UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Mecânica dos fluidos e transmissão de calor computacional
Prof. Gustavo dos Anjos | 04/06/2020

Aluno: Felipe Rodrigues de Mello Alves

DRE: 113278558

Graduando em Engenharia Mecânica | CT/UFRJ

Malhas computacionais

Este trabalho tem por objetivo descrever o procedimento de construção de malhas computacionais com 1 e 2 dimensões em linguagem Python.

As malhas consistem na discretização do plano físico em diversos segmentos, sendo composta portanto de pequenos elementos que são delimitados por pontos. Essa discretização é ideal para a solução de diversos problemas da ciência e engenharia, uma vez que divide um sólido ou fluido em diversos segmentos menores, facilitando o cálculo das variáveis envolvidas. Os elementos de malhas bidimensionais podem ter diversos formatos, entretanto nesse trabalho apresento apenas as malhas com elementos de formatos quadrangulares e triângulares.

O texto apresenta trechos fundamentais do código. Para qualquer entendimento mais profundo, os códigos inteiros se encontram nos Anexos 1 e 2.

1. **Entradas do usuário:**

   A malha, portanto, é composta pela interligação de pontos do seu objeto de estudo, formando assim os elementos. O comprimento do seu objeto, o número de lados dos seus elementos e o número total de pontos utilizados são tratados como uma entrada do usuário no código. As variáveis de entrada representam as seguintes grandezas:

   (a) $Lx-$ Comprimento do seu objeto no eixo x

   (b) $Ly-$ Comprimento do seu objeto no eixo y

   (c) nx - Número de pontos em x

   (d) ny - Número de pontos em y

   (e) *tipo* - Representa o número de lados do elemento (3 ou 4)

2. **Matrizes de coordenadas:**

   Para ser possível gerar a malha, é necessária a construção das matrizes de coordenadas de cada ponto que delimita a malha. Nesse trabalho os elementos e pontos são numerados a partir do 0 - ou seja o primeiro elemento é representado pelo algarismo zero. Cada linha dessas matrizes representa um ponto.

   (a) Coordenadas unidimensionais

   Consiste nas coordenadas de cada ponto de uma reta, tendo como valor final o comprimento Lx do seu objeto. Por exemplo uma matriz de coordenadas unidimensional de um objeto com comprimento unitário dividido em 5 pontos fica da forma:

$$\overrightarrow{X} = \begin{bmatrix} 0.0 \\ 0.25 \\ 0.5 \\ 0.75 \\ 1.0 \end{bmatrix}$$

   Portanto a construção da matriz unidimensional segue sempre este padrão. Segue abaixo o código utilizado para a construção dessa matriz:

```
X = np.zeros((nx, 1), dtype='float')
for i in range(0, nx):
    X[i] = (Lx / (nx - 1)) * i
```

(b) Coordenadas bidomensionais

Descreve as coordenadas de acordo com a sua numeração dos pontos. Por exemplo, as matrizes de coordenadas da malha da Figura 1 são:

$$\overrightarrow{X} = \begin{bmatrix} 0.0 \\ 0.5 \\ 1.0 \\ 0.0 \\ 0.5 \\ 1.0 \\ 0.0 \\ 0.5 \\ 1.0 \end{bmatrix} ; \overrightarrow{Y} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.5 \\ 0.5 \\ 0.5 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

