Universidade Federal de Santa Catarina

Centro de Tecnologia

Programa de Pós-Graduação em Engenharia Mecânica

EMC 410096 - Método de Elementos Finitos A

Carlos Augusto Soares Ferreira

Trabalho Final

Florianópolis

2019

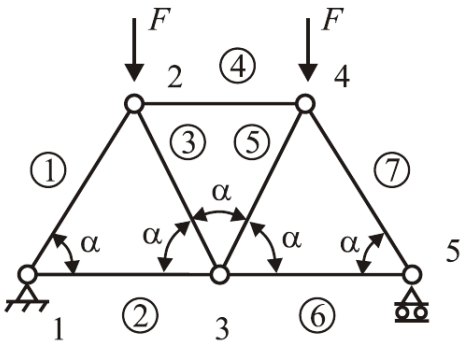
Sumário

1 Trabalho 1

1.1 Trabalho 1.1 - Treliças

1.1.1 Exercício 15.6

Dado o seguinte problema, determine:



A - Matriz de rigidez de cada elemento.

B - Sobreponha as matrizes elementares e obtenha a matriz de rigidez global.

C - Identifique e aplique no sistema as condições de contorno, de compatibilidade e as forças nodais aplicadas.

D - Obtenha os deslocamentos nodais.

E - Indique o processo de cálculo das reações e faça sua determinação.

F - Detalhe a determinação dos esforços e tensões em cada elemento.

G - Faça análise de falha por início de escoamento.

Considere as barras com seção retangular de 10×10 mm^2. Onde indicado, $L = 1$ m, $A = 100$ mm^2, $E = 200$ GPa, $F = 1$ kN, $\alpha = 60°$. Em cada elemento, enumere os nós em ordem crescente. $\sigma_E = 200$ MPa.

Etapa 1 - Dados de coordenadas nodais

As coordenadas dos nós são apresentadas na Tabela 1.

Tabela 1 – Coordenadas nodais.

Nó	**x**	**y**
1	0	0
2	1/2	$\sqrt{3}/2$
3	1	0
4	3/2	$\sqrt{3}/2$
5	2	0

Etapa 2 - Dados de conectividade dos elementos

A conectividade dos elementos é apresentada na Tabela 2.

Tabela 2 – Conectividade dos elementos.

Elemento	**Nó I**	**Nó J**
1	1	2
2	1	3
3	2	3
4	2	4
5	3	4
6	3	5
7	4	5

Etapa 3 - Propriedades geométricas e de material

Como assumimos barras homogêneas e de seção retangular constante, temos $E = 200 \cdot 10^6$ Pa e $A = 100 \cdot 10^{-6}$ m^2

Etapa 4 - Condições de contorno de deslocamento

Os valores prescritos de deslocamentos nodais são apresentados na Tabela 3.

Tabela 3 – Condições de contorno de deslocamento.

Nó	**Componente de deslocamento**	**Valor prescrito [m]**
1	u_1	0
1	v_1	0
5	v_5	0

Etapa 5 - Dados de carregamento

Os valores nodais de cargas externas são apresentados na Tabela 4.

Tabela 4 – Valores nodais de carregamentos.

Nó	**Componente de força**	**Valor aplicado [N]**
2	F^a_{2y}	-1000
4	F^a_{4y}	-1000

Etapa 6 - Cálculo das matrizes de rigidez elementares

A matriz de rigidez para um elemento finito de barra é dada por

$$\boldsymbol{K}^e = \frac{EA}{L_e}\begin{bmatrix} \cos^2\theta_e & \cos\theta_e\text{sen}\theta_e & -\cos^2\theta_e & -\cos\theta_e\text{sen}\theta_e \\ \cos\theta_e\text{sen}\theta_e & \text{sen}^2\theta_e & -\cos\theta_e\text{sen}\theta_e & -\text{sen}^2\theta_e \\ -\cos^2\theta_e & -\cos\theta_e\text{sen}\theta_e & \cos^2\theta_e & \cos\theta_e\text{sen}\theta_e \\ -\cos\theta_e\text{sen}\theta_e & -\text{sen}^2\theta_e & \cos\theta_e\text{sen}\theta_e & \text{sen}^2\theta_e \end{bmatrix} \tag{1.1}$$

onde

$$\theta_e = \arctan\left(\frac{y_J - y_I}{x_J - x_I}\right) \tag{1.2}$$

e

$$L_e = \sqrt{(x_I - x_J)^2 + (y_I - y_J)^2} \tag{1.3}$$

Elemento 1: Obtemos $L_1 = 1$ m e $\theta_1 = 60°$. Portanto, $\boldsymbol{K}_1$ é dado por

$$\boldsymbol{K}_1 = 5 \cdot 10^3 \begin{bmatrix} 1 & \sqrt{3} & -1 & -\sqrt{3} \\ \sqrt{3} & 3 & -\sqrt{3} & -3 \\ -1 & -\sqrt{3} & 1 & \sqrt{3} \\ -\sqrt{3} & -3 & \sqrt{3} & 3 \end{bmatrix}$$

Elemento 2: Obtemos $L_2 = 1$ m e $\theta_2 = 0°$. Portanto, $\boldsymbol{K}_2$ é dado por

$$\boldsymbol{K}_2 = 5 \cdot 10^3 \begin{bmatrix} 4 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 \\ -4 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Elemento 3: Obtemos $L_3 = 1$ m e $\theta_3 = 120°$. Portanto, $\boldsymbol{K}_3$ é dado por

$$\boldsymbol{K}_3 = 5 \cdot 10^3 \begin{bmatrix} 1 & -\sqrt{3} & -1 & \sqrt{3} \\ -\sqrt{3} & 3 & \sqrt{3} & -3 \\ -1 & \sqrt{3} & 1 & -\sqrt{3} \\ \sqrt{3} & -3 & -\sqrt{3} & 3 \end{bmatrix}$$

Elemento 4: Obtemos $L_4 = 1$ m e $\theta_4 = 0°$. Portanto, $\boldsymbol{K}_4$ é dado por

$$\boldsymbol{K}_4 = 5 \cdot 10^3 \begin{bmatrix} 4 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 \\ -4 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Elemento 5: Obtemos $L_5 = 1$ m e $\theta_5 = 60°$. Portanto, $\boldsymbol{K}_5$ é dado por

$$\boldsymbol{K}_5 = 5 \cdot 10^3 \begin{bmatrix} 1 & \sqrt{3} & -1 & -\sqrt{3} \\ \sqrt{3} & 3 & -\sqrt{3} & -3 \\ -1 & -\sqrt{3} & 1 & \sqrt{3} \\ -\sqrt{3} & -3 & \sqrt{3} & 3 \end{bmatrix}$$

Elemento 6: Obtemos $L_6 = 1$ m e $\theta_6 = 0°$. Portanto, $\boldsymbol{K}_6$ é dado por

$$\boldsymbol{K}_6 = 5 \cdot 10^3 \begin{bmatrix} 4 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 \\ -4 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Elemento 7: Obtemos $L_7 = 1$ m e $\theta_7 = 120°$. Portanto, $\boldsymbol{K}_7$ é dado por

$$\boldsymbol{K}_7 = 5 \cdot 10^3 \begin{bmatrix} 1 & -\sqrt{3} & -1 & \sqrt{3} \\ -\sqrt{3} & 3 & \sqrt{3} & -3 \\ -1 & \sqrt{3} & 1 & -\sqrt{3} \\ \sqrt{3} & -3 & -\sqrt{3} & 3 \end{bmatrix}$$

Etapa 7 - Cálculo da matriz de rigidez global

Fazendo a sobreposição das matrizes elementares, obtemos a matriz de rigidez global

$$\boldsymbol{K} = 5 \cdot 10^3 \begin{bmatrix} 5 & \sqrt{3} & -1 & -\sqrt{3} & -4 & 0 & 0 & 0 & 0 & 0 \\ & 3 & -\sqrt{3} & -3 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 6 & 0 & -1 & \sqrt{3} & -4 & 0 & 0 & 0 \\ & & & 6 & \sqrt{3} & -3 & 0 & 0 & 0 & 0 \\ & & & & 10 & 0 & -1 & -\sqrt{3} & -4 & 0 \\ & & & & & 6 & -\sqrt{3} & -3 & 0 & 0 \\ & & & & & & 6 & 0 & -1 & \sqrt{3} \\ & & & & & & & 6 & \sqrt{3} & -3 \\ & & & & & & & & 5 & -\sqrt{3} \\ \text{sim.} & & & & & & & & & 3 \end{bmatrix}$$

Etapa 8 - Definição do vetor de carregamento nodal

O vetor de forças aplicadas, conforme os dados da Tabela 4, é dado por

$$\boldsymbol{F}^a = \begin{Bmatrix} F^a_{1x} \\ F^a_{1y} \\ F^a_{2x} \\ F^a_{2y} \\ F^a_{3x} \\ F^a_{3y} \\ F^a_{4x} \\ F^a_{4y} \\ F^a_{5x} \\ F^a_{5y} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ -1000 \\ 0 \\ 0 \\ 0 \\ -1000 \\ 0 \\ 0 \end{Bmatrix}$$

O vetor de forças de reação é dado por

$$\boldsymbol{R} = \begin{Bmatrix} R_{1x} \\ R_{1y} \\ R_{2x} \\ R_{2y} \\ R_{3x} \\ R_{3y} \\ R_{4x} \\ R_{4y} \\ R_{5x} \\ R_{5y} \end{Bmatrix}$$

O sistema linear dado por $\boldsymbol{KU} = \boldsymbol{F}^a + \boldsymbol{R}$ tem a seguinte forma

$$5{\cdot}10^3 \begin{bmatrix} 5 & \sqrt{3} & -1 & -\sqrt{3} & -4 & 0 & 0 & 0 & 0 & 0 \\ & 3 & -\sqrt{3} & -3 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 6 & 0 & -1 & \sqrt{3} & -4 & 0 & 0 & 0 \\ & & & 6 & \sqrt{3} & -3 & 0 & 0 & 0 & 0 \\ & & & & 10 & 0 & -1 & -\sqrt{3} & -4 & 0 \\ & & & & & 6 & -\sqrt{3} & -3 & 0 & 0 \\ & & & & & & 6 & 0 & -1 & \sqrt{3} \\ & & & & & & & 6 & \sqrt{3} & -3 \\ & & & & & & & & 5 & -\sqrt{3} \\ \text{sim.} & & & & & & & & & 3 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \\ u_5 \\ v_5 \end{Bmatrix} = \begin{Bmatrix} R_{1x} \\ R_{1y} \\ R_{2x} \\ -1000 + R_{2y} \\ R_{3x} \\ R_{3y} \\ R_{4x} \\ -1000 + R_{4y} \\ R_{5x} \\ R_{5y} \end{Bmatrix}$$

Etapa 9 - Aplicação das condições de contorno

Alterando o sistema linear, considerando as condições de contorno apresentadas na Tabela 3, obtemos o sistema linear dado por $\overline{\boldsymbol{K}}\boldsymbol{U} = \boldsymbol{F}^a$ que tem a seguinte forma

$$5 \cdot 10^3 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 6 & 0 & -1 & \sqrt{3} & -4 & 0 & 0 & 0 \\ & & & 6 & \sqrt{3} & -3 & 0 & 0 & 0 & 0 \\ & & & & 10 & 0 & -1 & -\sqrt{3} & -4 & 0 \\ & & & & & 6 & -\sqrt{3} & -3 & 0 & 0 \\ & & & & & & 6 & 0 & -1 & 0 \\ & & & & & & & 6 & \sqrt{3} & 0 \\ & & & & & & & & 5 & 0 \\ \text{sim.} & & & & & & & & & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \\ u_5 \\ v_5 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ -1000 \\ 0 \\ 0 \\ 0 \\ -1000 \\ 0 \\ 0 \end{Bmatrix}$$

Etapa 10 - Determinação do campo de deslocamentos

O campo de deslocamentos, obtido pela solução do sistema linear $\boldsymbol{U} = \left(\overline{\boldsymbol{K}}\right)^{-1} \boldsymbol{F}^a$, é dado por

$$\boldsymbol{U} = \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \\ u_5 \\ v_5 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0,04330 \\ -0,09167 \\ 0,02887 \\ -0,1 \\ 0,01443 \\ -0,09167 \\ 0,05774 \\ 0 \end{Bmatrix}$$

Etapa 11 - Determinação das reações nos apoios

As reações nos apoios são dadas por $\boldsymbol{R} = \boldsymbol{K}\boldsymbol{U} - \boldsymbol{F}^a$. Assim, obtemos

$$\boldsymbol{R} = \begin{Bmatrix} R_{1x} \\ R_{1y} \\ R_{2x} \\ R_{2y} \\ R_{3x} \\ R_{3y} \\ R_{4x} \\ R_{4y} \\ R_{5x} \\ R_{5y} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 1000 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1000 \end{Bmatrix}$$

