MAC008 - Introdução ao Método dos Elementos Finitos - Lista 1

Antonio José de Medeiros Filho - 201965502B

1 Exercício 1

Utilize o método da projeção para determinar a aproximação da função $f = sen(z\pi)$ no intervalo $-2 < x < 2$ utilizando as funções 1, x^2, x^3, x^4, x^5. Considere $z = 1 + \frac{0.5N}{19}$. Apresente os gráficos da aproximação junto com a função dada.

1.1 Resolução

A função que iremos aproximar é da forma:

$$f = sen(z\pi x)$$

Temos que:

$$z = 1 + \frac{0.5N}{19} N = 0 \Rightarrow z = 1$$

Portanto:

$$f = sen(\pi x)$$

1.1.1 Calculando Coeficientes

Para calcular os coeficientes devemos resolver o sistema linear na forma:

$$A\mathbf{c} = \mathbf{b}$$

onde:

- A é a matriz tal que $A_{ij} = \langle \phi_i, \phi_j \rangle$;
- $\mathbf{b}$ é o vetor dado por $\langle f, \phi_i \rangle$;
- $\mathbf{c}$ é o vetor de coeficientes.

[51]:
```
import sympy as sp
import numpy as np

def f(x):
    return sp.sin(np.pi * x)
```

```
x = sp.symbols('x')

A = sp.zeros(6,6)
b = sp.zeros(6,1)
c = sp.zeros(6,1)

for i in range(6):
    b[i] = sp.integrate(f(x) * (x**i), (x, -2, 2))
    for j in range(6):
        A[i,j] = sp.integrate((x**i) * (x**j), (x, -2, 2))

c = A.inv() * b
display(c)
```

$$\begin{bmatrix} 0 \\ 2.0081971637608 \\ 0 \\ -2.23285228459836 \\ 0 \\ 0.453821069426289 \end{bmatrix}$$

1.1.2 Aproximação

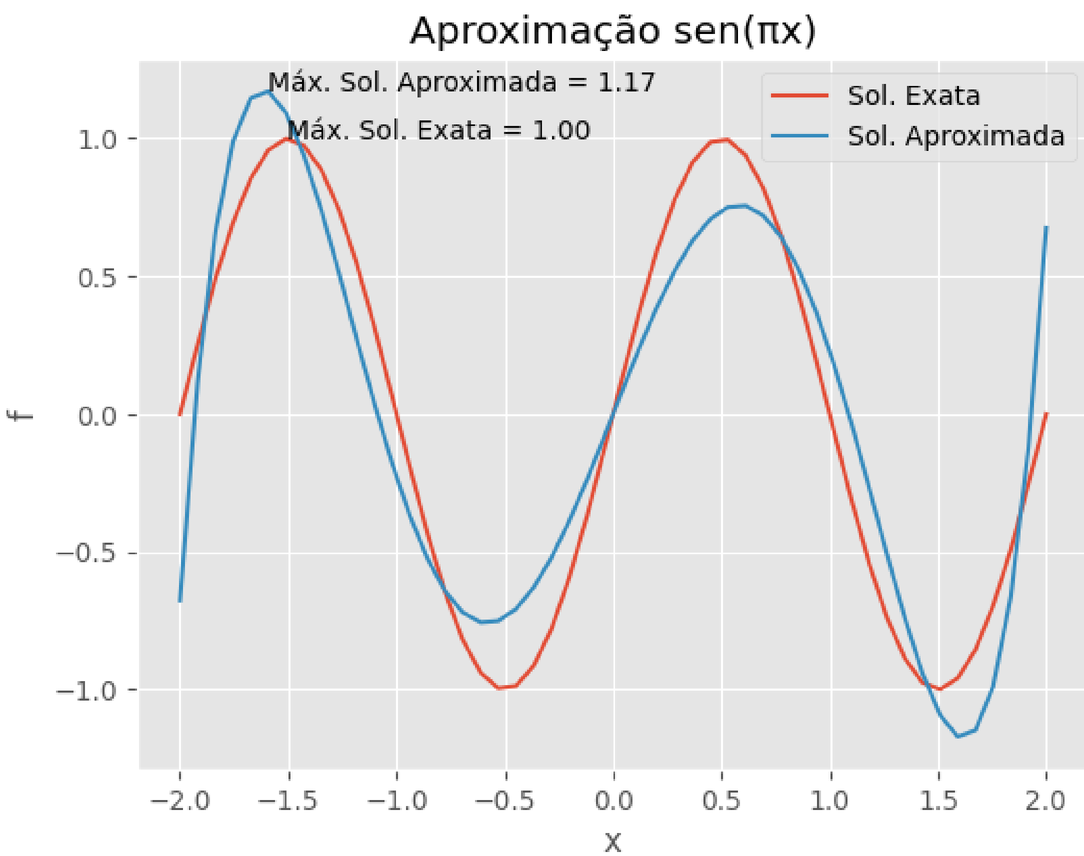
[57]:
```
def sol_approx(x):
    return sum(c[i] * x**i for i in range(6))

x, i = sp.symbols('x i')
display(sum(c[i] * x**i for i in range(6)))
```

$0.453821069426289x^5 - 2.23285228459836x^3 + 2.0081971637608x$

1.2 Resultados

[55]:
```
import numpy as np
from matplotlib import pyplot as plt
plt.style.use('ggplot')

def sol_exata(x):
    return np.sin(np.pi * x)

x = np.linspace(-2, 2)

plt.plot(x, sol_exata(x), label='Sol. Exata')
max_y_exata = max(sol_exata(x))
max_x_exata = x[np.where(sol_exata(x) == max_y_exata)]
plt.annotate(f'Máx. Sol. Exata = {max_y_exata:.2f}', (max_x_exata,max_y_exata))

plt.plot(x, sol_approx(x), label='Sol. Aproximada')
```

```
max_y_approx = max(sol_approx(x))
max_x_approx = x[np.where(sol_approx(x) == max_y_approx)]
plt.annotate(f'Máx. Sol. Aproximada = {max_y_approx:.2f}', (max_x_approx,␣
↪max_y_approx))

plt.title('Aproximação sen( x)')
plt.xlabel('x')
plt.ylabel('f')
plt.legend()
plt.show()
```



2 Exercício 2

Dada a função $f(x) = -1 - \frac{N}{19}\ \forall\ -\pi < x < 0$ e $f(x) = 1 + \frac{N}{19}\ \forall\ 0 < x < \pi$ use o método da projeção para determinar as componentes desta função neste intervalo $-\pi < x < \pi$ na "base" composta pelas funções 1, *sen kx*, *cos kx* com k = 1 até 5.

2.1 Resolução

Temos que $N = 0$, logo:

$$f(x) = -1, \forall -\pi < x < 0$$

e,

$$f(x) = 1, \forall 0 < x < \pi$$

[4]:

```
import numpy as np
from matplotlib import pyplot as plt
plt.style.use('ggplot')

def f(x):
    if type(x) == int:
        if x >= np.pi:
            return 1
        return -1

    y = []
    for i in x:
        if i >= 0:
            y.append(1)
        else:
            y.append(-1)
    return y

def proj(f, x, k):
    c = np.zeros(2 * k + 1)
    c[0] = np.trapz(f(x), x) / (2*np.pi)

    for i in range(1, k+1):
        c[i] = np.trapz(f(x) * np.sin(i*x), x) / np.pi
        c[k + i] = np.trapz(f(x) * np.cos(i*x), x) / np.pi

    return c

def approx(x, c, k):
    y = np.full_like(x, c[0]/2)

    for i in range(1, k+1):
        y += c[i] * np.sin(i*x) + c[k+i] * np.cos(i*x)

    return y

def plot(x, k):
```

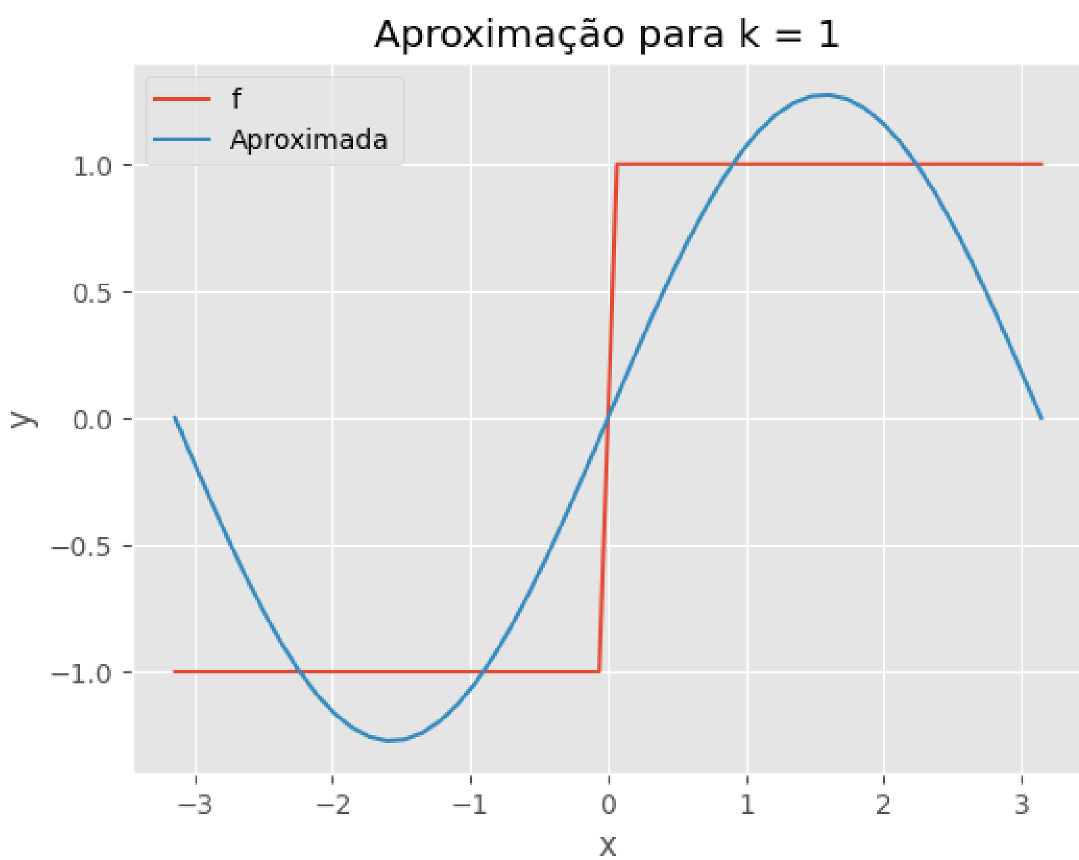
```
    coef = proj(f, x, k)
    print('Coeficientes: ', coef)
    plt.plot(x, f(x), label='f')
    plt.plot(x, approx(x, coef, k), label='Aproximada')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Aproximação para k = {k}')
    plt.legend()

x = np.linspace(-np.pi, np.pi)
```

