nu_var = sp.symbols('E_var,nu_var')
D_var = (E_var/(1-nu_var**2))*sp.
↪Matrix([[1,nu_var,0],[nu_var,1,0],[0,0,(1-nu_var)/2]])
display(D_var)
```

$$\begin{bmatrix}\frac{E_{var}}{1-\nu_{var}^2} & \frac{E_{var}\nu_{var}}{1-\nu_{var}^2} & 0\\ \frac{E_{var}\nu_{var}}{1-\nu_{var}^2} & \frac{E_{var}}{1-\nu_{var}^2} & 0\\ 0 & 0 & \frac{E_{var}\left(\frac{1}{2}-\frac{\nu_{var}}{2}\right)}{1-\nu_{var}^2}\end{bmatrix}$$

A equação do sistema, então, fica:

[7]:
```
eq = sp.Eq(sigma, D_var*epsilon)
display(eq)
```

$$\begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix} = \begin{bmatrix}\frac{E_{var}\epsilon_x}{1-\nu_{var}^2} + \frac{E_{var}\epsilon_y\nu_{var}}{1-\nu_{var}^2}\\ \frac{E_{var}\epsilon_x\nu_{var}}{1-\nu_{var}^2} + \frac{E_{var}\epsilon_y}{1-\nu_{var}^2}\\ \frac{E_{var}\gamma_{xy}\left(\frac{1}{2}-\frac{\nu_{var}}{2}\right)}{1-\nu_{var}^2}\end{bmatrix}$$

3 Estado Plano de Tensão - EPT

As condições de contorno do EPT, conforme definido na introdução, podem ser definidas matematicamente como:

$$\begin{cases}\sigma_z & = & 0\\ \tau_{yz} & = & 0\\ \tau_{zx} & = & 0\end{cases}$$

E,

$$\varepsilon_z = -\frac{\nu}{E}\left(\sigma_x + \sigma_y\right) \neq 0$$

Nessas condições, temos:

$$\begin{cases} E_{var} & = & E \\ \nu_{var} & = & \nu \end{cases}$$

Logo, $\mathbf{D}_{EPT} =$

[8]:
```
ept = eq.subs(E_var,E).subs(nu_var,nu)
display(ept)
```

$$\begin{bmatrix}\sigma_x \\ \sigma_y \\ \tau_{xy}\end{bmatrix} = \begin{bmatrix}\frac{E\epsilon_x}{1-\nu^2} + \frac{E\epsilon_y\nu}{1-\nu^2} \\ \frac{E\epsilon_x\nu}{1-\nu^2} + \frac{E\epsilon_y}{1-\nu^2} \\ \frac{E\gamma_{xy}\left(\frac{1}{2}-\frac{\nu}{2}\right)}{1-\nu^2}\end{bmatrix}$$

[ ]:
```

```

4 Estado Plano de Deformação

As condições de contorno do EPD, conforme definido na introdução, podem ser definidas matematicamente como:

$$\begin{cases} \varepsilon_z & = & 0 \\ \gamma_{yz} & = & 0 \\ \gamma_{zx} & = & 0 \end{cases}$$

E,

$$\sigma_z = \nu\left(\sigma_x + \sigma_y\right) \neq 0$$

Nessas condições, temos:

$$\begin{cases} E_{var} & = & \frac{E}{1-\nu^2} \\ \nu_{var} & = & \frac{\nu}{1-\nu} \end{cases}$$

Logo, $\mathbf{D}_{EPD} =$

[9]:
```
epd = eq.subs(E_var,E/(1-nu**2)).subs(nu_var,nu/(1-nu))
display(epd)
```

$$\begin{bmatrix}\sigma_x\\ \sigma_y\\ \tau_{xy}\end{bmatrix} = \begin{bmatrix}\frac{E\epsilon_x}{(1-\nu^2)\left(-\frac{\nu^2}{(1-\nu)^2}+1\right)} + \frac{E\epsilon_y\nu}{(1-\nu)(1-\nu^2)\left(-\frac{\nu^2}{(1-\nu)^2}+1\right)}\\ \frac{E\epsilon_x\nu}{(1-\nu)(1-\nu^2)\left(-\frac{\nu^2}{(1-\nu)^2}+1\right)} + \frac{E\epsilon_y}{(1-\nu^2)\left(-\frac{\nu^2}{(1-\nu)^2}+1\right)}\\ \frac{E\gamma_{xy}\left(-\frac{\nu}{2(1-\nu)}+\frac{1}{2}\right)}{(1-\nu^2)\left(-\frac{\nu^2}{(1-\nu)^2}+1\right)}\end{bmatrix}$$

5 Cálculo das Tensões

Tanto para EPT quanto para EPD, temos a seguinte equação para o cálculo das tensões (Derivada da Lei de Hooke):

$$\vec{\sigma} = \mathbf{D}\vec{\varepsilon}$$

Bem como,

$$\vec{\varepsilon} = \mathbf{L}\vec{u}$$

Onde,

- $\mathbf{L} \rightarrow$ é a matriz de operadores que relaciona os deslocamentos com as deformaçoes.