Etapa 12 - Determinação dos deslocamentos nos sistemas locais

Para determinação das deformações e tensões em cada elemento, primeiramente identifica-se no vetor global $\boldsymbol{U}$ os termos correspondentes a cada elemento, conforme Tabela 2. Para cada elemento, faz-se a rotação dos deslocamentos do sistema global para o sistema local do elemento, através da equação

$$\begin{Bmatrix} \overline{u}_I \\ \overline{v}_I \\ \overline{u}_J \\ \overline{v}_J \end{Bmatrix} = \begin{bmatrix} \cos\theta_e & \text{sen}\theta_e & 0 & 0 \\ -\text{sen}\theta_e & \cos\theta_e & 0 & 0 \\ 0 & 0 & \cos\theta_e & \text{sen}\theta_e \\ 0 & 0 & -\text{sen}\theta_e & \cos\theta_e \end{bmatrix} \begin{Bmatrix} u_I \\ v_I \\ u_J \\ v_J \end{Bmatrix} \tag{1.4}$$

Elemento 1: Temos $\theta_1 = 60°$, $I = 1$ e $J = 2$, portanto

$$\begin{Bmatrix} \overline{u}_1 \\ \overline{v}_1 \\ \overline{u}_2 \\ \overline{v}_2 \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & \sqrt{3} & 0 & 0 \\ -\sqrt{3} & 1 & 0 & 0 \\ 0 & 0 & 1 & \sqrt{3} \\ 0 & 0 & -\sqrt{3} & 1 \end{bmatrix} \begin{Bmatrix} 0 \\ 0 \\ 0,04330 \\ -0,09167 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0,0577 \\ -0,0833 \end{Bmatrix}$$

Elemento 2: Temos $\theta_1 = 0°$, $I = 1$ e $J = 3$, portanto

$$\begin{Bmatrix} \overline{u}_1 \\ \overline{v}_1 \\ \overline{u}_3 \\ \overline{v}_3 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} 0 \\ 0 \\ 0,02887 \\ -0,1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0,02887 \\ -0,1 \end{Bmatrix}$$

Elemento 3: Temos $\theta_1 = 120°$, $I = 2$ e $J = 3$, portanto

$$\begin{Bmatrix} \overline{u}_2 \\ \overline{v}_2 \\ \overline{u}_3 \\ \overline{v}_3 \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} -1 & \sqrt{3} & 0 & 0 \\ -\sqrt{3} & -1 & 0 & 0 \\ 0 & 0 & -1 & \sqrt{3} \\ 0 & 0 & -\sqrt{3} & -1 \end{bmatrix} \begin{Bmatrix} 0,04330 \\ -0,09167 \\ 0,02887 \\ -0,1 \end{Bmatrix} = \begin{Bmatrix} -0,10104 \\ 0,00833 \\ -0,10104 \\ 0,025 \end{Bmatrix}$$

Elemento 4: Temos $\theta_1 = 0°$, $I = 2$ e $J = 4$, portanto

$$\begin{Bmatrix} \overline{u}_2 \\ \overline{v}_2 \\ \overline{u}_4 \\ \overline{v}_4 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} 0,0433 \\ -0,09167 \\ 0,01443 \\ -0,09167 \end{Bmatrix} = \begin{Bmatrix} 0,0433 \\ -0,09167 \\ 0,01443 \\ -0,09167 \end{Bmatrix}$$

Elemento 5: Temos $\theta_1 = 60°$, $I = 3$ e $J = 4$, portanto

$$\begin{Bmatrix} \overline{u}_3 \\ \overline{v}_3 \\ \overline{u}_4 \\ \overline{v}_4 \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & \sqrt{3} & 0 & 0 \\ -\sqrt{3} & 1 & 0 & 0 \\ 0 & 0 & 1 & \sqrt{3} \\ 0 & 0 & -\sqrt{3} & 1 \end{bmatrix} \begin{Bmatrix} 0,02887 \\ -0,1 \\ 0,01443 \\ -0,09167 \end{Bmatrix} = \begin{Bmatrix} -0,07217 \\ -0,075 \\ -0,07217 \\ -0,05833 \end{Bmatrix}$$

Elemento 6: Temos $\theta_1 = 0°$, $I = 3$ e $J = 5$, portanto

$$\begin{Bmatrix} \overline{u}_3 \\ \overline{v}_3 \\ \overline{u}_5 \\ \overline{v}_5 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} 0,02887 \\ -0,1 \\ 0,05774 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0,02887 \\ -0,1 \\ 0,05774 \\ 0 \end{Bmatrix}$$

Elemento 7: Temos $\theta_1 = 120°$, $I = 4$ e $J = 5$, portanto

$$\begin{Bmatrix} \overline{u}_4 \\ \overline{v}_4 \\ \overline{u}_5 \\ \overline{v}_5 \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} -1 & \sqrt{3} & 0 & 0 \\ -\sqrt{3} & -1 & 0 & 0 \\ 0 & 0 & -1 & \sqrt{3} \\ 0 & 0 & -\sqrt{3} & -1 \end{bmatrix} \begin{Bmatrix} 0,01443 \\ -0,09167 \\ 0,05774 \\ 0 \end{Bmatrix} = \begin{Bmatrix} -0,0866 \\ 0,03333 \\ -0,02887 \\ -0,05 \end{Bmatrix}$$

12 - Determinação das deformações e tensões normais axiais

A deformação normal axial em qualquer ponto do elemento é dada por

$$\varepsilon_{xx}^{e} = \frac{\overline{u}_J - \overline{u}_I}{L_e} \tag{1.5}$$

e obtemos $\varepsilon^1_{xx} = 0,0433$, $\varepsilon^2_{xx} = 0,02887$, $\varepsilon^3_{xx} = -0,01443$, $\varepsilon^4_{xx} = -0,02887$, $\varepsilon^5_{xx} = -0,01443$, $\varepsilon^6_{xx} = 0,02887$ e $\varepsilon^7_{xx} = 0,0433$.

Com o auxílio da Lei de Hooke, podemos escrever

$$\sigma^e_{xx} = E\varepsilon^e_{xx} \tag{1.6}$$

e obtemos $\sigma^1_{xx} = 8,66$ MPa, $\sigma^2_{xx} = 5,77$ MPa, $\sigma^3_{xx} = -2,89$ MPa, $\sigma^4_{xx} = -5,77$ MPa, $\sigma^5_{xx} = -2,89$ MPa, $\sigma^6_{xx} = 5,77$ MPa e $\sigma^7_{xx} = 8,66$ MPa. Não temos, portanto, situação de falha por início de escoamento em nenhum dos elementos.

1.2 Trabalho 1.2 - Manual Fortran

1.2.1 Exercício 1

Definir uma FUNCTION para calcular $f(x) = x^2 + \sqrt{1 + 2x + 3x^2}$. Use a função para calcular:

$$a_1 = \frac{6,9 + y}{y^2 + \sqrt{1 + 2y + 3y^2}}$$

$$a_2 = \frac{21y + y^4}{y^2 + \sqrt{1 + 2y + 3y^2}}$$

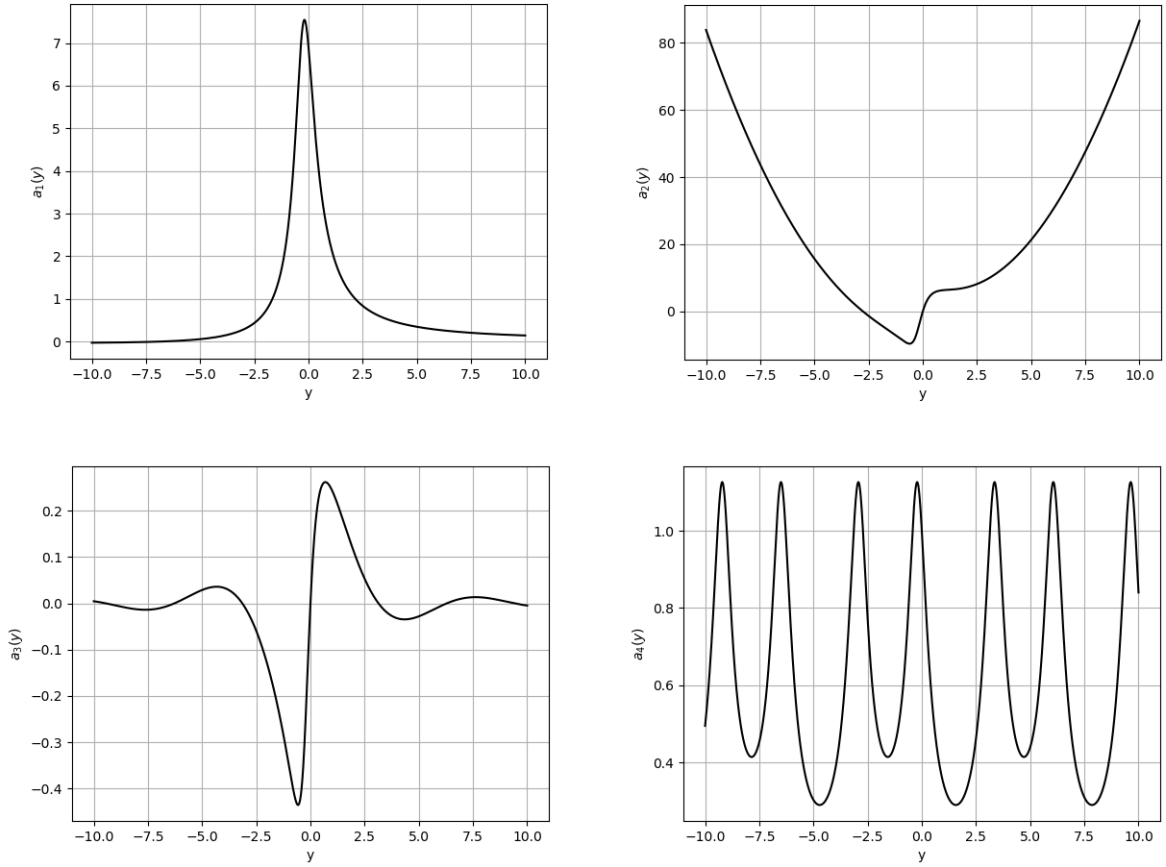
$$a_3 = \frac{\text{sen}y}{y^2 + \sqrt{1 + 2y + 3y^2}}$$

$$a_4 = \frac{1}{\text{sen}^2 y + \sqrt{1 + 2\text{sen}y + 3\text{sen}^2 y}}$$

Calcule os valores para um conjunto de coordenadas y num intervalo $[a, b]$ e armazene-os num vetor. Faça um relatório com estes valores num arquivo e no vídeo. Importe os valores como planilha e plote os gráficos.

O código **main.f** utilizado para o cálculo dos valores de a_1, a_2, a_3 e a_4 no intervalo $[-10, 10]$ é apresentado no apêndice A, seção A.1.1. Para a alocação dinâmica dos vetores que contém os valores calculados, foi elaborado o módulo **dynamicalArrays** cujo código é apresentado no início da seção A.1. A plotagem dos gráficos foi feita através de uma implementação em Python, cujo código **plot.py** também é apresentado na seção A.1.1. A Figura 1 apresenta os resultados obtidos.

Figura 1 – Valores de $a_1(y)$, $a_2(y)$, $a_3(y)$ e $a_4(y)$ no intervalo $[-10, 10]$.

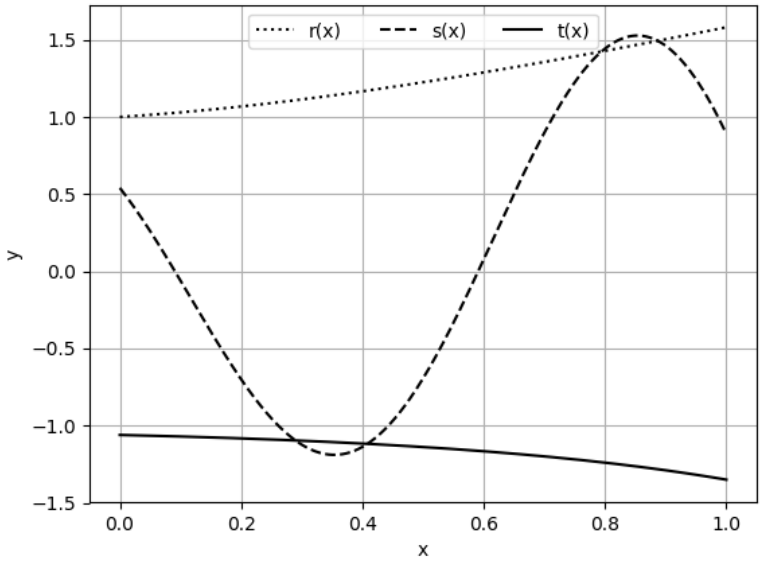


1.2.2 Exercício 2

Escrever uma subrotina para calcular $r = \sqrt{a + bx + x^2}$, $s = \cos(2\pi x + a)e^{bx}$ e $t = ((a + bx)/2)^{l+1} - ((a - bx)/2)^{l-1}$, onde os argumentos da rotina são a, b, x e l.