2.2 Resultados

2.2.1 k = 1

[5]:
```
plot(x, 1)
```

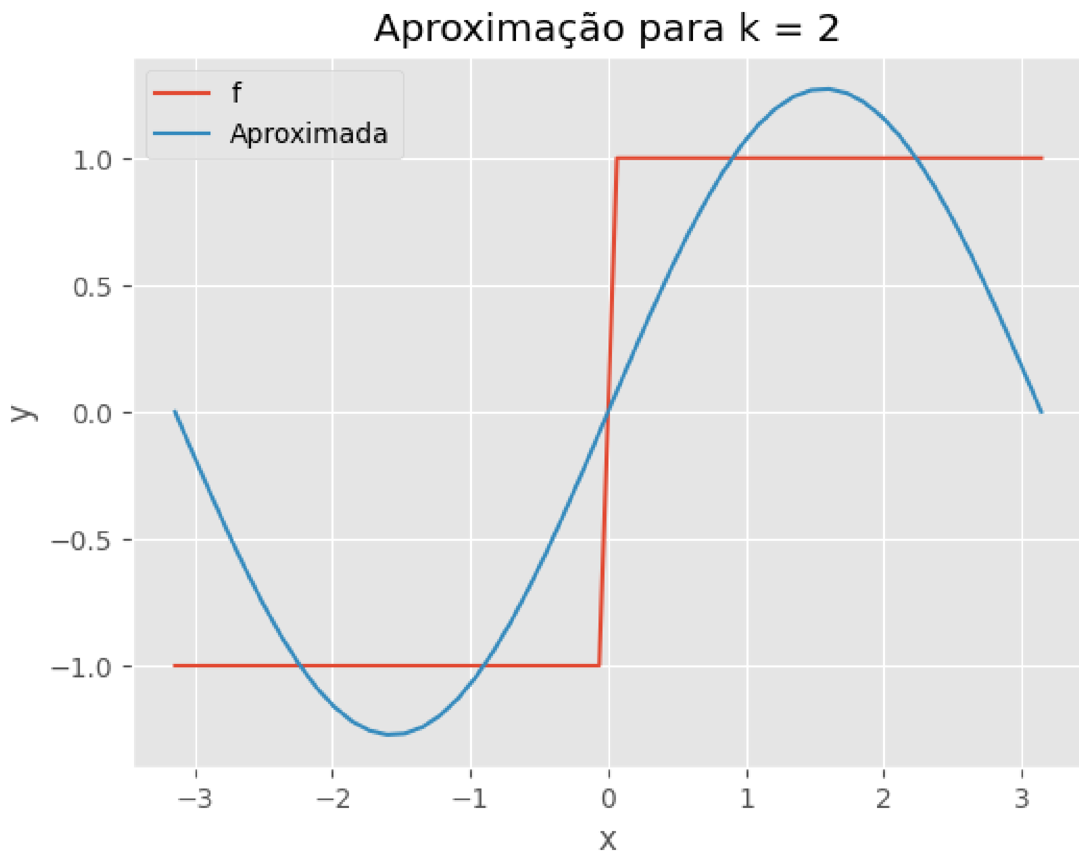
```
Coeficientes:  [7.06789929e-17 1.27280336e+00 1.23688238e-16]
```



2.2.2 k = 2

[6]:
```
plot(x, 2)
```

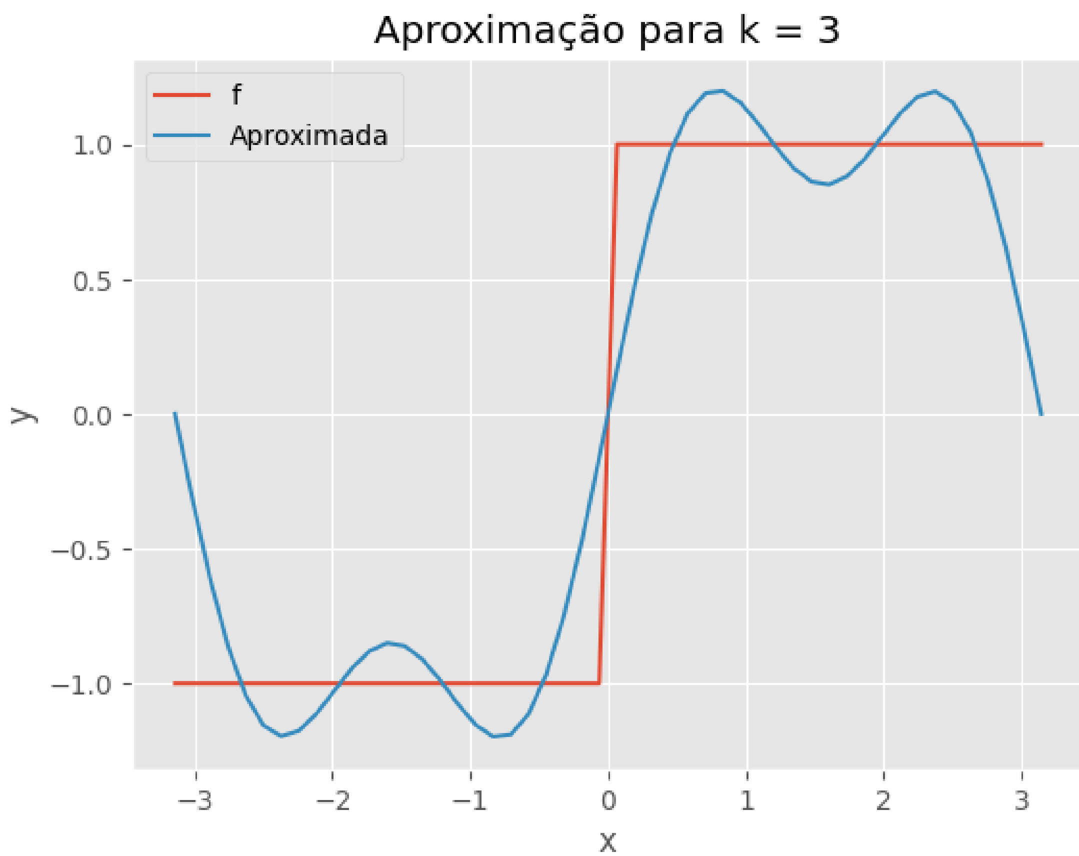
```
Coeficientes:  [7.06789929e-17 1.27280336e+00 2.62049511e-03 1.23688238e-16
 1.14853363e-16]
```



2.2.3 k = 3

[7]:
```
plot(x, 3)
```

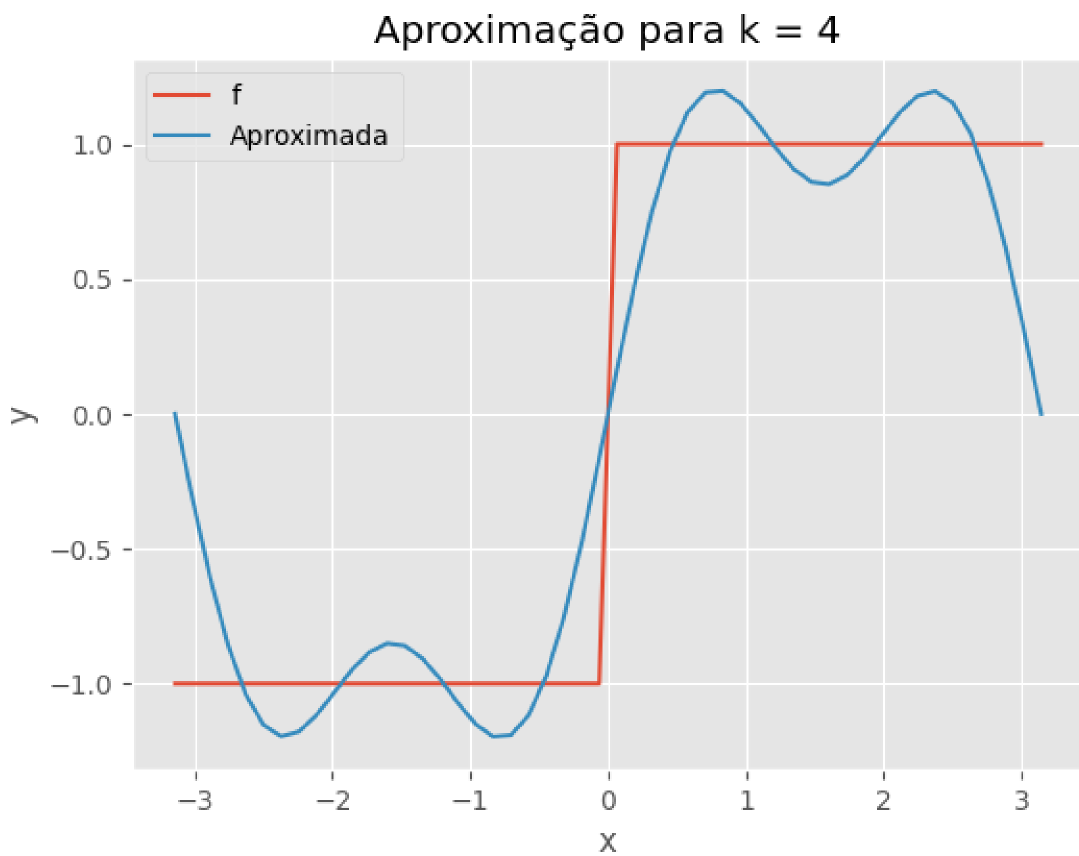
```
Coeficientes:  [7.06789929e-17 1.27280336e+00 2.62049511e-03 4.23103922e-01
 1.23688238e-16 1.14853363e-16 1.76697482e-16]
```