Como

$$\bar{u} = \sum_{i=1}^{N} N_i u_i$$

tem-se:

$$\mathbf{L} = \begin{bmatrix}\frac{\partial}{\partial x} & 0\\ 0 & \frac{\partial}{\partial y}\\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x}\end{bmatrix}$$

$$\vec{u} = \begin{Bmatrix}u\\ v\end{Bmatrix}$$

$$\vec{\varepsilon} = \begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{Bmatrix}$$

6 Método dos Elementos Finitos

O Elemento bidimensional a ser estudado é o Triangular de Deformação Constante, ou Constant Strain Triangle - CST, que possui como funções de interpolação N_i, para os deslocamentos u e v, polinômios do primeiro grau em x e y.

Portanto,

$$
\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & x & y \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \end{Bmatrix}
$$

As funções N_i têm a seguinte forma:

$$
N_1\left(x, y\right) = \frac{1}{2A}\left[\left(x_2 y_3 - x_3 y_2\right) + \left(y_2 - y_3\right) x + \left(x_3 - x_2\right) y\right]
$$

$$
N_2\left(x, y\right) = \frac{1}{2A}\left[\left(y_1 x_3 - y_3 x_1\right) + \left(y_3 - y_1\right) x + \left(x_1 - x_3\right) y\right]
$$

$$
N_3\left(x, y\right) = \frac{1}{2A}\left[\left(x_1 y_2 - x_2 y_1\right) + \left(y_1 - y_2\right) x + \left(x_2 - x_1\right) y\right]
$$

Sendo A a área do triiângulo, dada por:

$$
A = \frac{1}{2} \times \det \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}
$$

As equações acima para N_i podem ser resumidas em:

$$
N_i\left(x, y\right) = \frac{1}{2A}\left[\left(x_j y_k - x_k y_j\right) + \left(y_j - y_k\right) x + \left(x_k - x_j\right) y\right]
$$

Sendo,

$$
\begin{cases} i & = & 1, & 2, & 3; \\ j & = & 2, & 3, & 1; \\ k & = & 3, & 1, & 2; \end{cases}
$$

A matriz $\mathbf{N}$ é, então, formada:

$$
\mathbf{N} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix}
$$

Mas $\vec{\varepsilon} = \mathbf{L}\vec{u}$. Como $\vec{u} = \mathbf{N}\vec{u}_i$, tem-se

$$
\vec{\varepsilon} = \mathbf{LN}\vec{u}_i
$$

Fazendo-se $\mathbf{B} = \mathbf{LN}$, tem-se:

$$\mathbf{B} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix}$$

Que resulta em:

$$\mathbf{B} = \begin{bmatrix} (y_2 - y_3) & 0 & (y_3 - y_1) & 0 & (y_1 - y_2) & 0 \\ 0 & (x_3 - x_2) & 0 & (x_1 - x_3) & 0 & (x_2 - x_1) \\ (x_3 - x_2) & (y_2 - y_3) & (x_1 - x_3) & (y_3 - y_1) & (x_2 - x_1) & (y_1 - y_2) \end{bmatrix}$$

Tem-se que:

$$\mathbf{K}_e = t\mathbf{B}^T\mathbf{D}\mathbf{B} \times A$$

No EPT, t é a espessura do elemento (chapa); no EPD, $t = 1$.