O código **main.f** utilizado para o cálculo dos valores de r, s e t, para $a = 1$, $b = 0.5$ e $l = 0.5$, no intervalo $[0, 1]$ é apresentado no apêndice A, seção A.1.2. As subrotinas que calculam os valores de r, s e t, dados a, b, l e x, foram incluídas no módulo **ex2Subs**. Para a alocação dinâmica dos vetores que contém os valores calculados, foi elaborado o módulo **dynamicalArrays** cujo código é apresentado no início da seção A.1. A plotagem dos gráficos foi feita através de uma implementação em Python, cujo código **plot.py** também é apresentado na seção A.1.2. A Figura 2 apresenta os resultados obtidos.

Figura 2 – Valores de $r(x)$, $s(x)$ e $t(x)$, para $a = 1$, $b = 0.5$ e $l = 0.5$, no intervalo $[0, 1]$.



1.2.3 Exercício 3

Dado um vetor x de 20 compontentes:

(a) Escrever uma rotina para calcular o vetor y cujas componentes estão em ordem reversa em relação às componentes de x.

(b) Escreva uma rotina para colocar em ordem crescente as componentes de x, porém usando a mesma área de memória de x.

O código **main.f** utilizado para ordenação do vetor x e para determinação do vetor y é apresentado no apêndice A, seção A.1.3. As subrotinas **sortInverseArray** que determina o vetor y e **bubbleSort** que ordena o vetor x foram incluídas no módulo **ex3Subs**.

1.2.4 Exercício 4

Dada uma matriz quadrada simétria $A(50, 50)$, escrever uma rotina para trocar a diagonal principal pela diagonal secundária a partir do termo $(2, 2)$ até o termo $(49, 49)$. Ajuste a rotina para admitir uma matriz de ordem arbitrária (n, n), onde n é um inteiro lido de um arquivo de dados, junto aos termos da própria matriz.

Para criação da matriz simétrica A, foi criada a subrotina **createSimMatrix(n)**, que recebe o número inteiro n e cria uma matriz simétrica $n \times n$. A troca da diagonal principal pela secundária a partir do segundo termo até o penúltimo termo da diagonal é feita através da rotina **changeDiagonal**. Ambas foram incluídas no módulo **ex4Subs**. O código **main.f** pode ser encontrado no apêndice A, seção A.1.4.

1.2.5 Exercício 5

Escrever uma rotina para calcular o produto escalar entre dois vetores dados, $x(n)$, $y(n)$, com n também variável, dado como argumento.

Para criação dos vetores x e y, foi utilizado o módulo **dynamicalArrays**, cujo código é apresentado no apêndice A, no início da seção A.1. O produto interno entre os vetores x e y é calculado através da subrotina **innerProduct**, parte do módulo **ex5Subs**. Estes códigos são apresentados na seção A.1.5.

1.2.6 Exercício 6

Escreva uma rotina para calcular todas as raízes de um polinômio real de segundo grau. Os coeficientes devem ser lidos de um arquivo de texto, e a resposta mostrada em outro arquivo.

O código é apresentado no apêndice A, seção A.1.6. Para o caso do polinômio possuir duas raízes reais e distintas, os resultados são armazenados em sequência no arquivo de texto. Para o caso do polinômio possuir duas raízes reais iguais, este resultado é armazenado uma única vez no arquivo de texto. Por último, para o caso do polinômio possuir duas raízes complexas, os resultados são armazenados na forma dos pares $(\mathrm{Re}, \mathrm{Im})$ e $(\mathrm{Re}, -\mathrm{Im})$ no arquivo de texto.

1.2.7 Exercício 7

Considere um segmento reto contido no plano $0xy$, definido pelas coordenadas de dois nós, 1 e 2, de coordenadas (x_1, y_1) e (x_2, y_2), respectivamente. Um sistema auxiliar de coordenadas $\overline{xyz}$ é definido com origem no nó 1, através de uma rotação em torno do eixo z pelo ângulo θ, medido a partir do eixo x, no sentido anti-horário até o eixo $\overline{x}$. Dessa forma, o eixo $\overline{x}$ tem o sentido do nó 1 para o nó 2. Construa uma subrotina para determinar θ no intervalo $[0, 2\pi]$, usando as coordenadas nodais. Gere tabela mantendo (x_1, y_1) fixo e modificando (x_2, y_2) de forma a ter ângulos em diversos valores na faixa $[0, 2\pi]$.

Para cálculo do ângulo de rotação θ, foi criada a subrotina **rotation**, pertencente ao módulo **ex7Subs**. O código é apresentado no apêndice A, seção A.1.7.

1.2.8 Exercício 8

Considere uma matriz triangular superior, isto é, uma matriz simétrica cujos termos armazenados são apenas aqueles acima da diagonal principal. Esses termos são armazenados em um vetor $\boldsymbol{V}$, coluna após coluna, da seguinte forma:

$$\boldsymbol{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots \\ & A_{22} & A_{23} & \dots \\ & & A_{33} & \dots \\ & & & \ddots \end{bmatrix}_{N\times N} \quad \rightarrow \text{armazenados como: } \boldsymbol{V} = \begin{Bmatrix} A_{11} \\ A_{12} \\ A_{13} \\ A_{22} \\ A_{23} \\ A_{33} \\ \vdots \\ A_{NN} \end{Bmatrix}_{M}$$

O número de termos no vetor é obtido pela fórmula (facilmente deduzível): $M = N(N+1)/2$. Um termo arbitrário A_{ij} de $\boldsymbol{A}$ pode ser localizado na posição m de $\boldsymbol{V}$ por: $m = (j-1)j/2 + i$. Por exemplo, o termo A_{23} está em $\boldsymbol{V}(5)$.

(a) Programa uma subrotina que receba uma matriz de ordem $N \leq 8$ armazenada na forma triangular superior num vetor $\boldsymbol{V}$, e imprima os termos com a diagramação visual de uma matriz triangular superior.

A criação da matriz simétrica é feita através da subrotina **createSimMatrix**. Esta é armazenada num arquivo de texto que é lido através da subrotina **readSimMatrix**. A alocação da matriz triangular superior é feita através da subrotina **storeSimMatrix**. Seus termos são impressos com a diagramação de uma matriz triangular inferior através da subrotina **printTriSupMatrix**. Os códigos são apresentados no apêndice A, seção A.1.8.

(b) Programe uma subrotina que receba uma matriz $\boldsymbol{A}$ de ordem N qualquer, armazenada na forma triangular superior num vetor $\boldsymbol{V}$, e um vetor $\boldsymbol{X}$ também de ordem N. Programe uma subrotina que faça o produto $\boldsymbol{AX} = \boldsymbol{F}$. Em seguida, imprima o resultado $\boldsymbol{F}$.

O produto da matriz A, na forma do vetor V, pelo vetor X é feito através da subrotina **multMatrixArray**, que recebe como argumentos os vetores V e X e também o vetor F que contém o resultado do produto AX. Os códigos são apresentados no apêndice A, seção A.1.8.

1.2.9 Exercício 9

Programe um conjunto de rotinas para leitura de dados de elementos finitos a partir de um arquivo texto.

(a) Uma rotina para leitura de dados de coordenadas nodais. Cada linha contém: "no coorX coorY coorZ ".

(b) Uma rotina para leitura de dados de conectividade de elementos de até quatro nós. Cada linha contém: "elem noI noJ noK noL ".

- Cada rotina de leitura deve também enviar os valores lidos para compor um relatório num

arquivo texto de saída. Os valores lidos devem permanecer à disposição no programa principal em matrizes de dimensões "coor(NNOS,3)" e "iconec(NELEM,5)"onde NNOS e NELEM são os primeiros valores lidos no arquivo de dados, que serão em seguida utilizados para fazer a alocação dinâmica das matrizes de dados.

A leitura do arquivo que contém as coordenadas dos nós é feita através da subrotina **readNodesCoordinates** que recebe como argumentos o nome do arquivo e a matriz que armazenará as coordenadas dos nós. A leitura do arquivo que contém a conectividade dos elementos é feita através da subrotina **readElementConnectivity** que recebe como argumentos o nome do arquivo e a matriz que armazenará a conectividade dos elementos. Os códigos são apresentados no apêndice A, seção A.1.9.

2 Trabalho 2

2.1 Exercício 1

Explique qual a diferença entre o conjunto Kin e Kin_h.

$Kin(\Omega)$, denominado *conjunto de funções de teste*, é o conjunto de todas as funções que satisfazem às condições de serem contínuas e diferenciáveis por partes no domínio Ω do problema e às condições de contorno de Dirichlet deste. Além disso, suas primeiras derivadas devem ser integráveis ao quadrado em Ω. Somente as funções deste conjunto, cuja dimensão é infinita, podem ser solução do problema.

Seja $u(\boldsymbol{x})$ uma função candidata à solução do nosso problema. Denotando o contorno em que se prescreve $u(\boldsymbol{x}) = \overline{u}(\boldsymbol{x})$ (contorno essencial) como Γ_u, simbolicamente, podemos escrever

$$Kin(\Omega) = \{u(\boldsymbol{x}); u(\boldsymbol{x}) = \overline{u}(\boldsymbol{x}) \; \forall \boldsymbol{x} \in \Gamma_u\}$$

Por outro lado, o conjunto $Kin_h(\Omega_h)$ representa um conjunto de dimensão finita que contém as funções candidatas à solução aproximada do problema, no domínio discretizado Ω_h. $Kin_h(\Omega_h)$ é um subconjunto de $Kin(\Omega)$, isto é, $Kin_h \subset Kin$.

$Var(\Omega)$, denominado *conjunto de variações ou funções peso*, é o conjunto, de dimensão infinita, de todas as funções que satisfazem às condições de serem contínuas e diferenciáveis por partes no domínio Ω do problema e que possuem valor nulo no contorno essencial. Seja $\hat{u}(\boldsymbol{x})$ uma função tal que $\hat{u}(\boldsymbol{x}) = 0$ no contorno essencial, simbolicamente, podemos escrever

$$Var(\Omega) = \{\hat{u}(\boldsymbol{x}); \hat{u}(\boldsymbol{x}) = 0 \; \forall \boldsymbol{x} \in \Gamma_u\}$$

Já o conjunto $Var_h(\Omega_h)$ representa um conjunto de dimensão finita, subconjunto de $Var(\Omega)$, isto é, $Var_h(\Omega_h) \subset Var(\Omega)$, que contém as funções peso discretizadas.

2.2 Exercício 2

Qual a caracterização qualitativa de funções de dimensão finita e infinita?

Funções de dimensão finita são quaisquer funções que podem ser descritas por um número finito de parâmetros. Por outro lado, funções de dimensão infinita requerem um número infinito de parâmetros. Um exemplo de função infinita é a função sen(x) expandida

numa série de Maclaurin dada por

$$\text{sen}(x) = \sum_{j=0}^{\infty} \frac{(-1)^j}{(2j+1)!} x^{2j+1}$$

Uma aproximação de dimensão finita da função $\text{sen}(x)$ pode ser dada por qualquer série de Maclaurin truncada em $N > 0$, isto é

$$\text{sen}(x) \approx \sum_{j=0}^{N} \frac{(-1)^j}{(2j+1)!} x^{2j+1}$$

2.3 Exercício 3

Por que a matriz de rigidez de elementos finitos de condução de calor é simétrica?

A forma forte da equação governante da condução de calor é uma equação diferencial elíptica. No processo de obtenção da forma fraca, faz-se a integração por partes obtendo-se uma simetria entre as derivadas da função peso $\hat{u}$ e da temperatura T. A integral resultante é uma forma bilinear simétrica em $\hat{u}$ e T, diretamente responsável pela simetria da matriz de rigidez.

A integral original (obtida diretamente pelo método dos resíduos ponderados) é dada por $\int_\Omega k\hat{u}\nabla^2 T d\Omega$. Sendo $\varphi_i(x, y, z)$ as funções de forma, podemos escrever as aproximações

$$\hat{u}(x, y, z) = \sum_{i=1}^{ne} \varphi_l c_l$$

e

$$T(x, y, z) = \sum_{i=j}^{ne} \varphi_j T_j$$

sendo c_l, T_j constantes. Assim, podemos escrever

$$\nabla^2 T = \sum_{j=1}^{ne} \nabla^2(\varphi_j T_j) = \sum_{j=1}^{ne} \left(\nabla^2 \varphi_j\right) T_j$$

Voltando à integral, podemos escrever

$$\int_\Omega k\hat{u}\nabla^2 T d\Omega = \int_\Omega k \left(\sum_{l=1}^{ne} \varphi_l c_l\right) \left[\sum_{j=1}^{ne} \left(\nabla^2 \varphi_j\right) T_j\right] d\Omega$$

A equação acima deve ser válida para $\hat{u}$ qualquer, isto é, para qualquer conjunto de valores de c_l. Tomando todos os valores de c_l como sendo nulos, com exceção de $c_i = 1$,

podemos reescrever a equação acima na forma

$$\int_\Omega k\varphi_i \sum_{j=1}^{ne} \left(\nabla^2\varphi_j\right) T_j d\Omega = \sum_{j=1}^{ne} \underbrace{\left(\int_\Omega k\varphi_i \nabla^2\varphi_j d\Omega\right)}_{K_{ij}} T_j$$

e pode-se notar que $K_{ij} \neq K_{ji}$, donde a matriz de rigidez obtida seria assimétrica. Por outro lado, após integração por partes, tem-se a integral $\int_\Omega k\boldsymbol{\nabla}\hat{u}\cdot\boldsymbol{\nabla}T d\Omega$. Na forma aproximada, podemos escrever

$$\boldsymbol{\nabla}\hat{u} = \sum_{l=1}^{ne} \left(\boldsymbol{\nabla}\varphi_l\right) c_l$$

e

$$\boldsymbol{\nabla}T = \sum_{j=1}^{ne} \left(\boldsymbol{\nabla}\varphi_j\right) T_j$$

Voltando à integral, podemos escrever

$$\int_\Omega k\boldsymbol{\nabla}\hat{u}\cdot\boldsymbol{\nabla}T d\Omega = \int_\Omega k \left[\sum_{l=1}^{ne} \left(\boldsymbol{\nabla}\varphi_l\right) c_l\right] \cdot \left[\sum_{j=1}^{ne} \left(\boldsymbol{\nabla}\varphi_j\right) T_j\right] d\Omega$$

Tomando todos os valores de c_l como sendo nulos, com exceção de $c_i = 1$, podemos reescrever a equação acima na forma

$$\int_\Omega k\boldsymbol{\nabla}\varphi_i \cdot \left[\sum_{j=1}^{ne} \left(\boldsymbol{\nabla}\varphi_j\right) T_j\right] d\Omega = \sum_{j=1}^{ne} \underbrace{\left[\int_\Omega k\boldsymbol{\nabla}\varphi_i \cdot \boldsymbol{\nabla}\varphi_j d\Omega\right]}_{K_{ij}} T_j$$

e pode-se notar que $K_{ij} = K_{ji}$, donde a matriz de rigidez obtida é simétrica, conforque queríamos demonstrar.