2.2.4 k = 4

[8]:
```
plot(x, 4)
```

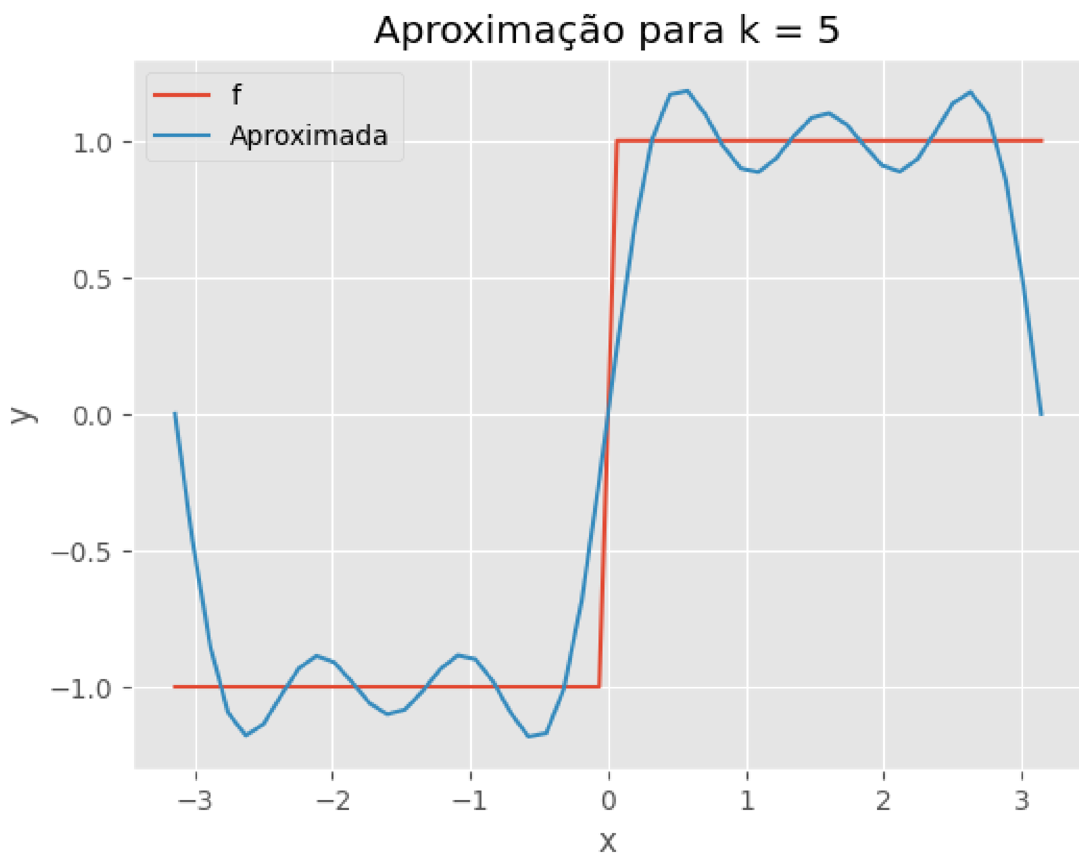
```
Coeficientes:  [7.06789929e-17 1.27280336e+00 2.62049511e-03 4.23103922e-01
 5.26268254e-03 1.23688238e-16 1.14853363e-16 1.76697482e-16
 1.32523112e-16]
```



2.2.5 k = 5

[9]:
```
plot(x, 5)
```

```
Coeficientes:  [7.06789929e-17 1.27280336e+00 2.62049511e-03 4.23103922e-01
 5.26268254e-03 2.52463412e-01 1.23688238e-16 1.14853363e-16
 1.76697482e-16 1.32523112e-16 2.65046223e-16]
```



3 Exercício 3

Dada a barra a esforço axial com secção constante ($b \times h$) engastada e livre com uma carga distribuída dada por q pede-se calcular o deslocamento em sua extremidade livre utilizando o MEF com aproximações linear e quadrática utilizando, respectivamente, 1 elemento e 5 elementos. Apresente a dedução da matriz do elemento quadrático para este caso. Na solução numérica indique claramente a numeração dos nós e dos elementos utilizados em sua discretização e apresentando a numeração dos nós e dos elementos utilizados em sua discretização e apresentado a matriz global e o campo de deslocamento aproximado comparado com a solução exata do problema. Dados:

- $b = 0,10m$
- $E = 10^8 \cdot T(Pa)$
- $h = 0,5 \cdot T(m)$
- $q(x) = 10 \cdot T(kN/m)$
- $l = 10 \cdot T(m)$

Com $T = \frac{20-N}{19}$

3.1 Resolução

3.1.1 Substituindo N

Inicialmente, substituiremos os valores de N e T:

$$N = 0 \Rightarrow T = \frac{20}{19}$$

Logo, temos que:

- $b = 0,10m$
- $E = 10^8 \cdot \frac{20}{19}(Pa)$
- $h = 0,5 \cdot \frac{20}{19}(m)$
- $q(x) = 10 \cdot \frac{20}{19}(kN/m)$
- $l = 10 \cdot \frac{20}{19}(m)$

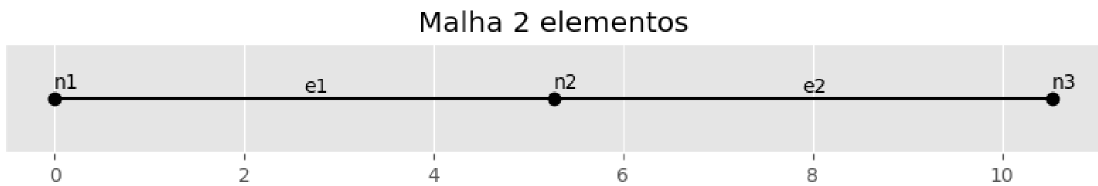
3.1.2 Equação do problema

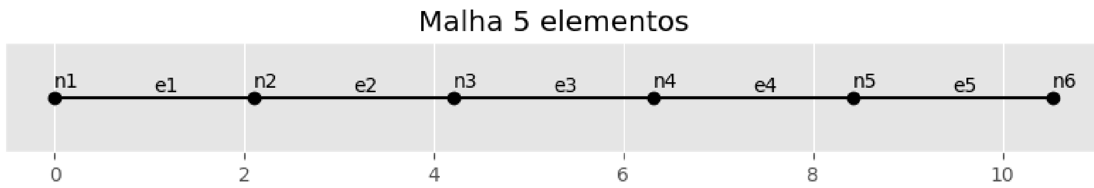
$$-EA\frac{\partial^2 u(x)}{\partial x^2} = q(x)$$

Onde $0 < x < L$, $A = bh$ é a área transversal e u é o vetor deslocamento.