[10]:
```python
x_1,x_2,x_3 = sp.symbols('x_1,x_2,x_3')
y_1,y_2,y_3 = sp.symbols('y_1,y_2,y_3')
B = sp.Matrix([
    [(y_2-y_3), 0, (y_3-y_1), 0, (y_1-y_2), 0],
    [0, (x_3-x_2), 0, (x_1-x_3), 0, (x_2-x_1)],
    [(x_3-x_2), (y_2-y_3), (x_1-x_3), (y_3-y_1), (x_2-x_1), (y_1-y_2)]
])
B
```

[10]:

$$\begin{bmatrix} y_2 - y_3 & 0 & -y_1 + y_3 & 0 & y_1 - y_2 & 0 \\ 0 & -x_2 + x_3 & 0 & x_1 - x_3 & 0 & -x_1 + x_2 \\ -x_2 + x_3 & y_2 - y_3 & x_1 - x_3 & -y_1 + y_3 & -x_1 + x_2 & y_1 - y_2 \end{bmatrix}$$

7 Exemplo 1

Calcular como EPD e EPT uma estrutura de área retangular de largura $75\ mm$ em x e altura $50\ mm$ em y. Considerar apoio de segundo gênero nos pontos $(0,\ 0)$ e $(0,\ 50)$ e uma carga pontual vertical de $4450\ N$ para baixo no ponto $(75,\ 50)$.

Dados:

$$\begin{cases} t & = & 13\ mm \\ E & = & 207\ GPa \\ \nu & = & 0,25 \end{cases}$$

[11]:
```python
geral = pd.DataFrame({
    "t": [13],
    "E": [207],
    "nu": [0.25]
})
```

A malha considerada será a seguinte:

[12]:
```python
pd.DataFrame({
    "Ponto": [1,2,3,4],
    "x": ["(75,","(75,","(0,","(0,"],
    "y": ["0)","75)","75)","0)"],
})
```

[12]:

	Ponto	x	y
0	1	(75,	0)
1	2	(75,	75)
2	3	(0,	75)
3	4	(0,	0)

Os pontos dos dois elementos utilizados são:

[13]:
```python
pd.DataFrame({
    "Elemento": [1, 2],
    "Pontos": [[1, 2, 4], [3, 4, 2]]
})
```

[13]:

	Elemento	Pontos
0	1	[1, 2, 4]
1	2	[3, 4, 2]

7.1 Cálculo Como EPT

No EPT, temos:

$$\mathbf{D} =$$

[14]:
```python
def Dept(data):
    E = data["E"][0]
    nu = data["nu"][0]
    D_var = (E_var/(1-nu_var**2))*sp.
↪Matrix([[1,nu_var,0],[nu_var,1,0],[0,0,(1-nu_var)/2]])
    return D_var.subs(E_var,E).subs(nu_var,nu)
```

7.2 Cálculo como EPD

No EPD, temos:

$$\mathbf{D} =$$

```
[15]: def Depd(data):
          E = data["E"][0]
          nu = data["nu"][0]
          D_var = (E_var/(1-nu_var**2))*sp.