2.4 Exercício 4

Por que as matrizes de elementos finitos são esparsas? O que é largura de banda? Ela sempre existe?

As matrizes de rigidez são esparsas por conta das funções de interpolação utilizadas. Estas são construídas em torno de cada nó i de forma que sejam não-nulas apenas nos elementos que contém i e sejam nulas em todos os demais elementos. Portanto, na construção do sistema linear de equações, cada nó i relaciona-se apenas com os nós contidos nos elementos que contém i.

Assim, a linha i da matriz de rigidez, associada à equação que determina o valor da variável calculada no nó i, possui valores não-nulos somente nas posições relativas aos nós vizinhos do nó i e valores nulos em todas as demais posições. Uma vez que, em geral, o número de nós vizinhos a um nó i é muito menor que o número total de nós no domínio discretizado, a matriz torna-se esparsa.

Nesta equação que determina o valor da variável no nó i, o coeficiente relativo ao nó i encontra-se na diagonal principal. Os coeficientes relativos aos nós vizinhos a i organizam-se em torno da diagonal principal em forma de *banda*. A largura de banda corresponde à maxima distância horizontal entre a diagonal principal e o último termo não-nulo da linha. Esta largura é fortemente influenciada pela sequência de numeração dos nós.

Uma vez que, em geral, as matrizes de elementos finitos são grandes, a estrutura de bandas pode ser utilizada na elaboração de um armazenamento mais eficiente dos termos da matriz de rigidez. Sendo a matriz de rigidez simétrica, pode-se armazenar apenas a matriz triangular superior ou inferior. Se a matriz possui estrutura de banda, pode-se armazenar apenas os elementos sob a banda. Isto acarreta numa economia de área de armazenamento e de processamento.

A largura de banda sempre existe e torna-se maior em malhas complexas não-estruturadas em que não é possível evidenciar uma melhor sequência de numeração dos nós. Neste caso, pode-se fazer uso de algoritmos desenvolvidos com o objetivo de fazer a numeração dos nós de forma a minimizar a largura de banda.

2.5 Exercício 5

Por que o PTV, apesar de envolver um "trabalho", não é uma relação termodinâmica?

O *Princípio dos Trabalhos Virtuais* (PTV) postula que as tensões $\boldsymbol{\sigma}(\boldsymbol{x})$ estão em equilíbrio com o carregamento externo $\boldsymbol{f}$ e $\boldsymbol{b}$ se, e somente se, os trabalhos virtuais interno e externos são iguais para qualquer função peso $\hat{u} \in Var(\Omega)$ [1].

A função peso neste caso representa um deslocamento virtual (ou fictício), isto é, não tem relação com os esforços aplicados. O PTV representa um equilíbrio termodinâmico que admite uma infinidade de soluções, uma vez que é válido para qualquer $\hat{u} \in Var(\Omega)$. No entando, nem todas estas soluções são termodinamicamente admissíveis. Portanto, o PTV não pode ser considerado uma relação termodinâmica.

2.6 Exercício 6

Por que, na dedução do PTV, a função peso é requerida ser nula em Γ_u?

As funções $\hat{u} \in Var(\Omega)$ são obtidas pela diferença entre duas funções quaisquer $u_i, u_j \in Kin(\Omega)$. Uma vez que para todo $\boldsymbol{x} \in \Gamma_u$ temos $u_i(\boldsymbol{x}) = u_j(\boldsymbol{x}) = \overline{u}(\boldsymbol{x})$, então devemos ter

$$\hat{u}(\boldsymbol{x}) = u_i(\boldsymbol{x}) - u_j(\boldsymbol{x}) = 0 \; \forall (\boldsymbol{x}) \in \Gamma_u$$

2.7 Exercício 8

Considere o seguinte problema de valor no contorno:

$$\begin{gathered} u_{,xx} + 4u = 12 \text{ para } x \in \Omega, \\ u_{,x}(0) = 4,399, \\ u(1) = 1. \end{gathered} \tag{2.1}$$

onde $\Omega = x \in \mathbb{R}$ tal que $x \in (0; 1)$.

(a.1) Determine a forma fraca simétrica correspondente, utilizando o método dos resíduos ponderados.

Integrando o produto da equação diferencial $D\{u\} = 0$ dada pela função peso $\hat{u}$, no domínio Ω (método dos resíduos ponderados), temos

$$\int_0^1 (u_{,xx} + 4u - 12)\, \hat{u} dx = 0 \therefore \int_0^1 (u_{,xx}\hat{u} + 4u\hat{u} - 12\hat{u})\, dx = 0 \tag{2.2}$$

Integrando por partes,

$$\int_0^1 u_{,xx}\hat{u} dx = u_{,x}\hat{u}\Big|_0^1 - \int_0^1 u_{,x}\hat{u}_{,x} dx$$

e substituindo na Equação 2.2, obtemos

$$u_{,x}(1)\hat{u}(1) - u_{,x}(0)\hat{u}(0) - \int_0^1 (u_{,x}\hat{u}_{,x} - 4u\hat{u} + 12\hat{u}) dx = 0$$

Sabemos que $\hat{u}$ é nula no contorno essencial. Uma vez que o problema é definido com uma condição de contorno de Dirichlet, $\hat{u}$ deve ser igual a zero neste ponto. Assim, temos $\hat{u}(1) = 0$, portanto

$$\int_0^1 (u_{,x}\hat{u}_{,x} - 4u\hat{u} + 12\hat{u}) dx = -4,399\hat{u}(0) \tag{2.3}$$

que é a forma fraca simétrica correspondente do problema.

(a.2) Defina os conjuntos Kin e Var.

Neste caso, temos

$$Kin(\Omega) = \{u(x); u(1) = 1\}$$

e

$$Var(\Omega) = \{\hat{u}(x); \hat{u}(1) = 0\}$$

(b) Obter uma aproximação de Galerkin usando a aproximação $u_G(x) = 3 - 2x + ax(x-1)$.

Substituindo $\hat{u} = \dfrac{du_G}{da} = x^2 - x$ e $u = u_G = 3 - 2x - ax + ax^2$ na Equação 2.3, obtemos

$$\int_0^1 \left[(-2 - a + 2ax)(2x - 1) - 4(3 - 2x - ax + ax^2)(x^2 - x) + 12(x^2 - x)\right] dx = 0$$

$$\therefore \int_0^1 \left[-4ax^4 + 8ax^3 - 4ax + a + 8x^3 - 8x^2 - 4x + 2\right] dx = 0$$

$$-\frac{4a}{5} + 2a - 2a + a + 2 - \frac{8}{3} - 2 + 2 = 0$$

$$\therefore a = \frac{10}{3}$$

3 Trabalho 3

3.1 Exercício 5

Dadas as funções Lagrangeanas quadráticas unidimensionais no domínio $-1 \leq r \leq 1$, determine as funções Lagrangeanas bi-quadráticas do elemento plano de 9 nós.

As funções Lagrangeanas quadráticas unidimensionais são dadas por

$$L_1(r) = \frac{r}{2}(r-1), \quad L_2(r) = (1-r^2), \quad L_3(r) = \frac{r}{2}(r+1)$$

em que $-1 \leq r \leq 1$ e o elemento unidimensional é composto de 3 nós. A família de funções Lagrangeanas bi-quadráticas do elemento plano de 9 nós pode ser formada pelo produto dos polinômios da família quadrática unidimensional, nas direções r e s. Assim, temos

	$L_1(r)$	$L_2(r)$	$L_3(r)$
$L_1(s)$	$\psi_1(r,s)$	$\psi_5(r,s)$	$\psi_2(r,s)$
$L_2(s)$	$\psi_8(r,s)$	$\psi_9(r,s)$	$\psi_6(r,s)$
$L_3(s)$	$\psi_4(r,s)$	$\psi_7(r,s)$	$\psi_3(r,s)$

donde

$$\psi_1(r,s) = L_1(r)L_1(s) = \frac{1}{4}rs(r-1)(s-1)$$

$$\psi_2(r,s) = L_3(r)L_1(s) = \frac{1}{4}rs(r+1)(s-1)$$

$$\psi_3(r,s) = L_3(r)L_3(s) = \frac{1}{4}rs(r+1)(s+1)$$

$$\psi_4(r,s) = L_1(r)L_3(s) = \frac{1}{4}rs(r-1)(s+1)$$

$$\psi_5(r,s) = L_2(r)L_1(s) = \frac{1}{2}s(1-r^2)(s-1)$$

$$\psi_6(r,s) = L_3(r)L_2(s) = \frac{1}{2}r(r+1)(1-s^2)$$

$$\psi_7(r,s) = L_2(r)L_3(s) = \frac{1}{2}s(1-r^2)(s+1)$$

$$\psi_8(r,s) = L_1(r)L_2(s) = \frac{1}{2}r(r-1)(1-s^2)$$

$$\psi_9(r,s) = L_2(r)L_2(s) = (1-r^2)(1-s^2)$$

3.2 Exercício 6

Considere um elemento triangular quadrático de 6 nós. Esboce a função de aproximação associada a um nó de vértice e a um nó de centro de lado.

Para um elemento triangular quadrático de 6 nós, temos as seguintes funções de aproximação para os nós nos vértices

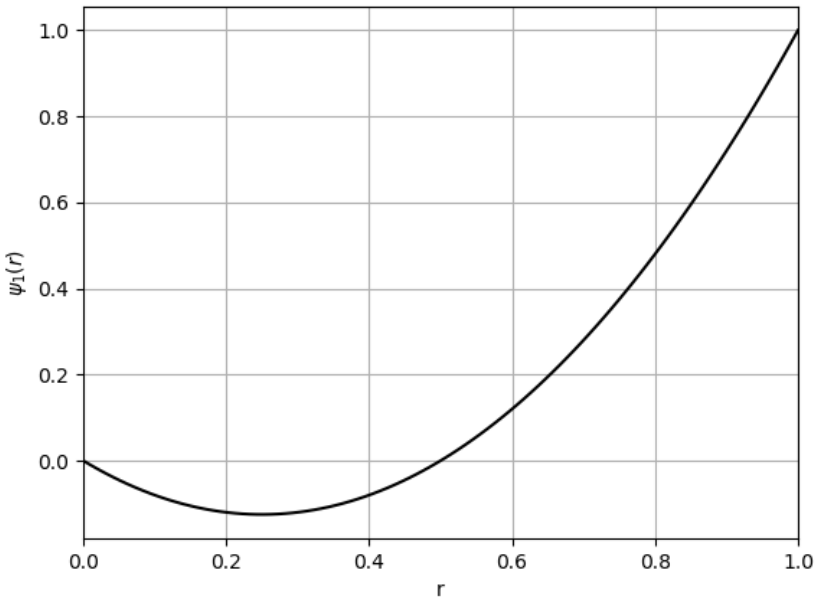
$$\psi_1(r) = r(2r-1)$$

$$\psi_2(s) = s(2s-1)$$

$$\psi_3(t) = t(2t-1)$$

O esboço da função $\psi_1(r)$ é apresentado na Figura 3

Figura 3 – Comportamento de ψ_1 para $r \in [0, 1]$.