3.1.3 Malhas dos elementos e nós

[7]:
```python
from matplotlib import pyplot as plt
plt.style.use('ggplot')
L=200/19
for n in [2, 5]:
    plt.figure(figsize=(10,1))
    plt.plot([i*L/n for i in range(n+1)],
             [0 for i in range(n+1)],
             '-o',
             color='black'
    )
    for i in range(n):
        plt.annotate(f'n{i+1}', (i*L/n,0.01))
        plt.annotate(f'e{i+1}', (i*L/n + (L/n)/2,0.005))
    plt.annotate(f'n{n+1}', (L, 0.01))
    plt.gca().get_yaxis().set_visible(False)
    plt.title(f'Malha {n} elementos')
    plt.show()
```





3.1.4 Condições de Contorno

Do enunciado, temos que a barra possui uma extremidade engastada e outra livre, portanto:

$$u(0) = 0$$

3.1.5 Aproximação Linear (1 elemento)

Fazendo uma aproximação linear, temos:

$u = u_1\phi_1 + u_2\phi_2$

Sendo as funções bases de Lagrange:

$$\phi_1 = 1 - \frac{x}{h}$$

$$\phi_2 = \frac{x}{h}$$

Calculando Matriz de Rigidez para cada elemento

$$K^i = EA \begin{bmatrix} \int_0^h \phi_1'\phi_1' dx & \int_0^h \phi_1'\phi_2' dx \\ \int_0^h \phi_2'\phi_1' dx & \int_0^h \phi_2'\phi_2' dx \end{bmatrix}$$

[36]:
```
import sympy as sp

E, A, q, x, h = sp.symbols('E A q x h')

phi_1 = 1 - x/h
phi_2 = x/h

integral_1 = sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_1, x), (x, 0, h))
integral_2 = sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_2, x), (x, 0, h))
integral_3 = sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_1, x), (x, 0, h))
integral_4 = sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_2, x), (x, 0, h))
```

```
K = E * A * sp.Matrix([[integral_1, integral_2], [integral_3, integral_4]])
display((E * A) / h, sp.Matrix([[integral_1, integral_2], [integral_3,␣
↪integral_4]]).subs({h: 1}))
```

$$\frac{AE}{h}$$

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Calculando Vetor Fonte por elemento

$$f^i = \begin{bmatrix} \int_0^h \phi_1 q(x) dx \\ \int_0^h \phi_2 q(x) dx \end{bmatrix}$$

[37]:
```
f = sp.symbols('f')

f = sp.Matrix([[sp.integrate(phi_1 * q, (x, 0, h))], [sp.integrate(phi_2 * q,␣
↪(x, 0, h))]])

display(f)
```

$$\begin{bmatrix} \frac{hq}{2} \\ \frac{hq}{2} \end{bmatrix}$$

Substituição dos valores

Para o caso linear, como temos apenas 1 elemento, a Matriz Global de Rigidez e o Vetor Fonte Global serão os mesmos já calculados:

[38]:
```
T = 20/19
L = 10 * T
n_elementos = 1

K_lin = K.subs({
    A: 0.1 * 0.5 * T,
    E: (10**8) * T,
    h: L/n_elementos
})

display(K_lin)
```

$$\begin{bmatrix} 526315.789473684 & -526315.789473684 \\ -526315.789473684 & 526315.789473684 \end{bmatrix}$$

[39]:
```
f_lin = f.subs({
    h: L/n_elementos,
    q: 10 * T
})
```

```
display(f_lin)
```

$$\begin{bmatrix} 55.4016620498615 \\ 55.4016620498615 \end{bmatrix}$$

Aplicando Condições de Contorno de Dirichlet ($u(0) = 0$)

Conhecendo o valor em $u(0)$, podemos utilizar o método de eliminação:

[40]:
```
K_lin[0, 0] = 1.0
K_lin[0, 1] = 0.0
K_lin[1, 0] = 0.0
f_lin[0]    = 0.0
```

Calculando Vetor de Deslocamentos u

[41]:
```
u_lin = K_lin.inv() * f_lin
display(u_lin)
```

$$\begin{bmatrix} 0 \\ 0.000105263157894737 \end{bmatrix}$$

3.1.6 Aproximação Quadrática (5 elementos)

Fazendo uma aproximação quadrática, temos que:

$$u = u_1\phi_1 + u_2\phi_2 + u_3\phi_3$$

Sendo as funções bases de Lagrange para este caso:

$$\phi_1 = 1 - \frac{3x}{h} + \frac{2x^2}{h^2}$$

$$\phi_2 = \frac{4x}{h} - \frac{4x^2}{h^2}$$

$$\phi_3 = -\frac{x}{h} + \frac{2x^2}{h^2}$$

Calculando Matriz de Rigidez para cada elemento

$$K^i = EA \begin{bmatrix} \int_0^h \phi_1'\phi_1' dx & \int_0^h \phi_1'\phi_2' dx & \int_0^h \phi_1'\phi_3' dx \\ \int_0^h \phi_2'\phi_1' dx & \int_0^h \phi_2'\phi_2' dx & \int_0^h \phi_2'\phi_3' dx \\ \int_0^h \phi_2'\phi_1' dx & \int_0^h \phi_3'\phi_2' dx & \int_0^h \phi_3'\phi_3' dx \end{bmatrix}$$

[42]:
```
E, A, q, x, h = sp.symbols('E A q x h')

phi_1 = 1 - 3*x/h + 2*(x**2)/(h**2)
```

```python
phi_2 = 4*x/h - 4*(x**2)/(h**2)
phi_3 = -x/h + 2 * (x**2)/(h**2)

integral_1 = sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_1, x), (x, 0, h))
integral_2 = sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_2, x), (x, 0, h))
integral_3 = sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_3, x), (x, 0, h))

integral_4 = sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_1, x), (x, 0, h))
integral_5 = sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_2, x), (x, 0, h))
integral_6 = sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_3, x), (x, 0, h))

integral_7 = sp.integrate(sp.diff(phi_3, x) * sp.diff(phi_1, x), (x, 0, h))
integral_8 = sp.integrate(sp.diff(phi_3, x) * sp.diff(phi_2, x), (x, 0, h))
integral_9 = sp.integrate(sp.diff(phi_3, x) * sp.diff(phi_3, x), (x, 0, h))

K_ = E * A * sp.Matrix([
    [integral_1, integral_2, integral_3],
    [integral_4, integral_5, integral_6],
    [integral_7, integral_8, integral_9]
])
display(E*A/h, K_.subs({E: 1, A: 1, h: 1}))
```

$$\frac{AE}{h}$$

$$\begin{bmatrix} \frac{7}{3} & -\frac{8}{3} & \frac{1}{3} \\ -\frac{8}{3} & \frac{16}{3} & -\frac{8}{3} \\ \frac{1}{3} & -\frac{8}{3} & \frac{7}{3} \end{bmatrix}$$

Calculando Matriz de Rigidez Global

Para aproximações quadráticas e considerando que serão utilizados 5 elementos, temos que a matriz de rigidez global K será dada por:

$$K = \begin{bmatrix} K_{11}^1 & K_{12}^1 & K_{13}^1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ K_{21}^1 & K_{22}^1 & K_{23}^1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ K_{31}^1 & K_{32}^1 & K_{33}^1 + K_{11}^2 & K_{12}^2 & K_{13}^2 & \cdots & 0 & 0 & 0 \\ 0 & 0 & K_{21}^2 & K_{22}^2 & K_{23}^2 & \cdots & 0 & 0 & 0 \\ 0 & 0 & K_{31}^2 & K_{32}^2 & K_{33}^2 + K_{11}^3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & K_{33}^4 + K_{11}^5 & K_{12}^5 & K_{13}^5 \\ 0 & 0 & 0 & 0 & 0 & \cdots & K_{21}^5 & K_{22}^5 & K_{23}^5 \\ 0 & 0 & 0 & 0 & 0 & \cdots & K_{31}^5 & K_{32}^5 & K_{33}^5 \end{bmatrix}$$

[43]:
```python
n_elements = 5

def round_expr(expr, num_digits):
    ## funcao para truncar valores
```

```
    return expr.xreplace({n : round(n, num_digits) for n in expr.atoms(sp.
↪Number)})

K_G = sp.zeros(n_elements*2 + 1)

for i in range(n_elements):
    first = i * 2
    last  = i * 2 + 3
    K_G[first:last, first:last] += K_

display(E*A, round_expr(K_G.subs({h: L/5, A: 1, E: 1}), 3))
```

AE

$$\begin{bmatrix} 1.108 & -1.267 & 0.158 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.267 & 2.533 & -1.267 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.158 & -1.267 & 2.217 & -1.267 & 0.158 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.267 & 2.533 & -1.267 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.158 & -1.267 & 2.217 & -1.267 & 0.158 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.267 & 2.533 & -1.267 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.158 & -1.267 & 2.217 & -1.267 & 0.158 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1.267 & 2.533 & -1.267 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.158 & -1.267 & 2.217 & -1.267 & 0.158 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.267 & 2.533 & -1.267 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.158 & -1.267 & 1.108 \end{bmatrix}$$

Calculando Vetor Fonte para cada elemento

Temos que:

$$f^i = \begin{bmatrix} \int_0^h \phi_1 q(x)dx \\ \int_0^h \phi_2 q(x)dx \\ \int_0^h \phi_3 q(x)dx \end{bmatrix}$$

[44]:
```
f_ = sp.symbols('f')

f_ = sp.Matrix([[sp.integrate(phi_1 * q, (x, 0, h))], [sp.integrate(phi_2 * q,␣
↪(x, 0, h))], [sp.integrate(phi_3 * q, (x, 0, h))]])

display(f_)
```

$$\begin{bmatrix} \frac{hq}{6} \\ \frac{2hq}{3} \\ \frac{hq}{6} \end{bmatrix}$$

[45]:
```
F = sp.zeros(n_elements * 2 + 1, 1)

for i in range(n_elements):
```

```
    first = i * 2
    last  = i * 2 + 3
    F[first:last, 0] += f_

display(F)
```

$$\left[\begin{matrix}\frac{hq}{6}\\\frac{2hq}{3}\\\frac{hq}{3}\\\frac{2hq}{3}\\\frac{hq}{3}\\\frac{2hq}{3}\\\frac{hq}{3}\\\frac{2hq}{3}\\\frac{hq}{3}\\\frac{2hq}{3}\\\frac{hq}{6}\end{matrix}\right]$$

Aplicando Condições de Contorno

[46]:
```
for i in range(n_elements * 2 + 1):
    K_G[0, i] = 0.0
    K_G[i, 0] = 0.0
K_G[0, 0] = 1.0
F[0]      = 0.0
```

Substituindo os valores

[47]:
```
K_qua = K_G.subs({
    A: 0.1 * 0.5 * T,
    E: (10**8) * T,
    h: L/n_elements
})

f_qua = F.subs({
    h: L/n_elements,
    q: 10 * T
})
```