       ↪Matrix([[1,nu_var,0],[nu_var,1,0],[0,0,(1-nu_var)/2]])
          return D_var.subs(E_var,E/(1-nu**2)).subs(nu_var,nu/(1-nu))
```

7.3 Cálculo da Área

$$A =$$

```
[16]: def area(data):
          A = np.array([
              [1,1,1],
              [data["x"][0],data["x"][1],data["x"][2]],
              [data["y"][0],data["y"][1],data["y"][2]]
          ])
          return np.linalg.det(A)/2
```

7.4 Cálculo de B

$$\mathbf{B} =$$

```
[17]: def B(data):
          x_1,x_2,x_3 = sp.symbols('x_1,x_2,x_3')
          y_1,y_2,y_3 = sp.symbols('y_1,y_2,y_3')
          A = sp.symbols('A')
          B = (1/(2*A))*sp.Matrix([
              [(y_2-y_3), 0, (y_3-y_1), 0, (y_1-y_2), 0],
              [0, (x_3-x_2), 0, (x_1-x_3), 0, (x_2-x_1)],
              [(x_3-x_2), (y_2-y_3), (x_1-x_3), (y_3-y_1), (x_2-x_1), (y_1-y_2)]
          ])
          return B.subs(x_1, data["x"][0]).subs(x_2, data["x"][1]).subs(x_3,␣
       ↪data["x"][2]).subs(y_1, data["y"][0]).subs(y_2, data["y"][1]).subs(y_3,␣
       ↪data["y"][2]).subs(A, area(data))
```

7.5 Cálculo de $\mathbf{K}_e$

$$\mathbf{K}_e =$$

```
[18]: def K_e(data,constantes,tipo="EPT"):
          B_el = B(data)
          if tipo == "EPT":
              D = Dept(constantes)
```

```python
        t = constantes["t"][0]
    else:
        D = Depd(constantes)
        t = 1
    return t * B_el.transpose() @ D @ B_el * area(data)
```

7.6 Elemento 1

[19]:
```python
el1 = pd.DataFrame({
    "Incidência": [1,2,3],
    "Nó": [1,2,4],
    "x": [75,75,0],
    "y": [0,50,0]
})
el1
```

[19]:

	Incidência	Nó	x	y
0	1	1	75	0
1	2	2	75	50
2	3	4	0	0

[20]:
```python
K_1 = K_e(el1,geral,"EPT")
K_1
```

[20]:

$$\begin{bmatrix} 1764.1 & -897.0 & -807.3 & 358.8 & -956.8 & 538.2 \\ -897.0 & 2511.6 & 538.2 & -2152.8 & 358.8 & -358.8 \\ -807.3 & 538.2 & 807.3 & 0 & 0 & -538.2 \\ 358.8 & -2152.8 & 0 & 2152.8 & -358.8 & 0 \\ -956.8 & 358.8 & 0 & -358.8 & 956.8 & 0 \\ 538.2 & -358.8 & -538.2 & 0 & 0 & 358.8 \end{bmatrix}$$

7.7 Elemento 2

[21]:
```python
el2 = pd.DataFrame({
    "Incidência": [1,2,3],
    "Nó": [3,4,2],
    "x": [0,0,75],
    "y": [50,0,50]
})
el2
```

[21]:

	Incidência	Nó	x	y
0	1	3	0	50
1	2	4	0	0
2	3	2	75	50

[22]:
```python
K_2 = K_e(el2,geral,"EPT")
K_2
```

[22]:

$$\begin{bmatrix} 1764.1 & -897.0 & -807.3 & 358.8 & -956.8 & 538.2 \\ -897.0 & 2511.6 & 538.2 & -2152.8 & 358.8 & -358.8 \\ -807.3 & 538.2 & 807.3 & 0 & 0 & -538.2 \\ 358.8 & -2152.8 & 0 & 2152.8 & -358.8 & 0 \\ -956.8 & 358.8 & 0 & -358.8 & 956.8 & 0 \\ 538.2 & -358.8 & -538.2 & 0 & 0 & 358.8 \end{bmatrix}$$

7.8 Montagem de $\mathbf{K}_g$

$$\mathbf{K}_g =$$

[23]:
```python
def K_g(n,datalist,constantes,tipo="EPT"):
    K = np.zeros([2*n,2*n])
    for el in datalist:
        K_loc = K_e(el,constantes,tipo)
        for k in range(3):