Para os nós de centro de lado, temos as seguintes funções de aproximação

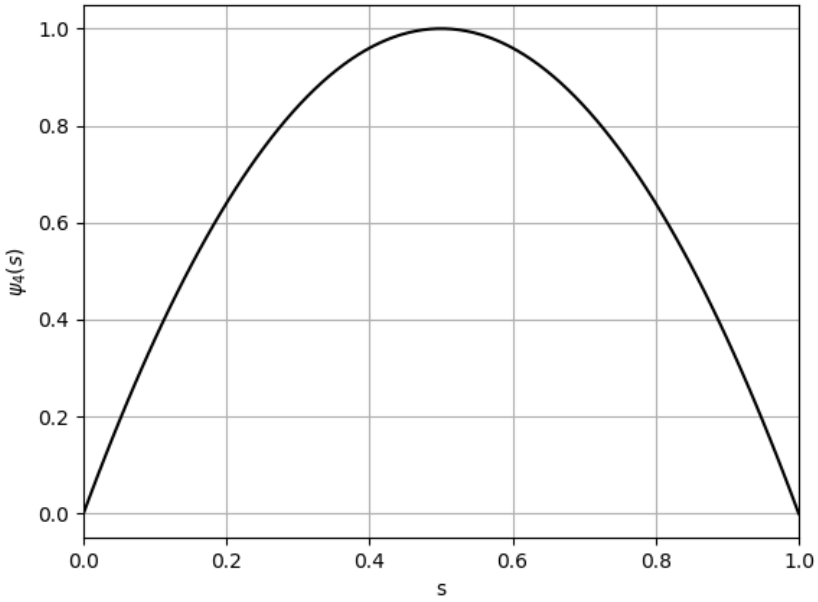
$$\psi_4(r, s) = 4rs$$

$$\psi_5(s, t) = 4st$$

$$\psi_6(r, t) = 4rt$$

O esboço da função $\psi_4(r, s)$, para $t = 0$ e $r = 1 - s$, é apresentado na Figura 4

Figura 4 – Comportamento de ψ_4 para $r \in [0,1]$ e $s \in [0,1]$.



3.3 Exercício 7

Para o problema elastostático-linear, considere:

Dado $\boldsymbol{b} : \Omega \to R^3, \overline{\boldsymbol{t}} : \Gamma_f \to R^3$, obter $\boldsymbol{u} \in Var$ tal que $a(\boldsymbol{u}, \hat{\boldsymbol{u}}) = (\boldsymbol{b}, \hat{\boldsymbol{u}}) + (\overline{\boldsymbol{t}}, \hat{\boldsymbol{u}})_{\Gamma_f}$, onde $\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{0}$ para todo $\boldsymbol{x} \in \Gamma_u$ e a forma bilinear $a(\boldsymbol{u}, \hat{\boldsymbol{u}})$ é positiva-definida. Prove a unicidade da solução.

Para o problema elastostático-linear, temos

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega \boldsymbol{\sigma}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) \, d\Omega = \int_\Omega \boldsymbol{C}\boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{v}) \, d\Omega = \int_\Omega \boldsymbol{C}\boldsymbol{\nabla}^s\boldsymbol{u} \cdot \boldsymbol{\nabla}^s\boldsymbol{v} \, d\Omega$$

Dada a relação acima, podemos notar que

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega \boldsymbol{C}\boldsymbol{\nabla}^s\boldsymbol{u} \cdot \boldsymbol{\nabla}^s\boldsymbol{v} \, d\Omega = \int_\Omega \boldsymbol{C}\boldsymbol{\nabla}^s\boldsymbol{v} \cdot \boldsymbol{\nabla}^s\boldsymbol{u} \, d\Omega = a(\boldsymbol{v}, \boldsymbol{u})$$

e o operador $a(\cdot, \cdot)$ é simétrico. Também nota-se que

$$a(\alpha\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega \boldsymbol{C}\boldsymbol{\nabla}^s \left(\alpha\boldsymbol{u}\right) \cdot \boldsymbol{\nabla}^s\boldsymbol{v} \, d\Omega = \alpha \int_\Omega \boldsymbol{C}\boldsymbol{\nabla}^s\boldsymbol{u} \cdot \boldsymbol{\nabla}^s\boldsymbol{v} \, d\Omega = \alpha a(\boldsymbol{u}, \boldsymbol{v})$$

$$a(\boldsymbol{u}, \beta\boldsymbol{v}) = \int_\Omega \boldsymbol{C}\boldsymbol{\nabla}^s\boldsymbol{u} \cdot \boldsymbol{\nabla}^s \left(\beta\boldsymbol{v}\right) \, d\Omega = \beta \int_\Omega \boldsymbol{C}\boldsymbol{\nabla}^s\boldsymbol{u} \cdot \boldsymbol{\nabla}^s\boldsymbol{v} \, d\Omega = \beta a(\boldsymbol{u}, \boldsymbol{v})$$

e o operador $a(\cdot, \cdot)$ é bilinear.

Ainda, condicionada à propriedade de positividade definida da matriz elástica $\boldsymbol{C}$, podemos escrever $\boldsymbol{C\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) \geq 0 \; \forall \boldsymbol{\varepsilon} \in \mathbb{R}^6$ e $\boldsymbol{C\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) = 0$ se, e somente se, $\boldsymbol{\varepsilon}(\boldsymbol{u}) = \boldsymbol{0}$. Dadas as condições de contorno $\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{0} \; \forall \boldsymbol{x} \in \Gamma_u$, os movimentos de corpo rígido são impedidos e $\boldsymbol{\varepsilon} \neq \boldsymbol{0} \; \forall \boldsymbol{u} \neq \boldsymbol{0}$ [2]. Assim, podemos escrever

$$\begin{cases} \boldsymbol{C\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) > 0 \; \forall \boldsymbol{u} \neq \boldsymbol{0} \\ \boldsymbol{C\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) = 0 \Leftrightarrow \boldsymbol{u} = \boldsymbol{0} \end{cases}$$

Integrando no domínio Ω, obtemos

$$\begin{cases} a(\boldsymbol{u}, \boldsymbol{u}) = \displaystyle\int_\Omega \boldsymbol{C\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) d\Omega > 0 \; \forall \boldsymbol{u} \neq \boldsymbol{0} \\ a(\boldsymbol{u}, \boldsymbol{u}) = \displaystyle\int_\Omega \boldsymbol{C\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u}) d\Omega = 0 \Leftrightarrow \boldsymbol{u} = \boldsymbol{0} \end{cases}$$

e o operador $a(\cdot, \cdot)$ é positivo-definido.

Vamos supor que $\boldsymbol{u}_1, \boldsymbol{u}_2 \in Var$ são soluções do problema dado. Assim, devemos ter

$$\begin{cases} a(\boldsymbol{u}_1, \hat{\boldsymbol{u}}) = (\boldsymbol{b}, \hat{\boldsymbol{u}}) + (\bar{\boldsymbol{t}}, \hat{\boldsymbol{u}})_{\Gamma_f} \\ a(\boldsymbol{u}_2, \hat{\boldsymbol{u}}) = (\boldsymbol{b}, \hat{\boldsymbol{u}}) + (\bar{\boldsymbol{t}}, \hat{\boldsymbol{u}})_{\Gamma_f} \end{cases}, \; \forall \hat{\boldsymbol{u}} \in Var$$

Subtraindo a segunda equação da primeira, obtemos

$$a(\boldsymbol{u}_1 - \boldsymbol{u}_2, \hat{\boldsymbol{u}}) = 0, \; \forall \hat{\boldsymbol{u}} \in Var$$

Notando que $\boldsymbol{u}_1 - \boldsymbol{u}_2 \in Var$, podemos tomar $\hat{\boldsymbol{u}} = \boldsymbol{u}_1 - \boldsymbol{u}_2$, uma vez que a relação acima é válida para qualquer $\hat{\boldsymbol{u}} \in Var$, e obtemos

$$a(\boldsymbol{u}_1 - \boldsymbol{u}_2, \boldsymbol{u}_1 - \boldsymbol{u}_2) = 0; \; \boldsymbol{u}_1, \boldsymbol{u}_2 \in Var$$

Conforme demonstrado anteriormente, o operador $a(\cdot, \cdot)$ é positivo-definido. Assim, $a(\boldsymbol{v}, \boldsymbol{v}) = 0$ se, e somente se, $\boldsymbol{v} = \boldsymbol{0}$. Portanto $\boldsymbol{u}_1 - \boldsymbol{u}_2 = \boldsymbol{0}$, donde $\boldsymbol{u}_1 = \boldsymbol{u}_2$ e a solução do problema dado é única.

4 Trabalho 4

4.1 Exercício 8

Programar forças de corpo com dados nas direções normal e tangencial ao contorno.

O programa **Elasticidade2D.for** faz inicialmente a leitura do arquivo *dadosEL2D.txt*, em que constam, dentre outras informações do problema, as forças de corpo. Neste arquivo, são incluídas as forças de corpo. A leitura do arquivo é feita através da subrotina **ledados** que armazenda no array **cargad** os valores e direções em que as forças de corpo atuam, para cada elemento. O trecho do arquivo *dadosEL2D.txt* em que se inclui as forças de corpo pode ser verificado na Listing 4.1. Por conveniência, os demais trechos foram omitidos.

Listing 4.1 – dadosEL2D.txt

```
!(...)
-1 0 0 0 ! sinaliza o final do bloco de dados de condicoes de
! cont.
1 0.0 1 ! Forca de corpo. ELEMENTO = 1, valor = 0 N/mm3,
! direcao = 1(X)
1 0.0 2 ! Forca de corpo. ELEMENTO = 1, valor = 0 N/mm3,
! direcao = 2(Y)
-1 0.0 0
!(...)
```

A subrotina **elemEPT2D** é quem gera a matriz de rigidez do elemento e o vetor força nodal aplicado (devido às forças de corpo) no elemento. Valor e direção armazenados no array **cargad** são passados às variáveis **fcorpo** e **idir**, esta última através da função **IDINT** que converte o valor real em inteiro. Mais adiante, caso **fcorpo** seja não-nulo, *i.e.* caso haja força de corpo atuando no elemento, o vetor **Felem** é montado com os valores e direções em que a força de corpo atua no elemento em seus nós.

O programa então segue com a montagem do sistema linear (sobreposição) e com a solução deste. A saída do programa, que lista no arquivo *saidaEL2D.txt*, dentre outras informações e resultados, as forças de corpo, é apresentada na Listing 4.2. Por conveniência, as demais saídas foram omitidas.

Listing 4.2 – saidaEL2D.txt

```
(...)
 Forcas de corpo nos elementos
 Elemento carga direcao = 1 0.0000000000000000 1
 Elemento carga direcao = 1 0.0000000000000000 2
 Elemento carga direcao = -1 0.0000000000000000 0
(...)
```

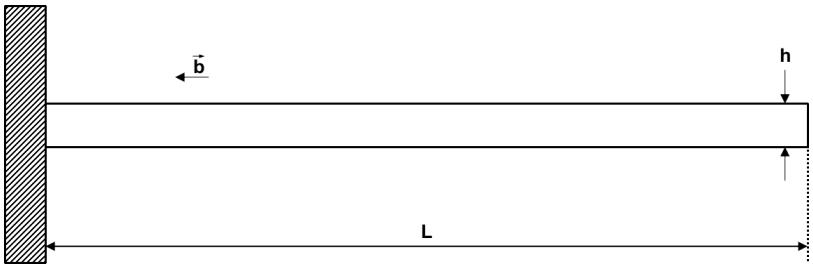
Para facilitar a plotagem dos campos de deslocamento, foi adicionada à subrotina **impU** que imprime os valores dos deslocamentos no arquivo de saída *saidaEL2D.txt* dois laços que escrevem nos arquivos *exportU.txt* e *exportV.txt* os campos de deslocamento u e v e suas posições ao longo dos eixos x e y. Estes laços são apresentados na Listing 4.3.

Listing 4.3 – Elasticidade2D.for

```
!(...)
        open(24,file="exportU.txt",status="unknown")
        do 24 n = 1,nnos
24      write(24,12) coor(1,n),(Uglob((n-1)*nglno+1))
12      format(2(1E12.5,5X),2(1E12.5))
        close(24)
        open(69,file="exportV.txt",status="unknown")
        do 69 n = 1,nnos
69      write(69,11) coor(2,n),(Uglob((n-1)*nglno+2))
11      format(2(1E12.5,5X),2(1E12.5))
        close(69)
!(...)
```

Para validação do programa, resolvemos o problema da barra horizontal de comprimento $L = 4000$ mm e seção transversal quadrada, de lado $h = 5$ mm, engastada na lateral esquerda e sujeita à força de corpo horizontal $b = -7,84 \cdot 10^{-5}$ N/mm^3, conforme mostra a Figura 5. A barra possui $E = 200 \cdot 10^3$ N/mm^2 e $\nu = 0.0$.

Figura 5 – Problema de validação: barra de comprimento L, engastada na lateral esquerda, sujeita à força de corpo horizontal b.