Calculando Matriz de Deslocamentos

[48]:
```
u_qua = K_qua.inv() * f_qua
display(u_qua)
```

$$\begin{bmatrix} 0 \\ 1.99999999999998 \cdot 10^{-5} \\ 3.7894736842105 \cdot 10^{-5} \\ 5.36842105263153 \cdot 10^{-5} \\ 6.7368421052631 \cdot 10^{-5} \\ 7.89473684210519 \cdot 10^{-5} \\ 8.84210526315781 \cdot 10^{-5} \\ 9.57894736842097 \cdot 10^{-5} \\ 0.000101052631578946 \\ 0.000104210526315789 \\ 0.000105263157894736 \end{bmatrix}$$

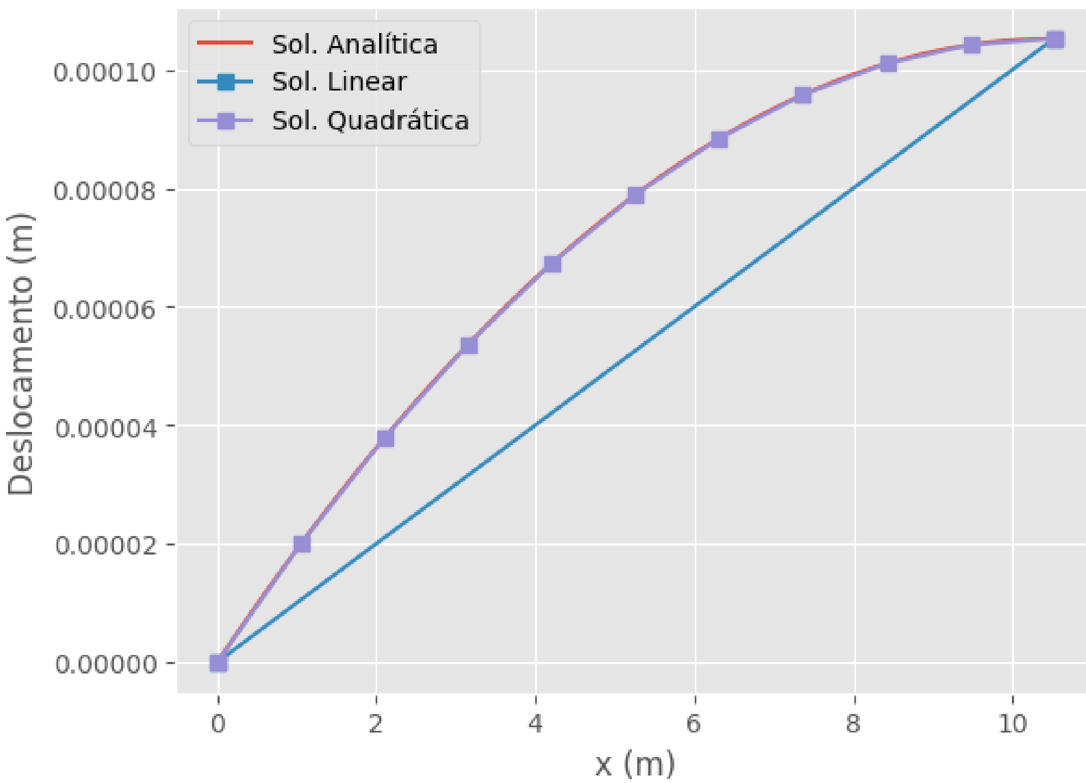
[ ]:
```
from matplotlib import pyplot as plt
plt.style.use('ggplot')
plt.figure(figsize=(10,2))
plt.plot([i*L/2 for i in range(3)],
         [0 for i in range(3)],
         '-o',
         color='black'
)
for i in range(2):
    plt.annotate(f'n{i+1}', (i*L/2,0.01))
    plt.annotate(f'e{i+1}', (i*L/2 + L/4,0.005))
plt.annotate('n3', (L, 0.01))
plt.show()
```

3.1.7 Resultados

[49]:
```
import numpy as np
from matplotlib import pyplot as plt
plt.style.use('ggplot')
```

[50]:
```
# funcoes

def sol_analitica(x, E, A, q, c1, c2):
    return -(1/(E*A)) * ((q * x**2/2) + c1 * x + c2)

L = 10 * T
E = (10 ** 8) * T
A = 0.1 * 0.5 * T
q = 10 * T

c1 = -1 * q * L
c2 = 0

# dominio
x_analitica = np.linspace(0, L)
```

```
y_analitica = sol_analitica(x_analitica, E, A, q, c1, c2)


# resultados
plt.plot(x_analitica, y_analitica, label='Sol. Analítica')
plt.plot([i*L for i in range(2)], u_lin, '-s', label='Sol. Linear')
plt.plot([i*L/10 for i in range(11)], u_qua, '-s', label='Sol. Quadrática')

plt.xlabel('x (m)')
plt.ylabel('Deslocamento (m)')
plt.legend()
plt.show()
```



4 Exercício 4

Dada uma barra com uma carga P aplicada no meio do seu vão, conforme indicado na Figura, pede-se, utilizando elementos finitos lineares, calcular os deslocamentos da mesma adotando-se uma divisão de 2, 4 e 6 elementos lineares. Comparar os campos de deslocamentos obtidos com os deslocamentos exatos para o problema em questão. Considere:

- $EA = 10^5 \cdot (N + 1)$ (Newton)
- $P = (N + 1) \cdot 10^3$ (Newton)

- $l = 2 + \frac{N+1}{20}$ (metro)

Resolução

4.0.1 Substituindo N

Substituindo $N = 0$, temos:

- $EA = 10^5$ (Newton)
- $P = 10^3$ (Newton)
- $l = \frac{41}{20}$ (metro)

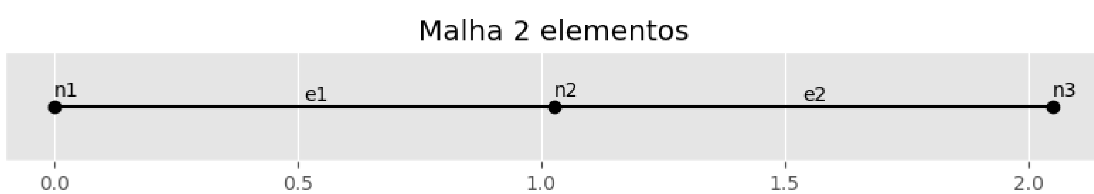
4.0.2 Equação do problema

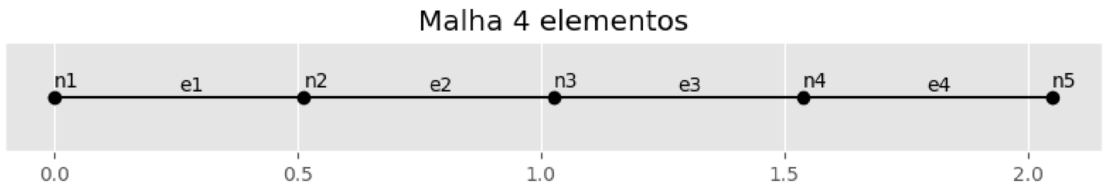
$$-EA\frac{\partial^2 u(x)}{\partial x^2} = P\delta\left(x - \frac{L}{2}\right), 0 \leq x \leq L$$

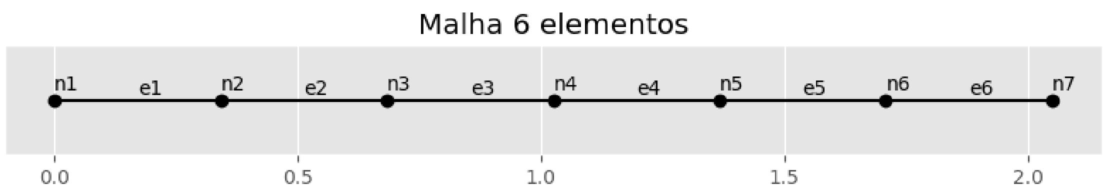
onde δ é a função delta de Dirac.

4.0.3 Malhas

[2]:
```
from matplotlib import pyplot as plt
plt.style.use('ggplot')
L=41/20
for n in [2, 4, 6]:
    plt.figure(figsize=(10,1))
    plt.plot([i*L/n for i in range(n+1)],
             [0 for i in range(n+1)],
             '-o',
             color='black'
    )
    for i in range(n):
        plt.annotate(f'n{i+1}', (i*L/n,0.01))
        plt.annotate(f'e{i+1}', (i*L/n + (L/n)/2,0.005))
    plt.annotate(f'n{n+1}', (L, 0.01))
    plt.gca().get_yaxis().set_visible(False)
    plt.title(f'Malha {n} elementos')
    plt.show()
```







4.0.4 Condições de Contorno

Da figura, temos que $u(0) = 0$ (condição de Dirichlet).