            for l in range(3):
                i = el["Nó"][el["Incidência"][k] - 1] - 1
                j = el["Nó"][el["Incidência"][l] - 1] - 1
                K[2*i:2*i+2,2*j:2*j+2] += K_loc[2*k:2*k+2,2*l:2*l+2]
    return K
```

Assim, temos a seguinte Matriz de Rigidez Global para o Exemplo:

[24]:
```python
K = K_g(4,[el1,el2],geral,"EPT")
K
```

[24]:
```
array([[ 1764.1,  -897. ,  -807.3,   358.8,     0. ,     0. ,  -956.8,
          538.2],
       [ -897. ,  2511.6,   538.2, -2152.8,     0. ,     0. ,   358.8,
         -358.8],
       [ -807.3,   538.2,  1764.1,     0. ,  -956.8,   358.8,     0. ,
         -897. ],
       [  358.8, -2152.8,     0. ,  2511.6,   538.2,  -358.8,  -897. ,
            0. ],
       [    0. ,     0. ,  -956.8,   538.2,  1764.1,  -897. ,  -807.3,
          358.8],
       [    0. ,     0. ,   358.8,  -358.8,  -897. ,  2511.6,   538.2,
        -2152.8],
       [ -956.8,   358.8,     0. ,  -897. ,  -807.3,   538.2,  1764.1,
            0. ],
       [  538.2,  -358.8,  -897. ,     0. ,   358.8, -2152.8,     0. ,
         2511.6]])
```

7.9 Vetor de Cargas Nodais

[25]:
```
f = np.array([0,np.nan,0,-4450,np.nan,np.nan,np.nan,np.nan])
f
```

[25]:
```
array([    0.,    nan,     0., -4450.,    nan,    nan,    nan,    nan])
```

7.10 Vetor de Deslocamentos

[26]:
```
u = np.array([np.nan,0,np.nan,np.nan,0,0,0,0])
u
```

[26]:
```
array([nan,  0., nan, nan,  0.,  0.,  0.,  0.])
```

7.11 Redução do Sistema

A função abaixo recebe um sistema com valores conhecidos, reduz o sistema, resolve o sistema reduzido e, em seguida, restaura a solução devolvendo os valores conhecidos.

[27]:
```
def solve(A,x,b):
    n = x.shape[0]
    sol = x.copy()
    b_r = b.copy()
    A_r = A.copy()
    for i in range(n):
        if not np.isnan(x[i]):
            b_r -= x[i]*A[:,i]
    for i in range(n):
        k = n - 1 - i
        if np.isnan(b[k]):
            A_r = np.delete(A_r,k,0)
            A_r = np.delete(A_r,k,1)
            b_r = np.delete(b_r,k,0)
    x_r = np.linalg.solve(A_r,b_r)
    k = 0
    for i in range(n):
        if np.isnan(sol[i]):
            sol[i] = x_r[k]
            k += 1
    return sol
```

7.12 Solução do Vetor de Deslocamentos

[28]:
```
u = solve(K,u,f)
u
```

[28]:
```
array([ 0.47321279,  0.        ,  0.216555  , -1.83938077,  0.        ,
        0.        ,  0.        ,  0.        ])
```

7.13 Cálculo das Deformações

Com o sistema resolvido, utilizaremos uma função análoga aos índices globais utilizados na matriz $\mathbf{K}_g$ para encontrar o vetor de deslocamentos do elemento.

$$\vec{u}_e =$$

[29]:
```
def u_e(el,u):
    u_el = np.zeros(6)
    for k in range(3):
        i = el["Nó"][el["Incidência"][k] - 1] - 1
        u_el[2*k:2*k+2] = u[2*i:2*i+2]
    return u_el
```

Assim, podemos calcular as deformações.