Uma vez que $L \gg h > e$, a equação do balanço mecânico em x pode ser simplificada para

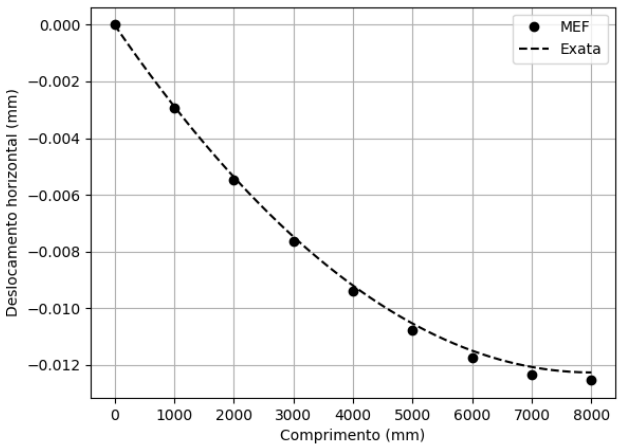
$$\frac{E(1-\nu)}{(1+\nu)(1-2\nu)} u_{,xx} + b = 0$$

donde, sabendo-se que $u(0) = 0$ e $u_{,x}(L) = 0$, obtém-se a solução

$$u(x) = \frac{b(1+\nu)(1-2\nu)}{E(1-\nu)} \left(L - \frac{x}{2}\right) x \tag{4.1}$$

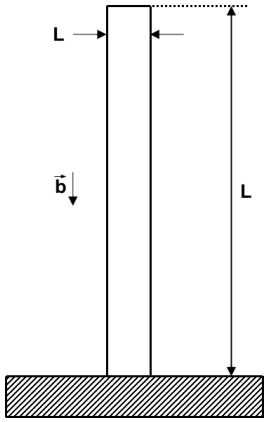
Os dados de entrada para uma malha cartesiana de 3×9 nós, com 4 elementos quadriláteros bilineares de 9 nós, no arquivo *dadosEL2Dbenchmark1.txt*, são apresentados no apêndice A, seção A.2. O campo de deslocamentos horizontais u obtido é plotado em relação à posição x do nó, através de uma implementação em Python, cujo código **plot1.py** também é apresentado na seção A.2. Nesta mesma implementação, plota-se a solução analítica dada pela Equação 4.1. A Figura 6 apresenta os resultados obtidos. Pode-se perceber que a solução obtida pelo MEF se aproxima da solução exata.

Figura 6 – Comparação entre a solução analítica e a solução numérica para o problema da barra horizontal.



Resolve-se o mesmo problema rotacionado em $-90°$, conforme mostra a Figura 7, a fim de verificar se a aplicação das forças de corpo funciona em ambas as direções.

Figura 7 – Problema de validação: barra de altura L, engastada na extremidade inferior, sujeita à força de corpo vertical b.

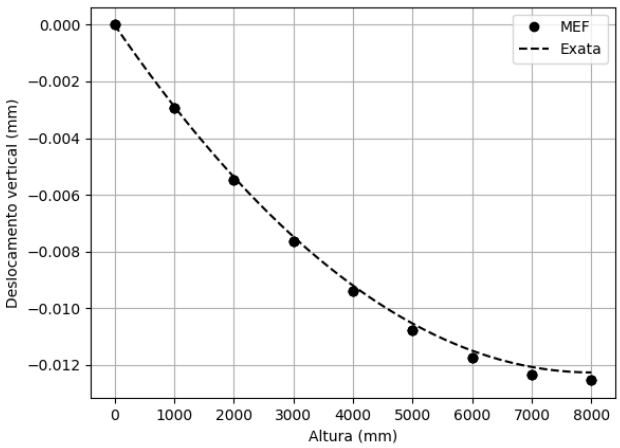


A solução analítica para este problema é dada por

$$v(y) = \frac{b(1+\nu)(1-2\nu)}{E(1-\nu)} \left(L - \frac{y}{2}\right) y \tag{4.2}$$

Os dados de entrada para uma malha cartesiana de 9×3 nós, com 4 elementos quadriláteros bilineares de 9 nós, no arquivo *dadosEL2Dbenchmark2.txt*, são apresentados no apêndice A, seção A.2. A Figura 8 apresenta os resultados obtidos. Pode-se perceber que a solução obtida pelo MEF se aproxima da solução exata.

Figura 8 – Comparação entre a solução analítica e a solução numérica para o problema da barra vertical.



4.2 Exercício 10a

Programar pós-processamento de tensões nos elementos via médias nodais.

O cálculo das tensões é feito através da subrotina **tens2D**. Na declaração de variáveis desta subrotina, foram incluídas as declarações das seguintes variáveis:

- **tensaoMediaNodal**: array de valores reais de tamanho NNOS $\times$ 3 que armazena as tensões calculadas em cada nó através da média nodal;
- **contagem**: array de valores inteiros que armazena o número de nós envolvidos na média nodal;
- **posicaoEquivalente**: array de valores inteiros que armazena as posições equivalentes dos nós locais do elemento na matriz de conectividade;
- **noeq**: variável de valor inteiro que armazena a posição equivalente de um nó local do elemento na matriz de conectividade.

A declaração é mostrada na Listing 4.4.

Listing 4.4 – Subrotina: Tens2D, Média Nodal (parte 1)

```
!(...)
!media nodal (parte 1)
dimension tensaoMediaNodal(nnos,3)
integer :: contagem(nnos)
integer, dimension(9) :: posicaoEquivalente=
        (/1,5,2,8,9,6,4,7,3/)
integer :: noeq
!(...)
```

A seguir, zera-se os valores armazenados nos arrays **contagem** e **tensaoMediaNodal**, conforme mostra a Listing 4.5.

Listing 4.5 – Subrotina: Tens2D, Média Nodal (parte 2)

```
!(...)
!media nodal (parte 2)
contagem=0
tensaoMediaNodal=0
!(...)
```

O cálculo da tensão via média nodal é feito através da soma dos valores nodais, dentro do laço que faz a varredura dos elementos da malha. O trecho apresentado na Listing 4.6 foi incluído no laço *DO* que corre os elementos da malha. Neste laço, o array **contagem** armazena o número de nós das nuvens de cada nó.

Listing 4.6 – Subrotina: Tens2D, Média Nodal (parte 3)

```
!(...)
!media nodal (parte 3)
do nol=1,nnoe ! corre os nos locais de NE
noeq=posicaoEquivalente(nol) ! posicao equivalente
nog=iconec(noeq,NE) ! no global corresp
do ngl=1,3 ! corre os gl do no
tensaoMediaNodal(nog,ngl)=tensaoMediaNodal(nog,ngl)
        +tensoesLOB(NE,ngl,nol)
contagem(nog)=contagem(nog)+1
enddo
enddo
!(...)
```

Na sequência, divide-se cada valor armazenado no array **tensaoMediaNodal** pelo valor correspondente no array **contagem**. Os valores obtidos são exportados para os arquivos *exportSigmaXMediaNodal.txt* e *exportSigmaYMediaNodal.txt*, juntamente com suas posições x e y. Estas operações são apresentadas na Listing 4.7.

Listing 4.7 – Subrotina: Tens2D, Média Nodal (parte 4)

```
!(...)
!media nodal (parte 4)
do nog=1,nnos ! corre os nos globais
contagem(nog)=contagem(nog)/3
do ngl=1,3 ! corre os gl do no
tensaoMediaNodal(nog,ngl)=tensaoMediaNodal(nog,ngl)
        /contagem(nog)
enddo
enddo
open(69,file="exportSigmaXMediaNodal.txt",status='unknown')
do 69 nog=1,nnos
69 write(69,699) coor(1,nog),tensaoMediaNodal(nog,1)
699 format(2(1E12.5,5X),2(1E12.5,5X))
```

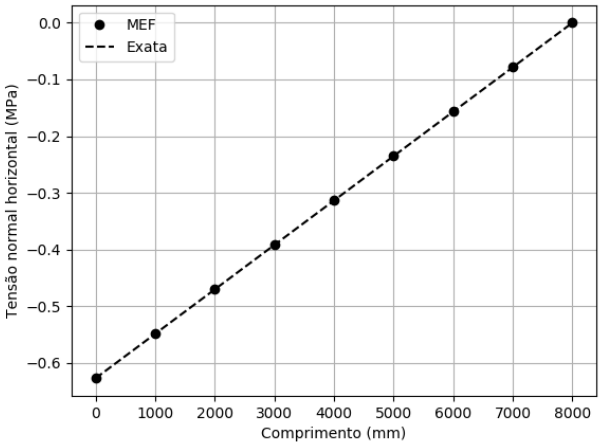
```
close(69)
open(24,file="exportSigmaYMediaNodal.txt",status='unknown')
do 24 nog=1,nnos
24 write(24,244) coor(2,nog),tensaoMediaNodal(nog,2)
244 format(2(1E12.5,5X),2(1E12.5,5X))
close(24)
!(...)
```

Para validação do programa, utilizamos os problemas apresentados pelas Figuras 5 e 6. Derivando a Equação 4.1 em relação a x e rearranjando, obtemos

$$\sigma_{xx} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} u_{,x} = b\,(L-x) \tag{4.3}$$

Foi utilizada a mesma malha usada no exercício anterior e a mesma entrada de dados, apresentada no apêndice A, seção A.2. O campo de tensões obtido é plotado em relação à posição x do nó, através do código em Python **plot1.py**, também apresentado na seção A.2. Na mesma implementação, plota-se a solução analítica dada pela Equação 4.4 e percebe-se que a solução obtida pelo MEF via médias nodais se aproxima bem da solução exata. A Figura 9.

Figura 9 – Comparação entre a solução analítica e a solução numérica para o problema da barra horizontal.

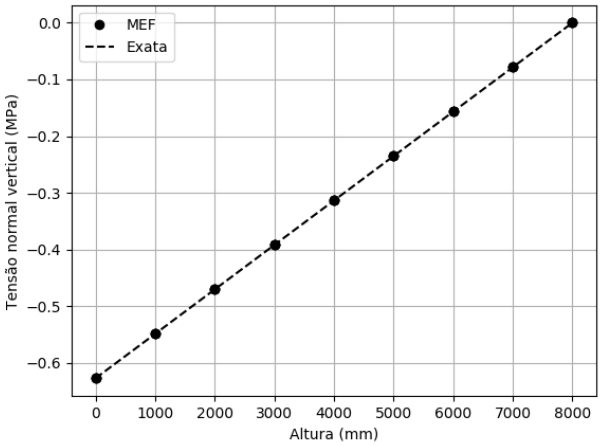


Derivando a Equação 4.2 em relação a y e rearranjando, obtemos

$$\sigma_{yy} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} v_{,y} = b\,(L-y) \tag{4.4}$$

A Figura 10 apresenta a comparação entre os valores de tensão obtidos pelo MEF via médias nodais e a solução analítica. Nota-se que o resultado obtido aproxima-se bem da solução exata.

Figura 10 – Comparação entre a solução analítica e a solução numérica para o problema da barra horizontal.



Referências

1 MENDONÇA, P. d. T. R.; FANCELLO, E. A. *O Método de Elementos Finitos Aplicado à Mecânica dos Sólidos*. Florianópolis: Editora Orsa Maggiore, 2019.

2 MALVERN, L. E. *Introduction to the Mechanics of a Continuous Medium*. New Jersey: Prentice Hall, 1969.

A Códigos

A.1 Trabalho 1.2 - Manual Fotran

Listing A.1 – module: dynamicalArrays

```
module dynamicalArrays
contains

subroutine push_back(oldArray,elementAdded)
implicit none

integer :: i,iSize
real*4, intent(in):: elementAdded
real*4, dimension(:), allocatable, intent(inout) :: oldArray
real*4, dimension(:), allocatable :: newArray

if(allocated(oldArray)) then
iSize=size(oldArray)
allocate(newArray(iSize+1))
do i=1,iSize
newArray(i)=oldArray(i)
end do
newArray(iSize+1)=elementAdded
deallocate(oldArray)
call move_alloc(newArray,oldArray)
else
allocate(oldArray(1))
oldArray(1)=elementAdded
end if

end subroutine push_back

end module dynamicalArrays
```

A.1.1 Exercício 1

Listing A.2 – function: func(x) (parte 1 de main.f)

```
function func(x) result(f)

real*4, intent(in) :: x
real*4 :: f

f=x**2+(1+2*x+3*x**2)**0.5

end function func
```

Listing A.3 – program: exercicio1 (parte 2 de main.f)

```
program exercicio1

use dynamicalArrays

real*4, dimension(:), allocatable :: y
real*4, dimension(0:4) :: a
integer*2 :: i
logical :: file_exists
integer*2 :: pointsHalf=1000
real*4 :: intervalSize=10

do i=-pointsHalf,pointsHalf
call push_back(y,(i*intervalSize)/pointsHalf)
end do

inquire(file="exercicio1.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="exercicio1.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="exercicio1.txt",form="formatted",status="new",
        action="write")
end if
```

```
do i=1,size(y)
a(0)=y(i)
a(1)=(6.9+y(i))/func(y(i))
a(2)=(21*y(i)+y(i)**4)/func(y(i))
a(3)=sin(y(i))/func(y(i))
a(4)=1/func(sin(y(i)))
! print *, a(:)
write(1,*) a(:)
end do

end program exercicio1
```