4.0.5 Aproximação Linear

Neste problema utilizaremos somente a aproximação linear, na forma:

$$u = u_1\phi_1 + u_2\phi_2$$

4.1 Aproximação com 2 Elementos

4.1.1 Definindo variáveis e calculando h

[46]:
```
import sympy as sp

EA, P, x, h, K, L = sp.symbols('EA P x h K L')

n_elements = 2
h = L/n_elements
```

4.1.2 Matriz de Rigidez por elemento

Inicialmente, calculamos a matriz de rigidez conforme calculado no exercício anterior:

[47]:
```
phi_1 = 1 - x/h
phi_2 = x/h

K = EA * sp.Matrix([
```

```
    [sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_1, x), (x, 0, h)), sp.
 ↪integrate(sp.diff(phi_1, x) * sp.diff(phi_2, x), (x, 0, h))],
    [sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_1, x), (x, 0, h)), sp.
 ↪integrate(sp.diff(phi_2, x) * sp.diff(phi_2, x), (x, 0, h))]
])
display(K)
```

$$\begin{bmatrix} \frac{2EA}{L} & -\frac{2EA}{L} \\ -\frac{2EA}{L} & \frac{2EA}{L} \end{bmatrix}$$

4.1.3 Matriz de Rigidez Global

Diferentemente da aproximação linear abordada anteriormente, neste caso como temos mais do que apenas 1 elemento, será necessário calcular a Matriz de Rigidez Global conforme:

$$K = \begin{bmatrix} k_{11}^1 & k_{12}^1 & 0 & 0 & \cdots & 0 & 0 \\ k_{21}^1 & k_{22}^1 + k_{11}^2 & k_{12}^2 & 0 & \cdots & 0 & 0 \\ 0 & k_{21}^2 & k_{12}^2 + k_{11}^3 & k_{12}^3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & k_{21}^e & k_{22}^e \end{bmatrix}$$

onde e é o número de elementos.

[48]:
```
K_G = sp.zeros(n_elements + 1)

for i in range(n_elements):
    K_G[i:i+2, i:i+2] += K

display(K_G)
```

$$\begin{bmatrix} \frac{2EA}{L} & -\frac{2EA}{L} & 0 \\ -\frac{2EA}{L} & \frac{4EA}{L} & -\frac{2EA}{L} \\ 0 & -\frac{2EA}{L} & \frac{2EA}{L} \end{bmatrix}$$

4.1.4 Vetor Fonte Global

[49]:
```
def dirac(d):
    if d == L/2:
        return 1
    return 0

F = sp.zeros(n_elements + 1, 1)

for i in range(n_elements):
    F[i, 0] = P * dirac(i*h)

display(F)
```

$$\begin{bmatrix} 0 \\ P \\ 0 \end{bmatrix}$$

4.1.5 Aplicando Condições de Contorno

[50]:
```
for i in range(n_elements + 1):
    K_G[0, i] = 0.0
    K_G[i, 0] = 0.0
K_G[0, 0] = 1.0
F[0]      = 0.0
display(K_G)
display(F)
```

$$\begin{bmatrix} 1.0 & 0 & 0 \\ 0 & \frac{4EA}{L} & -\frac{2EA}{L} \\ 0 & -\frac{2EA}{L} & \frac{2EA}{L} \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ P \\ 0 \end{bmatrix}$$

4.1.6 Substituindo valores

[51]:
```
K2 = K_G.subs({
    EA: 10**5,
    L: 41/20
})
display(K2)
```

$$\begin{bmatrix} 1.0 & 0 & 0 \\ 0 & 195121.951219512 & -97560.9756097561 \\ 0 & -97560.9756097561 & 97560.9756097561 \end{bmatrix}$$

[52]:
```
F2 = F.subs({
    P: 10**3
})
display(F2)
```

$$\begin{bmatrix} 0 \\ 1000 \\ 0 \end{bmatrix}$$

4.1.7 Calculando u

[53]:
```
u2 = K2.inv() * F2
display(u2)
```

$$\begin{bmatrix} 0 \\ 0.01025 \\ 0.01025 \end{bmatrix}$$

4.2 Aproximação com 4 elementos

Neste caso iremos repetir os mesmos passos porém utilizando 4 elementos:

4.2.1 Calculando h

[54]:
```
EA, P, x, h, K, L = sp.symbols('EA P x h K L')

n_elements = 4
h = L/n_elements
```

4.2.2 Matriz de Rigidez por elemento

Inicialmente, calculamos a matriz de rigidez conforme calculado no exercício anterior:

[55]:
```
phi_1 = 1 - x/h
phi_2 = x/h

K = EA * sp.Matrix([
    [sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_1, x), (x, 0, h)), sp.
 ↪integrate(sp.diff(phi_1, x) * sp.diff(phi_2, x), (x, 0, h))],
    [sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_1, x), (x, 0, h)), sp.
 ↪integrate(sp.diff(phi_2, x) * sp.diff(phi_2, x), (x, 0, h))]
])
display(K)
```

$$\begin{bmatrix} \frac{4EA}{L} & -\frac{4EA}{L} \\ -\frac{4EA}{L} & \frac{4EA}{L} \end{bmatrix}$$

4.2.3 Matriz de Rigidez Global

Diferentemente da aproximação linear abordada anteriormente, neste caso como temos mais do que apenas 1 elemento, será necessário calcular a Matriz de Rigidez Global conforme:

$$K = \begin{bmatrix} k_{11}^1 & k_{12}^1 & 0 & 0 & \cdots & 0 & 0 \\ k_{21}^1 & k_{22}^1 + k_{11}^2 & k_{12}^2 & 0 & \cdots & 0 & 0 \\ 0 & k_{21}^2 & k_{12}^2 + k_{11}^3 & k_{12}^3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & k_{21}^e & k_{22}^e \end{bmatrix}$$

onde e é o número de elementos.

[56]:
```
K_G = sp.zeros(n_elements + 1)

for i in range(n_elements):
```

```
    K_G[i:i+2, i:i+2] += K

display(K_G)
```

$$\begin{bmatrix} \frac{4EA}{L} & -\frac{4EA}{L} & 0 & 0 & 0 \\ -\frac{4EA}{L} & \frac{8EA}{L} & -\frac{4EA}{L} & 0 & 0 \\ 0 & -\frac{4EA}{L} & \frac{8EA}{L} & -\frac{4EA}{L} & 0 \\ 0 & 0 & -\frac{4EA}{L} & \frac{8EA}{L} & -\frac{4EA}{L} \\ 0 & 0 & 0 & -\frac{4EA}{L} & \frac{4EA}{L} \end{bmatrix}$$

4.2.4 Vetor Fonte Global

[57]:
```
def dirac(d):
    if d == L/2:
        return 1
    return 0

F = sp.zeros(n_elements + 1, 1)

for i in range(n_elements):
    F[i, 0] = P * dirac(i*h)

display(F)
```

$$\begin{bmatrix} 0 \\ 0 \\ P \\ 0 \\ 0 \end{bmatrix}$$

4.2.5 Aplicando Condições de Contorno

[58]:
```
for i in range(n_elements + 1):
    K_G[0, i] = 0.0
    K_G[i, 0] = 0.0
K_G[0, 0] = 1.0
F[0]      = 0.0
display(K_G)
display(F)
```

$$\begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 \\ 0 & \frac{8EA}{L} & -\frac{4EA}{L} & 0 & 0 \\ 0 & -\frac{4EA}{L} & \frac{8EA}{L} & -\frac{4EA}{L} & 0 \\ 0 & 0 & -\frac{4EA}{L} & \frac{8EA}{L} & -\frac{4EA}{L} \\ 0 & 0 & 0 & -\frac{4EA}{L} & \frac{4EA}{L} \end{bmatrix}$$

$$\begin{bmatrix}0\\0\\P\\0\\0\end{bmatrix}$$

4.2.6 Substituindo valores

[59]:
```
K4 = K_G.subs({
    EA: 10**5,
    L: 41/20
})
display(K4)
```

$$\begin{bmatrix}1.0 & 0 & 0 & 0 & 0\\0 & 390243.902439024 & -195121.951219512 & 0 & 0\\0 & -195121.951219512 & 390243.902439024 & -195121.951219512 & 0\\0 & 0 & -195121.951219512 & 390243.902439024 & -195121.951219512\\0 & 0 & 0 & -195121.951219512 & 195121.951219512\end{bmatrix}$$

[60]:
```
F4 = F.subs({
    P: 10**3
})
display(F4)
```

$$\begin{bmatrix}0\\0\\1000\\0\\0\end{bmatrix}$$

4.2.7 Calculando u

[61]:
```
u4 = K4.inv() * F4
display(u4)
```

$$\begin{bmatrix}0\\0.005125\\0.01025\\0.01025\\0.01025\end{bmatrix}$$

4.3 Aproximação com 6 elementos

4.3.1 Calculando h

[62]:
```
EA, P, x, h, K, L = sp.symbols('EA P x h K L')

n_elements = 6
h = L/n_elements
```

4.3.2 Matriz de Rigidez por elemento

Inicialmente, calculamos a matriz de rigidez conforme calculado no exercício anterior:

[63]:
```
phi_1 = 1 - x/h
phi_2 = x/h

K = EA * sp.Matrix([
    [sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_1, x), (x, 0, h)), sp.
 ↪integrate(sp.diff(phi_1, x) * sp.diff(phi_2, x), (x, 0, h))],
    [sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_1, x), (x, 0, h)), sp.
 ↪integrate(sp.diff(phi_2, x) * sp.diff(phi_2, x), (x, 0, h))]
])
display(K)
```

$$\begin{bmatrix} \frac{6EA}{L} & -\frac{6EA}{L} \\ -\frac{6EA}{L} & \frac{6EA}{L} \end{bmatrix}$$

4.3.3 Matriz de Rigidez Global

Diferentemente da aproximação linear abordada anteriormente, neste caso como temos mais do que apenas 1 elemento, será necessário calcular a Matriz de Rigidez Global conforme:

$$K = \begin{bmatrix} k_{11}^1 & k_{12}^1 & 0 & 0 & \cdots & 0 & 0 \\ k_{21}^1 & k_{22}^1 + k_{11}^2 & k_{12}^2 & 0 & \cdots & 0 & 0 \\ 0 & k_{21}^2 & k_{12}^2 + k_{11}^3 & k_{12}^3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & k_{21}^e & k_{22}^e \end{bmatrix}$$

onde e é o número de elementos.