$$\vec{\varepsilon}_e =$$

[30]:
```
def epsilon_e(el,u):
    B_l = B(el)
    u_el = u_e(el,u)
    return B_l @ u_el
```

7.14 Cálculo das Tensões

A partir das deformações no elemento, podemos calcular as tensões.

$$\vec{\sigma}_e =$$

[31]:
```
def sigma_e(data,u,constantes,tipo="EPT"):
    if tipo == "EPT":
        D = Dept(constantes)
    else:
        D = Depd(constantes)
    return D @ epsilon_e(data,u)
```

7.15 Tensões no Elemento 1

[32]:
```
s1 = sigma_e(el1,u,geral)
s1
```

```
[32]: array([-0.637537930920125, -7.77442087983153, -0.425025287280084],
      dtype=object)
```

Ou seja,

$$\begin{cases} \sigma_x^{(1)} & = & -0.63753 & MPa \\ \sigma_y^{(1)} & = & -7.77442 & MPa \\ \gamma_{xy}^{(1)} & = & -0.42502 & MPa \end{cases}$$

7.16 Tensões no Elemento 2

```
[33]: s2 = sigma_e(el2,u,geral)
s2
```

```
[33]: array([0.637537930920125, 0.159384482730031, -2.03067637256040],
      dtype=object)
```

Ou seja,

$$\begin{cases} \sigma_x^{(2)} & = & 0.63753 & MPa \\ \sigma_y^{(2)} & = & 0.15938 & MPa \\ \gamma_{xy}^{(2)} & = & -2.03067 & MPa \end{cases}$$

8 Exemplo 2

Livro Método dos Elementos Finitos,, Eloy:

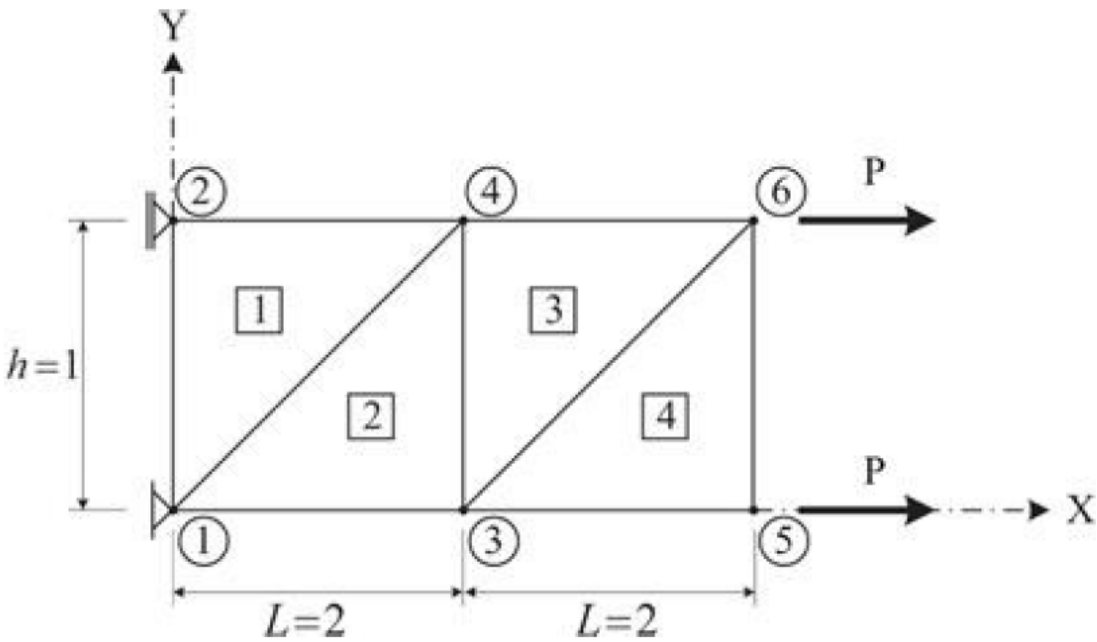
8.1 4.5.2 Exemplo Da Barra Tracionada, Elemento CST

8.1.1 Dados do problema:

- Módulo de elasticidade, $E = 20000$;
- Coeficiente de Poisson, $\nu = 0,2$;
- Carga $P = 10$
- Largura da seção transversal, $b = 1$;
- Altura da seção transversal, $h = 1$;
- Comprimento da barra, $2L = 4$.