Listing A.4 – plot.py

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

parentDirectory=pathlib.Path(__file__).resolve().parents[0]

fileName=str(parentDirectory)+"/exercicio1.txt"
results=np.loadtxt(fname=fileName)

for i in range(1,5):
        plt.figure()
        plotName="a"+str(i)+".png"
        plt.plot(results[:,0],results[:,i],'-k')
        plt.xlabel('y')
        plt.ylabel('$a_'+str(i)+'(y)$')
        plt.grid(which='major',axis='both')
        plt.savefig(plotName)
```

A.1.2 Exercício 2

Listing A.5 – module: ex2Subs (parte 1 de main.f)

```
module ex2Subs
contains
```

```
subroutine rSub(a,b,x,r)

implicit none

real*4, intent(in) :: a,b,x
real*4 :: r

r=(a+b*x+x**2)**0.5

end subroutine rSub

subroutine sSub(a,b,x,s)

implicit none

real*4, intent(in) :: a,b,x
real*4 :: s
real*4, parameter :: pi=3.14159265359

s=cos(2*pi*x+a)*exp(b*x)

end subroutine sSub

subroutine tSub(a,b,l,x,t)

implicit none

real*4, intent(in) :: a,b,l,x
real*4 :: t

t=((a+b*x)*0.5)**(l+1)-((a-b*x)*0.5)**(l-1)

end subroutine tSub

end module ex2Subs
```

Listing A.6 – program: exercicio2 (parte 2 de main.f)

```
program exercicio2

use dynamicalArrays
use ex2Subs

real*4, dimension(:), allocatable :: x,r,s,t
real*4, dimension(0:3) :: export
integer*2 :: i
logical :: file_exists
integer*2 :: pointsNum=1000
real*4 :: intervalSize=1
real*4 :: a,b,l,xValue,rValue,sValue,tValue

a=1.
b=0.5
l=0.5

do i=0,pointsNum
xValue=(i*intervalSize)/pointsNum
call rSub(a,b,xValue,rValue)
call sSub(a,b,xValue,sValue)
call tSub(a,b,l,xValue,tValue)
call push_back(x,xValue)
call push_back(r,rValue)
call push_back(s,sValue)
call push_back(t,tValue)
end do

inquire(file="exercicio2.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="exercicio2.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="exercicio2.txt",form="formatted",status="new",
        action="write")
end if
```

```
do i=1,size(x)
export(0)=x(i)
export(1)=r(i)
export(2)=s(i)
export(3)=t(i)
! print *, export(:)
write(1,*) export(:)
end do

end program exercicio2
```

Listing A.7 – plot.py

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

parentDirectory=pathlib.Path(__file__).resolve().parents[0]

fileName=str(parentDirectory)+"/exercicio2.txt"
results=np.loadtxt(fname=fileName)

plt.figure()
plotName="exercicio2.png"
plt.xlabel('x')
plt.ylabel('y')
plt.grid(which='major',axis='both')

colours=['b','g','r']
legend=['r','s','t']

for i in range(1,4):
        plt.plot(results[:,0],results[:,i],'-',
                color=colours[i-1])

plt.legend(('r(x)','s(x)','t(x)'),loc='upper␣left',ncol=3)
plt.savefig(plotName)
```

A.1.3 Exercício 3

Listing A.8 – module: ex3Subs (parte 1 de main.f)

```
module ex3Subs
contains

subroutine sortInverseArray(oldArray,newArray)

implicit none

real*4, dimension(:), intent(in):: oldArray
real*4, dimension(:), intent(out) :: newArray
integer :: i,mySize

mySize=size(oldArray)

do i=1,mySize
newArray(i)=oldArray(mySize-i+1)
end do

end subroutine sortInverseArray

subroutine bubbleSort(myArray)

implicit none

real*4, dimension(:), intent(inout) :: myArray
integer :: i,mySize,counter=1
real*4 :: this,next

mySize=size(myArray)

do while(counter>0)
counter=0
do i=1,mySize-1
this=myArray(i)
next=myArray(i+1)
```

```
if(this>next) then
myArray(i)=next
myArray(i+1)=this
counter=counter+1
end if
end do
end do

end subroutine bubbleSort

end module ex3Subs
```

Listing A.9 – program: exercicio3 (parte 2 de main.f)

```
program exercicio3

use ex3Subs

logical :: file_exists
real*4, dimension(1:20) :: x,oldx
real*4, dimension(1:20) :: y
integer :: i
real*4, dimension(0:2) :: export
real*4 :: pi=3.14159265359

inquire(file="exercicio3.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="exercicio3.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="exercicio3.txt",form="formatted",status="new",
        action="write")
end if

do i=1,size(x)
x(i)=0.5*(exp(20.0-i)+exp(-20.0+i))
oldx(i)=x(i)
end do
```

```
call sortInverseArray(x,y)
call bubbleSort(x)

do i=1,size(x)
export(0)=oldx(i)
export(1)=y(i)
export(2)=x(i)
write(1,*) export(:)
end do

end program exercicio3
```

A.1.4 Exercício 4

Listing A.10 – module: ex4Subs (parte 1 de main.f)

```
module ex4Subs
contains

subroutine createSimMatrix(n)

logical :: file_exists
integer :: n,i,j
integer, dimension(:,:), allocatable:: myMatrix
integer, dimension(:), allocatable :: myArray

allocate(myMatrix(n,n))
allocate(myArray(n))

inquire(file="matrizLida.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="matrizLida.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="matrizLida.txt",form="formatted",status="new",
        action="write")
end if
```

```
do i=1,n
myMatrix(i,i)=1
do j=i,n-1
myMatrix(i,j+1)=myMatrix(i,j)+1
myMatrix(j+1,i)=myMatrix(i,j+1)
end do
end do

write(1,*) n
do i=1,n
do j=1,n
myArray(j)=myMatrix(i,j)
end do
write(1,*) myArray(:)
end do
close(1)

end subroutine createSimMatrix

subroutine changeDiagonal()

logical :: file_exists
integer :: n,i,j
integer, dimension(:,:), allocatable:: myMatrix
integer, dimension(:), allocatable :: myArray
integer :: value

open(unit=1,file="matrizLida.txt",form="formatted",status="old",
        action="read")
read(1,*) n
allocate(myMatrix(n,n))
allocate(myArray(n))

do i=1,n
read(1,*) (myMatrix(i,j),j=1,n)
end do
```

```
close(1)

inquire(file="exercicio4.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="exercicio4.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="exercicio4.txt",form="formatted",status="new",
        action="write")
end if

do i=2,n-1
value=myMatrix(i,i)
myMatrix(i,i)=myMatrix(i,n-(i-1))
myMatrix(i,n-(i-1))=value
end do

do i=1,n
do j=1,n
myArray(j)=myMatrix(i,j)
end do
write(1,*) myArray(:)
end do
close(1)

end subroutine changeDiagonal

end module ex4Subs
```

Listing A.11 – program: exercicio4 (parte 2 de main.f)

```
program exercicio4

use ex4Subs

call createSimMatrix(50)
call changeDiagonal()
```

```
end program exercicio4
```

A.1.5 Exercício 5

Listing A.12 – module: ex5Subs (parte 1 de main.f)

```
module ex5Subs
contains

subroutine innerProduct(array1,array2,n,result)

implicit none

integer, intent(in) :: n
real*4, dimension(:), intent(in):: array1,array2
real*4, intent(out) :: result
integer :: i

result=0.0

do i=1,n
result=result+array1(i)*array2(i)
end do

end subroutine innerProduct

end module ex5Subs
```

Listing A.13 – program: exercicio5 (parte 2 de main.f)

```
program exercicio5

use dynamicalArrays
use ex5Subs

real*4, dimension(:), allocatable :: x,y
integer :: i,n=10
real*4 :: baseX,baseY,pi=3.14159265359,result
```

```
do i=1,n
baseX=exp(-pi*i)
baseY=exp(pi*i)
call push_back(x,baseX)
call push_back(y,baseY)
end do

call innerProduct(x,y,n,result)

end program exercicio5
```

A.1.6 Exercício 6

Listing A.14 – program: exercicio6

```
program exercicio6

logical :: file_exists
real*4 :: a,b,c,delta
real*4 :: R1,R2,Re,Im
character(len=80) :: str1,str2

open(unit=1,file="coeficientes.txt",form="formatted",
        status="old",action="read")
read(1,*) a
read(1,*) b
read(1,*) c
close(1)

delta=b*b-4*a*c

inquire(file="exercicio5.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="exercicio5.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="exercicio5.txt",form="formatted",status="new",
        action="write")
```

```
end if

if(delta>0) then
R1=(-b+sqrt(delta))/(2*a)
R2=(-b-sqrt(delta))/(2*a)
write(1,'(F7.4)') R1
write(1,'(F7.4)') R2
else if(delta==0) then
Re=-b/(2*a)
write(1,'(F7.4)') Re
else
Re=-b/(2*a)
Im=sqrt(-delta)/(2*a)
write(str1,'(F7.4)') Re
write(str2,'(F7.4)') Im
write(1,*) "(",trim(str1),",",trim(str2),")"
write(str2,'(F7.4)') -Im
write(1,*) "(",trim(str1),",",trim(str2),")"
end if

close(1)

end program exercicio6
```

A.1.7 Exercício 7

Listing A.15 – program: exercicio7

```
module ex7Subs
contains

subroutine rotation(point1,point2,angle)

implicit none

real*4, dimension(1:2), intent(in) :: point1,point2
real*4, intent(out) :: angle
real*4 :: pi=3.14159265359
```

```
angle=atan((point2(2)-point1(2))/(point2(1)-point1(1)))
if(angle<0) then
angle=angle+2*pi
end if

end subroutine rotation

end module ex7Subs

program exercicio7

use ex7Subs

real*4, dimension(1:2) :: point1=(/0.0,0.0/)
real*4, dimension(1:2) :: point2
real*4, dimension(1:3) :: export
real*4 :: angle,x,y,pi=3.14159265359
integer :: i,j
logical :: file_exists

inquire(file="exercicio7.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="exercicio7.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="exercicio7.txt",form="formatted",status="new",
        action="write")
end if

do i=1,10
do j=1,10
x=sin(pi/i)
y=cos(pi/j)
point2=(/x,y/)
call rotation(point1,point2,angle)
export(1)=x
```

```
export(2)=y
export(3)=angle
write(1,*) export(:)
end do
end do

end program exercicio7
```

A.1.8 Exercício 8

Listing A.16 – program: exercicio8

```
module ex8Subs
contains

subroutine createSimMatrix(n)

logical :: file_exists
integer :: n,i,j
real*4, dimension(:,:), allocatable :: myMatrix
real*4, dimension(:), allocatable :: myArray

allocate(myMatrix(n,n))
allocate(myArray(n))

inquire(file="matrizSimetrica.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="matrizSimetrica.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="matrizSimetrica.txt",form="formatted",
        status="new",action="write")
end if

do i=1,n
myMatrix(i,i)=1
do j=i,n-1
myMatrix(i,j+1)=myMatrix(i,j)+1.0
```

```
myMatrix(j+1,i)=myMatrix(i,j+1)
end do
end do

write(1,*) n
do i=1,n
do j=1,n
myArray(j)=myMatrix(i,j)
end do
write(1,*) myArray(:)
end do
close(1)

end subroutine createSimMatrix

subroutine readSimMatrix(myMatrix)

real*4, dimension(:,:), allocatable :: myMatrix
integer :: n

open(unit=1,file="matrizSimetrica.txt",form="formatted",
        status="old",action="read")
read(1,*) n
allocate(myMatrix(n,n))

do i=1,n
read(1,*) (myMatrix(i,j),j=1,n)
end do
close(1)

end subroutine readSimMatrix

subroutine storeSimMatrix(myMatrix,myArray)

real*4, dimension(:,:), allocatable :: myMatrix
real*4, dimension(:), allocatable :: myArray
integer :: n,m,i,j
```

```
logical :: file_exists

inquire(file="vetormatrizSimetrica.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="vetormatrizSimetrica.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="vetormatrizSimetrica.txt",form="formatted",
        status="new",action="write")
end if

n=size(myMatrix)**0.5
m=(n*(n+1))*0.5
allocate(myArray(m))

do i=1,n
do j=i,n
m=(i-1)*n-(i-1)*i*0.5+j
myArray(m)=myMatrix(i,j)
end do
end do

m=(n*(n+1))*0.5
do i=1,m
write(1,*) myArray(i)
end do
close(1)

end subroutine storeSimMatrix

subroutine printTriSupMatrix(myArray)

real*4, dimension(:,:), allocatable :: myMatrix
real*4, dimension(:), allocatable :: myArray
integer :: n,m,i,j
logical :: file_exists
real*4, dimension(:), allocatable :: export
```

```
m=size(myArray)
n=0.5*(1+8*m)**0.5-0.5
allocate(export(n))

inquire(file="matrizTriInferior.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="matrizTriInferior.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="matrizTriInferior.txt",form="formatted",
        status="new",action="write")
end if

do j=1,n
do i=1,j
m=(i-1)*n-(i-1)*i*0.5+j
write(1,"(3f8.3)",advance='no') myArray(m)
write(1,"(A)",advance='no') char(9)
end do
write(1,*) ''
end do
close(1)

end subroutine printTriSupMatrix

subroutine multMatrixArray(A,X,F)

real*4, dimension(:), allocatable :: A,X,F
integer :: i,j,n,m

n=size(X)
F=0.0

do i=1,n
do j=1,n
m=(i-1)*n-(i-1)*i*0.5+j
```

```
F(i)=F(i)+A(m)*X(j)
end do
end do

end subroutine multMatrixArray

end module ex8Subs

program exercicio8

use ex8Subs

integer :: N=8,i
real*4, dimension(:,:), allocatable :: simMatrix
real*4, dimension(:), allocatable :: stgArray,X,F
logical :: file_exists

allocate(X(N))
allocate(F(N))

inquire(file="x.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="x.txt",form="formatted",status="replace",
        action="write")
else
open(unit=1,file="x.txt",form="formatted",status="new",
        action="write")
end if
do i=1,N
X(i)=exp(i*0.5)
write(1,*) X(i)
end do
close(1)

call createSimMatrix(N)
call readSimMatrix(simMatrix)
call storeSimMatrix(simMatrix,stgArray)
```

```
call printTriSupMatrix(stgArray)
call multMatrixArray(stgArray,X,F)

inquire(file="f.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="f.txt",form="formatted",status="replace",
        action="write")
else
open(unit=1,file="f.txt",form="formatted",status="new",
        action="write")
end if
do i=1,N
write(1,*) F(i)
end do
close(1)

end program exercicio8
```