[64]:
```
K_G = sp.zeros(n_elements + 1)

for i in range(n_elements):
    K_G[i:i+2, i:i+2] += K

display(K_G)
```

$$\begin{bmatrix} \frac{6EA}{L} & -\frac{6EA}{L} & 0 & 0 & 0 & 0 & 0 \\ -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} & 0 & 0 & 0 & 0 \\ 0 & -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} & 0 & 0 & 0 \\ 0 & 0 & -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} & 0 & 0 \\ 0 & 0 & 0 & -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} & 0 \\ 0 & 0 & 0 & 0 & -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} \\ 0 & 0 & 0 & 0 & 0 & -\frac{6EA}{L} & \frac{6EA}{L} \end{bmatrix}$$

4.3.4 Vetor Fonte Global

[65]:
```
def dirac(d):
    if d == L/2:
        return 1
    return 0

F = sp.zeros(n_elements + 1, 1)

for i in range(n_elements):
    F[i, 0] = P * dirac(i*h)

display(F)
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ P \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

4.3.5 Aplicando Condições de Contorno

[66]:
```
for i in range(n_elements + 1):
    K_G[0, i] = 0.0
    K_G[i, 0] = 0.0
K_G[0, 0] = 1.0
F[0]      = 0.0
display(K_G)
display(F)
```

$$\begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EA}{L} & -\frac{6EA}{L} & 0 & 0 & 0 & 0 \\ 0 & -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} & 0 & 0 & 0 \\ 0 & 0 & -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} & 0 & 0 \\ 0 & 0 & 0 & -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} & 0 \\ 0 & 0 & 0 & 0 & -\frac{6EA}{L} & \frac{12EA}{L} & -\frac{6EA}{L} \\ 0 & 0 & 0 & 0 & 0 & -\frac{6EA}{L} & \frac{6EA}{L} \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ P \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

4.3.6 Substituindo valores

[67]:
```python
K6 = K_G.subs({
    EA: 10**5,
    L: 41/20
})
display(K6)
```

$$\begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 585365.853658537 & -292682.926829268 & 0 & 0 & 0 \\ 0 & -292682.926829268 & 585365.853658537 & -292682.926829268 & 0 & 0 \\ 0 & 0 & -292682.926829268 & 585365.853658537 & -292682.926829268 & 0 \\ 0 & 0 & 0 & -292682.926829268 & 585365.853658537 & -292682.926829268 \\ 0 & 0 & 0 & 0 & -292682.926829268 & 585365.853658537 \\ 0 & 0 & 0 & 0 & 0 & -292682.926829268 \end{bmatrix}$$

[68]:
```python
F6 = F.subs({
    P: 10**3
})
display(F6)
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1000 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

4.3.7 Calculando u

[69]:
```python
u6 = K6.inv() * F6
display(u6)
```

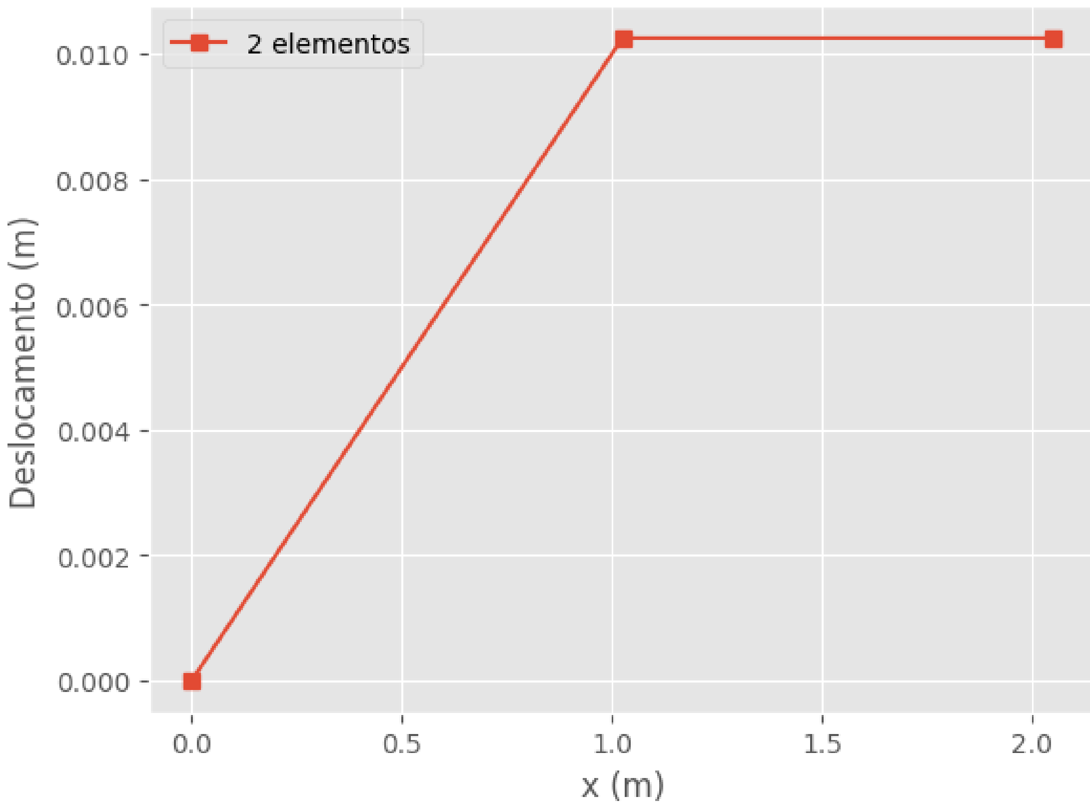
$$\begin{bmatrix} 0 \\ 0.00341666666666667 \\ 0.00683333333333333 \\ 0.01025 \\ 0.01025 \\ 0.01025 \\ 0.01025 \end{bmatrix}$$

4.4 Resultados

[74]:
```python
from matplotlib import pyplot as plt
plt.style.use('ggplot')
L = 41/20
```

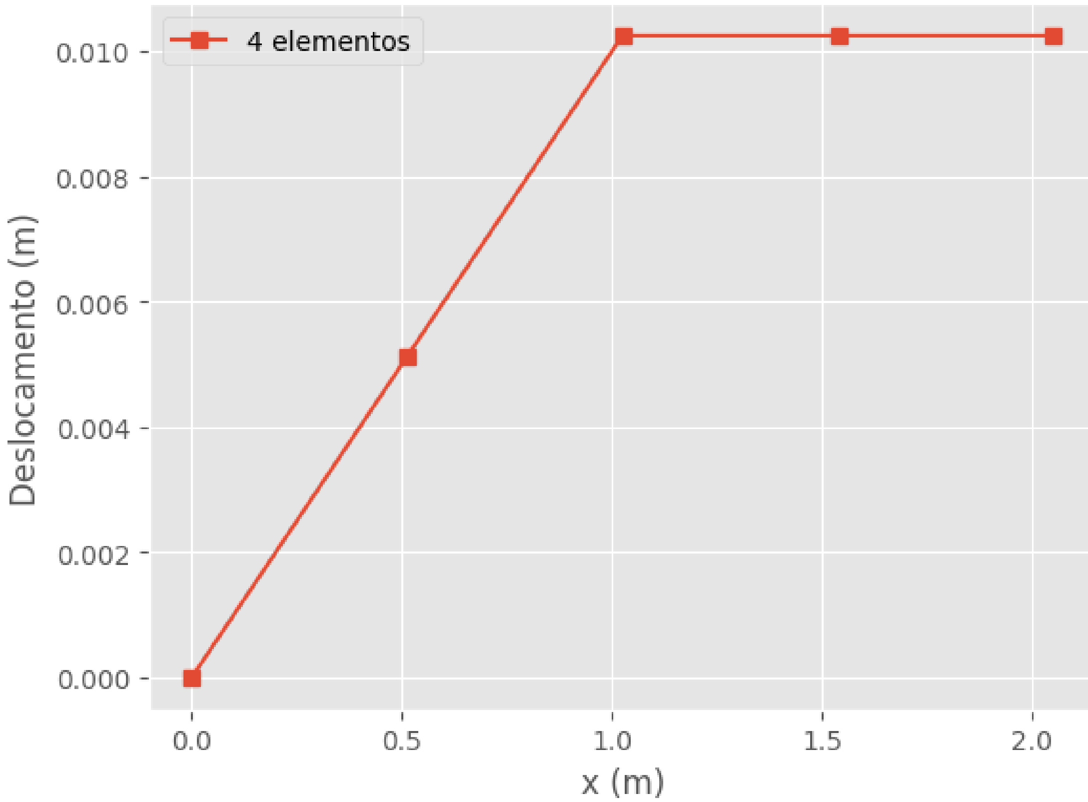
4.4.1 2 elementos

[77]:
```
plt.plot([i*L/2 for i in range(3)], u2, '-s', label='2 elementos')

plt.xlabel('x (m)')
plt.ylabel('Deslocamento (m)')
plt.legend()
plt.show()
```