8.1.2 Dados do modelo de elementos finitos:

- $nnodes = 6$ (número de nós);
- $nelem = 4$ (número de elementos)



8.2 Solução

Dados gerais:

[34]:
```
dados = pd.DataFrame({
    "t": [1],
    "E": [20000],
    "nu": [0.2]
})
dados
```

[34]:

	t	E	nu
0	1	20000	0.2

Temos, portanto, a seguinte malha considerada:

[35]:
```
pd.DataFrame({
    "Ponto": [1,2,3,4,5,6],
    "x": ["(0,","(0,","(2,","(2,","(4,","(4,"],
    "y": ["0)","1)","0)","1)","0)","1)"],
})
```

[35]:

	Ponto	x	y
0	1	(0,	0)
1	2	(0,	1)
2	3	(2,	0)
3	4	(2,	1)
4	5	(4,	0)

```
5      6  (4,  1)
```

Os pontos dos quatro Elementos utilizados são:

[36]:
```
pd.DataFrame({
    "Elemento": [1, 2, 3, 4],
    "Pontos": [[1, 4, 2], [1, 3, 4], [3,6,4], [3,5,6]]
})
```

[36]:

	Elemento	Pontos
0	1	[1, 4, 2]
1	2	[1, 3, 4]
2	3	[3, 6, 4]
3	4	[3, 5, 6]

8.3 Elemento 1

[37]:
```
el1 = pd.DataFrame({
    "Incidência": [1,2,3],
    "Nó": [1,4,2],
    "x": [0,2,0],
    "y": [0,1,1]
})
el1
```

[37]:

	Incidência	Nó	x	y
0	1	1	0	0
1	2	4	2	1
2	3	2	0	1

8.4 Elemento 2

[38]:
```
el2 = pd.DataFrame({
    "Incidência": [1,2,3],
    "Nó": [1,3,4],
    "x": [0,2,2],
    "y": [0,0,1]
})
el2
```

[38]:

	Incidência	Nó	x	y
0	1	1	0	0
1	2	3	2	0
2	3	4	2	1

8.5 Elemento 3

[39]:
```
el3 = pd.DataFrame({
    "Incidência": [1,2,3],
    "Nó": [3,6,4],
    "x": [2,4,2],
    "y": [0,1,1]
})
el3
```

[39]:

	Incidência	Nó	x	y
0	1	3	2	0
1	2	6	4	1
2	3	4	2	1

8.6 Elemento 4

[40]:
```
el4 = pd.DataFrame({
    "Incidência": [1,2,3],
    "Nó": [3,5,6],
    "x": [2,4,4],
    "y": [0,0,1]
})
el4
```

[40]:

	Incidência	Nó	x	y
0	1	3	2	0
1	2	5	4	0
2	3	6	4	1

8.7 Vetor de Cargas Nodais

$$\vec{f} =$$

[41]:
```
f = np.array([np.nan,np.nan,np.nan,0,0,0,0,0,10,0,10,0])
```

8.8 Vetor de Deslocamentos

$$\vec{u} =$$

[42]:
```
u = np.array([0,0,0,np.nan,np.nan,np.nan,np.nan,np.nan,np.nan,np.nan,np.nan,np.