A.1.9 Exercício 9

Listing A.17 – program: exercicio9

```
module ex9Subs
contains

subroutine readNodesCoordinates(myFile,coord)

character(len=*):: myFile
real*4, dimension(:,:), allocatable :: coord
integer :: NNOS,i,j
real*4, dimension(1:4) :: vImport

open(unit=1,file=myFile,form="formatted",status="old",
        action="read")
read(1,*) NNOS
allocate(coord(NNOS,3))
do i=1,NNOS
read(1,*) vImport(:)
```

```
coord(i,1)=vImport(2)
coord(i,2)=vImport(3)
coord(i,3)=vImport(4)
end do
close(1)

end subroutine readNodesCoordinates

subroutine readElementConnectivity(myFile,iconec)

character(len=*):: myFile
real*4, dimension(:,:), allocatable :: iconec
integer :: NELEM,i,j
real*4, dimension(1:5) :: vImport

open(unit=1,file=myFile,form="formatted",status="old",
        action="read")
read(1,*) NELEM
allocate(iconec(NELEM,5))
do i=1,NELEM
read(1,*) (iconec(i,j),j=1,5)
end do
close(1)

end subroutine readElementConnectivity

end module ex9Subs

program exercicio9

use ex9Subs

logical :: file_exists
real*4, dimension(1:5) :: export
integer :: NNOS=9,NELEM=4
real*4 :: pi=3.14159265359
real*4, dimension(:,:), allocatable :: coord
```

```
real*4, dimension(:,:), allocatable :: iconec

inquire(file="nodesCoordinates.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="nodesCoordinates.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="nodesCoordinates.txt",form="formatted",
        status="new",action="write")
end if
write(1,*) NNOS
do i=1,NNOS
export(1)=i
export(2)=sin(i*pi*0.5)
export(3)=cos(i*pi)
export(4)=tan(i*pi*0.25)
write(1,*) export(1:4)
end do
close(1)

inquire(file="elemConnectivity.txt",exist=file_exists)
if(file_exists.eqv..true.) then
open(unit=1,file="elemConnectivity.txt",form="formatted",
        status="replace",action="write")
else
open(unit=1,file="elemConnectivity.txt",form="formatted",
        status="new",action="write")
end if
write(1,*) NELEM
do i=1,NELEM
export(1)=i
export(2)=i+1
export(3)=i
export(4)=i+3
export(5)=i+4
write(1,*) export(:)
end do
```

```
close(1)

call readNodesCoordinates("nodesCoordinates.txt",coord)
call readElementConnectivity("elemConnectivity.txt",iconec)

open(unit=1,file="nodesCoordinates.txt",form="formatted",
        status="replace",action="write")
do i=1,NNOS
export(1)=coord(i,1)
export(2)=coord(i,2)
export(3)=coord(i,3)
write(1,*) export(1:3)
end do
close(1)

open(unit=1,file="elemConnectivity.txt",form="formatted",
        status="replace",action="write")
do i=1,NELEM
write(1,*) (iconec(i,j),j=1,5)
end do
close(1)

end program exercicio9
```

A.2 Trabalho 4

Listing A.18 – dadosEL2Dbenchmark1.txt

```
9 3 ! NNOE NPX, NNOE = numero de nos do elemento; NPX = numero
! de pontos Gauss-Lobatto em cada direcao do elemento p/ calculo
! de tensoes
1 0.0 0.0 ! NoI XI YI, dados de coordenadas nodais (em mm)
2 1000.0 0.0
3 2000.0 0.0
4 3000.0 0.0
5 4000.0 0.0
6 5000.0 0.0
```

```
7 6000.0 0.0
8 7000.0 0.0
9 8000.0 0.0
10 0.0 2.5
11 1000.0 2.5
12 2000.0 2.5
13 3000.0 2.5
14 4000.0 2.5
15 5000.0 2.5
16 6000.0 2.5
17 7000.0 2.5
18 8000.0 2.5
19 0.0 5.0
20 1000.0 5.0
21 2000.0 5.0
22 3000.0 5.0
23 4000.0 5.0
24 5000.0 5.0
25 6000.0 5.0
26 7000.0 5.0
27 8000.0 5.0
-1 0 0 ! sinaliza o final do bloco de dados de coordenadas
! nodais
1 1 3 21 19 2 12 20 10 11 ! Elem No1 No2 No3, conectividade dos
! elementos
2 3 5 23 21 4 14 22 12 13
3 5 7 25 23 6 16 24 14 15
4 7 9 27 25 8 18 26 16 17
-1 0 0 0 0 0 0 0 0 0 ! sinaliza o final do bloco de dados de
! conectividade
200D3 0.10 5.0 ! E Poison Espessura, propriedades do material,
! [E]=N/mm2, [Espessura]=mm
1 1 0.0 ! No Direcao Valor, cargas concentradas; Direcao = 1(X)
! ou 2(Y), [Valor]=N
-1 0 0 ! sinaliza o final do bloco de dados de carga concentrada
1 1 0.0 0 ! No Direcao Valor NoJ, deslocamentos prescritos;
! Direcao = 1(X) ou 2(Y), [Valor]=mm, NoJ usado apenas no tipo
```

```
! de condicao Ui=Valor*Uj
1 2 0.0 0
10 1 0.0 0
10 2 0.0 0
19 1 0.0 0
19 2 0.0 0
-1 0 0.0 0 ! sinaliza o final do bloco de dados de CC essenciais
1 -7.84D-5 1 ! Elem Valor Direcao, forcas de corpo; Direcao =
! 1(X) ou 2(Y), [Valor]=N/mm3
2 -7.84D-5 1
3 -7.84D-5 1
4 -7.84D-5 1
-1 0 0 ! sinaliza o final do bloco de forcas de corpo
4 1 1 0 0.0 0.0 ! Elem Lado direcao ixy Valor1 Valor2, tensoes
! prescritas; Direcao = 1(X) ou 2(Y), ixy=0(X,y),
! [Valor1]=[Valor2]=N/m2
-1 0 0 0 0.0 0.0 ! sinaliza o final do bloco de dados de CC
! naturais
```

Listing A.19 – plot1.py

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

parentDirectory=pathlib.Path(__file__).resolve().parents[0]

nu=0.1
E=200e3
b=-7.84e-5
L=8000
x=np.arange(0.0,8000.0, 1.0)
def u(x):
        return b*(1+nu)*(1-2*nu)/(E*(1-nu))*(L-x/2)*x
def sig(x):
        return b*(L-x)

fig=plt.figure()
```

```
fileName=str(parentDirectory)+"/exportU.txt"
results=np.loadtxt(fname=fileName)
plotName="deslocamentoHorizontal.png"
plt.plot(results[0:9,0],results[0:9,1],'ok',label='MEF')
plt.plot(x,u(x),'--k',label='Exata')
plt.ylabel('Deslocamento␣horizontal␣(mm)')
plt.xlabel('Comprimento␣(mm)')
plt.grid(which='major',axis='both')
plt.legend()
plt.savefig(plotName)

fig=plt.figure()
fileName=str(parentDirectory)+"/exportSigmaXMediaNodal.txt"
results=np.loadtxt(fname=fileName)
plotName="tensaoHorizontal.png"
plt.plot(results[0:9,0],results[0:9,1],'ok',label='MEF')
plt.plot(x,sig(x),'--k',label='Exata')
plt.ylabel('Tens o␣normal␣horizontal␣(MPa)')
plt.xlabel('Comprimento␣(mm)')
plt.grid(which='major',axis='both')
plt.legend()
plt.savefig(plotName)
```

Listing A.20 – dadosEL2Dbenchmark2.txt

```
9 3 ! NNOE NPX, NNOE = numero de nos do elemento; NPX = numero
! de pontos Gauss-Lobatto em cada direcao do elemento p/ calculo
! de tensoes
1 0.0 0.0 ! NoI XI YI, dados de coordenadas nodais (em mm)
2 2.5 0.0
3 5.0 0.0
4 0.0 1000.0
5 2.5 1000.0
6 5.0 1000.0
7 0.0 2000.0
8 2.5 2000.0
9 5.0 2000.0
10 0.0 3000.0
```

```
11 2.5 3000.0
12 5.0 3000.0
13 0.0 4000.0
14 2.5 4000.0
15 5.0 4000.0
16 0.0 5000.0
17 2.5 5000.0
18 5.0 5000.0
19 0.0 6000.0
20 2.5 6000.0
21 5.0 6000.0
22 0.0 7000.0
23 2.5 7000.0
24 5.0 7000.0
25 0.0 8000.0
26 2.5 8000.0
27 5.0 8000.0
-1 0 0 ! sinaliza o final do bloco de dados de coordenadas
! nodais
1 1 3 9 7 2 6 8 4 5 ! Elem No1 No2 No3, conectividade dos
! elementos
2 7 9 15 13 8 12 14 10 11
3 13 15 21 19 14 18 20 16 17
4 19 21 27 25 20 24 26 22 23
-1 0 0 0 0 0 0 0 0 0 ! sinaliza o final do bloco de dados de
! conectividade
200D3 0.10 5.0 ! E Poison Espessura, propriedades do material,
! [E]=N/mm2, [Espessura]=mm
1 1 0.0 ! No Direcao Valor, cargas concentradas; Direcao = 1(X)
! ou 2(Y), [Valor]=N
-1 0 0 ! sinaliza o final do bloco de dados de carga concentrada
1 1 0.0 0 ! No Direcao Valor NoJ, deslocamentos prescritos;
! Direcao = 1(X) ou 2(Y), [Valor]=mm, NoJ usado apenas no tipo
! de condicao Ui=Valor*Uj
1 2 0.0 0
2 1 0.0 0
2 2 0.0 0
```

```
3 1 0.0 0
3 2 0.0 0
-1 0 0.0 0 ! sinaliza o final do bloco de dados de CC essenciais
1 -7.84D-5 2 ! Elem Valor Direcao, forcas de corpo; Direcao =
! 1(X) ou 2(Y), [Valor]=N/mm3
2 -7.84D-5 2
3 -7.84D-5 2
4 -7.84D-5 2
-1 0 0 ! sinaliza o final do bloco de forcas de corpo
4 1 2 0 0.0 0.0 ! Elem Lado direcao ixy Valor1 Valor2, tensoes
! prescritas; Direcao = 1(X) ou 2(Y), ixy=0(X,y),
! [Valor1]=[Valor2]=N/m2
-1 0 0 0 0.0 0.0 ! sinaliza o final do bloco de dados de CC
! naturais
```

Listing A.21 – plot2.py

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

parentDirectory=pathlib.Path(__file__).resolve().parents[0]

nu=0.1
E=200e3
b=-7.84e-5
L=8000
y=np.arange(0.0,8000.0, 1.0)
def v(y):
        return b*(1+nu)*(1-2*nu)/(E*(1-nu))*(L-y/2)*y
def sig(y):
        return b*(L-y)

fig=plt.figure()
fileName=str(parentDirectory)+"/exportV.txt"
results=np.loadtxt(fname=fileName)
plotName="deslocamentoVertical.png"
plt.plot(results[:,0],results[:,1],'ok',label='MEF')
```

```
plt.plot(y,v(y),'--k',label='Exata')
plt.ylabel('Deslocamento␣vertical␣(mm)')
plt.xlabel('Altura␣(mm)')
plt.grid(which='major',axis='both')
plt.legend()
plt.savefig(plotName)

fig=plt.figure()
fileName=str(parentDirectory)+"/exportSigmaYMediaNodal.txt"
results=np.loadtxt(fname=fileName)
plotName="tensaoVertical.png"
plt.plot(results[:,0],results[:,1],'ok',label='MEF')
plt.plot(y,sig(y),'--k',label='Exata')
plt.ylabel('Tens o␣normal␣vertical␣(MPa)')
plt.xlabel('Altura␣(mm)')
plt.grid(which='major',axis='both')
plt.legend()
plt.savefig(plotName)
```