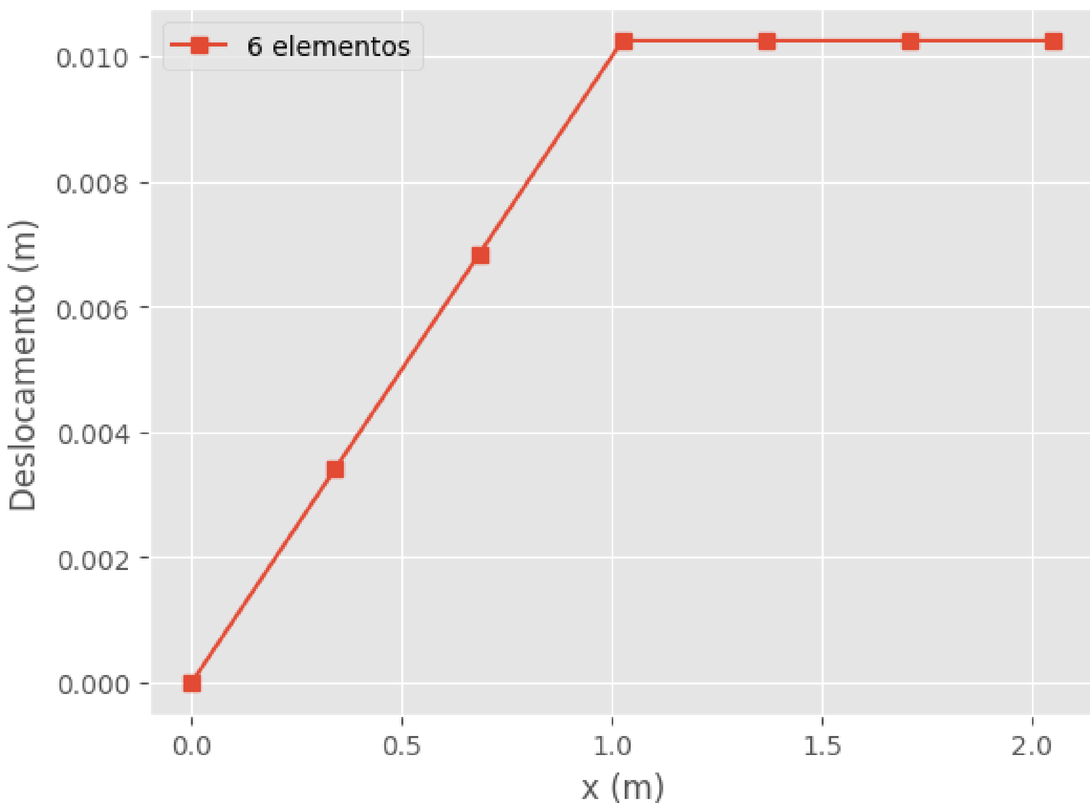
4.4.2 4 elementos

[76]:
```
plt.plot([i*L/4 for i in range(5)], u4, '-s', label='4 elementos')

plt.xlabel('x (m)')
plt.ylabel('Deslocamento (m)')
plt.legend()
plt.show()
```



4.4.3 6 elementos

[75]:
```
plt.plot([i*L/6 for i in range(7)], u6, '-s', label='6 elementos')

plt.xlabel('x (m)')
plt.ylabel('Deslocamento (m)')
plt.legend()
plt.show()
```



5 Exercício 5

Considerando a barra do exercício anterior com a carga P aplicada na sua extremidade livre, com EA igual a deste exercício, utilizando-se uma divisão com 7 elementos finitos com aproximações lineares, cada um com um comprimento diferente denominados por h_1 a h_7, sendo $h_1 = \frac{l}{10}$ e $h_i = h_1 + \frac{i*l}{70}$ para $i = 2$ a 7 pede-se montar a matriz de rigidez global para a numeração conforme indicado abaixo. Cada aluno deve adotar uma numeração dos nós dependente do valor de N, associado conforme a tabela fornecida em função do seu número de ordem na lista de chamada. Determine também a matriz de rigidez para uma numeração sequencial dos nós da malha.

1 - Apresente as matrizes de cada elemento para o valor de N associado ao seu caso.

2 - Apresente a matriz de conectividade adotada para a numeração escolhida

3 - Apresente as matrizes globais resultante da composição das matrizes individuais

4 - Apresente a solução obtida e compare graficamente com a solução exata do problema

5.1 Resolução

5.1.1 Equação do Problema

Temos que o problema é descrito pela equação:

$$EA\frac{d^2u(x)}{dx^2} = P\delta(x-a), 0 \leq x \leq L$$

onde $a = L$.

5.1.2 Numeração dos nós

A numeração para $N = 0$ será:

$$\begin{bmatrix} 1 & 3 & 5 & 7 & 2 & 6 & 4 \end{bmatrix}$$

5.1.3 Cáculo dos comprimentos dos elementos

Consideraremos o parâmetro $l = 20$ para o cálculo do tamanho dos elementos:

[2]:
```
import sympy as sp

h1, h2, h3, h4, h5, h6, h7 = sp.symbols('h_1 h_2 h_3 h_4 h_5 h_6 h_7')

l = 20

hi = [h1, h2, h3, h4, h5, h6, h7]
hl = [l/10]
display(sp.Eq(hi[0], hl[0]))

for i in range(1,7):
    hl.append(hl[0] + (i*l)/70)
    display(sp.Eq(hi[i], hl[i]))
```

$h_1 = 2.0$

$h_2 = 2.28571428571429$

$h_3 = 2.57142857142857$

$h_4 = 2.85714285714286$

$h_5 = 3.14285714285714$

$h_6 = 3.42857142857143$

$h_7 = 3.71428571428571$

5.1.4 1- Matrizes de cada elemento

[4]:
```
EA, x, h, k1, k2, k3, k4, k5, k6, k7 = sp.symbols('EA x h k_1 k_2 k_3 k_4 k_5␣
↪k_6 k_7')

phi_1 = 1 - x/h
phi_2 = x/h
```

```
k = EA * sp.Matrix([
    [
        sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_1, x), (x, 0, h)),
        sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_2, x), (x, 0, h))
    ],
    [
        sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_1, x), (x, 0, h)),
        sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_2, x), (x, 0, h))
    ]
])
display(EA/h, k.subs({EA: 1, h: 1}))
```

$$\frac{EA}{h}$$

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

k_1

[5]:
```
k1 = k.subs({h: hl[0]})
display(k1)
```

$$\begin{bmatrix} 0.5EA & -0.5EA \\ -0.5EA & 0.5EA \end{bmatrix}$$

k_2

[6]:
```
k2 = k.subs({h: hl[1]})
display(k2)
```

$$\begin{bmatrix} 0.4375EA & -0.4375EA \\ -0.4375EA & 0.4375EA \end{bmatrix}$$

k_3

[7]:
```
k3 = k.subs({h: hl[2]})
display(k3)
```

$$\begin{bmatrix} 0.388888888888889EA & -0.388888888888889EA \\ -0.388888888888889EA & 0.388888888888889EA \end{bmatrix}$$

k_4

[8]:
```
k4 = k.subs({h: hl[3]})
display(k4)
```

$$\begin{bmatrix} 0.35EA & -0.35EA \\ -0.35EA & 0.35EA \end{bmatrix}$$

k_5

[9]:
```python
k5 = k.subs({h: hl[4]})
display(k5)
```

$$\begin{bmatrix} 0.318181818181818EA & -0.318181818181818EA \\ -0.318181818181818EA & 0.318181818181818EA \end{bmatrix}$$

k_6

[10]:
```python
k6 = k.subs({h: hl[5]})
display(k6)
```

$$\begin{bmatrix} 0.291666666666667EA & -0.291666666666667EA \\ -0.291666666666667EA & 0.291666666666667EA \end{bmatrix}$$

k_7

[27]:
```python
k7 = EA * sp.Matrix([
    [sp.integrate(sp.diff(phi_1, x) * sp.diff(phi_1, x), (x, 0, h)), sp.
 ↪integrate(sp.diff(phi_1, x) * sp.diff(phi_2, x), (x, 0, h))],
    [sp.integrate(sp.diff(phi_2, x) * sp.diff(phi_1, x), (x, 0, h)), sp.
 ↪integrate(sp.diff(phi_2, x) * sp.diff(phi_2, x), (x, 0, h))]
]).subs({h: hl[6]})
display(k7)
```

$$\begin{bmatrix} 0.269230769230769EA & -0.269230769230769EA \\ -0.269230769230769EA & 0.269230769230769EA \end{bmatrix}$$

5.1.5 2- Matriz de conectividade

[22]:
```python
C = sp.Matrix([
[1, 3],
[3, 5],
[5, 7],
[7, 2],
[2, 6],
[6, 4],
[4, 8],
])
display(C)
```

$$\begin{bmatrix} 1 & 3 \\ 3 & 5 \\ 5 & 7 \\ 7 & 2 \\ 2 & 6 \\ 6 & 4 \\ 4 & 8 \end{bmatrix}$$

Malha com índices dos nós e elementos

[51]:
```
from matplotlib import pyplot as plt
import numpy as np

plt.style.use('ggplot')
plt.figure(figsize=(10,1))

n_elementos = len(C[:,0])
n_nos = n_elementos + 1

h_acumulado = np.concatenate(([0], np.cumsum(hl)))

plt.plot(h_acumulado, [0 for i in range(n_nos)], '-o', color='black')

for i in range(n_nos - 1):
    plt.annotate(f'Nó {C[i,0]}', (h_acumulado[i],0.01))
    plt.annotate(f'e{i+1}', ((h_acumulado[i] + (h_acumulado[i+1] -
 ↪h_acumulado[i])/2),-0.025))
plt.annotate(f'Nó 8', (h_acumulado[-1],0.01))

plt.gca().get_yaxis().set_visible(False)
plt.title(f'Malha {n_elementos} elementos')
plt.show()
```