↪nan])
```

8.9 Matriz de Rigidez Global

Cálculo como EPT:

$$\mathbf{K}_g =$$

[43]:
```
K = K_g(6,[el1,el2,el3,el4],dados,"EPT")
```

8.10 Solução do Sistema

[44]:
```
u = solve(K,u,f)
```

8.11 Tensões nos Elementos

8.11.1 Elemento 1

[45]:
```
s1 = sigma_e(el1,u,dados)
s1
```

[45]:
```
array([20.0000000000000, -7.99360577730113e-15, -1.59242194083808e-14],
      dtype=object)
```

Ou seja,

$$\begin{cases} \sigma_x^{(1)} & = & 19 & MPa \\ \sigma_y^{(1)} & = & -1 & MPa \\ \gamma_{xy}^{(1)} & = & 0 & MPa \end{cases}$$

8.11.2 Elemento 2

[46]:
```
s2 = sigma_e(el2,u,dados)
s2
```

[46]:
```
array([20.0000000000000, 1.77635683940025e-15, 3.61400724161835e-15],
      dtype=object)
```

Ou seja,

$$\begin{cases} \sigma_x^{(1)} & = & 20 & MPa \\ \sigma_y^{(1)} & = & 0 & MPa \\ \gamma_{xy}^{(1)} & = & 0 & MPa \end{cases}$$

8.11.3 Elemento 3

[47]:
```
s3 = sigma_e(el1,u,dados)
s3
```

[47]:
```
array([20.0000000000000, -7.99360577730113e-15, -1.59242194083808e-14],
      dtype=object)
```

Ou seja,

$$\begin{cases} \sigma_x^{(1)} & = & 20 & MPa \\ \sigma_y^{(1)} & = & 0 & MPa \\ \gamma_{xy}^{(1)} & = & 0 & MPa \end{cases}$$

8.11.4 Elemento 4

[48]:
```
s4 = sigma_e(el4,u,dados)
s4
```

[48]:
```
array([20.0000000000000, 2.66453525910038e-15, 0], dtype=object)
```

Ou seja,

$$\begin{cases} \sigma_x^{(1)} & = & 20 & MPa \\ \sigma_y^{(1)} & = & 0 & MPa \\ \gamma_{xy}^{(1)} & = & 0 & MPa \end{cases}$$

8.12 Cálculo como EPD

[49]:
```
u = np.array([0,0,0,np.nan,np.nan,np.nan,np.nan,np.nan,np.nan,np.nan,np.nan,np.
 ↪nan])
K = K_g(6,[el1,el2,el3,el4],dados,"EPD")
u = solve(K,u,f)
s1 = sigma_e(el1,u,dados)
s2 = sigma_e(el2,u,dados)
s3 = sigma_e(el3,u,dados)
s4 = sigma_e(el4,u,dados)
display(s1)
display(s2)
display(s3)
display(s4)
```

```
array([19.0000000000000, -1.00000000000000, -4.96925995722523e-15],
      dtype=object)
```

```
array([19.0000000000000, -0.999999999999996, 0], dtype=object)
```

```
array([19.0000000000000, -0.999999999999997, -3.38813178901720e-15],
      dtype=object)
```

```
array([19.0000000000000, -0.999999999999999, -3.61400724161835e-15],
      dtype=object)
```

8.13 Elemento 1

$$\begin{cases} \sigma_x^{(1)} & = & 19 & MPa \\ \sigma_y^{(1)} & = & -1 & MPa \\ \gamma_{xy}^{(1)} & = & 0 & MPa \end{cases}$$

8.14 Elemento 2

$$\begin{cases} \sigma_x^{(1)} & = & 19 & MPa \\ \sigma_y^{(1)} & = & -1 & MPa \\ \gamma_{xy}^{(1)} & = & 0 & MPa \end{cases}$$

8.15 Elemento 3

$$\begin{cases} \sigma_x^{(1)} & = & 19 & MPa \\ \sigma_y^{(1)} & = & -1 & MPa \\ \gamma_{xy}^{(1)} & = & 0 & MPa \end{cases}$$

8.16 Elemento 4

$$\begin{cases} \sigma_x^{(1)} & = & 19 & MPa \\ \sigma_y^{(1)} & = & -1 & MPa \\ \gamma_{xy}^{(1)} & = & 0 & MPa \end{cases}$$