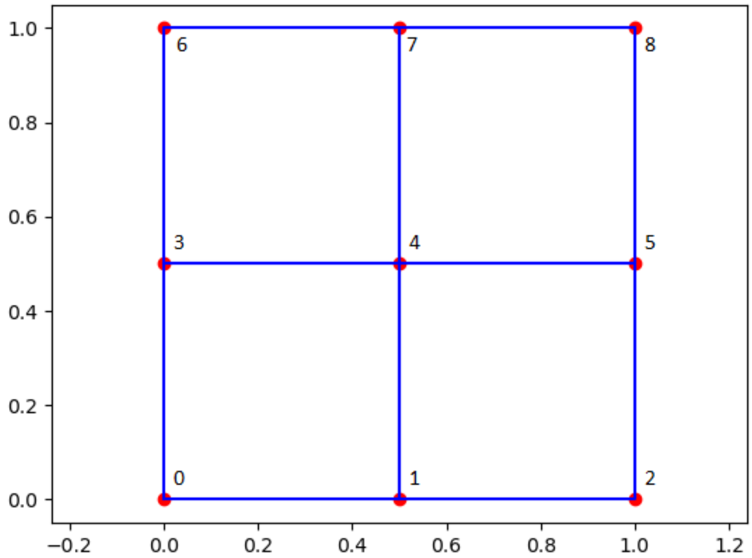

Figura 1: Numeração de pontos de uma malha bidimensional com 4 elementos quadrados.

Para a construção dessas matrizes foi utilizada a seguinte lógica:

```
for i in range(0, nx * ny):
    X[i] = Lx * i / (nx - 1)
    if i > (nx - 1):
        X[i] = X[i - nx]

for i in range(0, nx * ny):
    Y[i] = 0.0
    if i >= (nx):
        Y[i] = Y[i - nx] + dy
```

Se for de interesse gerar uma malha irregular, pode-se gerar uma malha com uma perturbação senoidal do tipo:

$$y(x) = A \cdot \sin(\frac{2\Pi \cdot x}{\lambda} - \Phi)$$

através do código:

```python
for i in range(0, npoints):
    Y[i] = 0.0
    #  Malha com perturbação:
    if i >= nx:
        A = 0.1  #  "A = 0.0" é a malha sem perturbacao
        fi = 2 * np.pi / 4.0
        Y[i] = (Y[i - nx] + dy) + A * np.sin((2 * np.pi / 5.0) * X[i] - fi)
```

3. **Matriz de conectividade:**

   Para organizar as delimitações de cada elemento (quais pontos compõem cada elemento) é construída uma matriz de conectividade denominada "IEN". Cada linha dessas matrizes representa um elemento.

   (a) IEN unidimensional:

   Um exemplo de matriz "IEN" unidimensional com n pontos é demonstrado a seguir:

$$IEN = \begin{bmatrix} 0 & 1 \\ 1 & 2 \\ 2 & 3 \\ \dots & \dots \\ n-2 & n-1 \end{bmatrix}$$

   (b) IEN bidimensional

   A matriz "IEN" bidimensional de elementos quadrangulares da Figura 1 é demonstrada a seguir:

$$IEN = \begin{bmatrix} 0 & 1 & 4 & 3 \\ 1 & 2 & 5 & 4 \\ 3 & 4 & 7 & 6 \\ 4 & 5 & 8 & 7 \end{bmatrix}$$

   Portanto a construção dessa matriz bidimensional segue um padrão anti-horário em cada elemento.

   O algoritmo de construção de matriz IEN bidimensional são particulares para cada formato de elemento.

   i. Elementos quadrangulares

   Foi criada uma variável auxiliar s, além da variável iterativa e.

```python
if tipo == 4:
    s = -1
    for e in range(0, ne):
        if e % (nx - 1) == 0:
            s = s + 1
        IEN[e] = [e + s, e + 1 + s, e + nx + 1 + s, e + nx + s]
```

   ii. Elementos triangulares

   Foi criada uma variável auxiliar i, além das variáveis iterativas a e b.

```python
if tipo == 3:
    i = 1
    for a in range(ny - 1):
        for b in range(nx - 1):
            IEN[i] = [nx * a + b, nx + 1 + b + nx * a, nx * a + b + 1]
            IEN[i - 1] = [nx * a + b, nx + nx * a + b, nx + nx * a + b + 1]
            i += 2
```

4. **Matriz de contorno:**

   Para a resolução de problemas, também é importante termos conhecimento de quais nós da malha pertencem ao contorno e quais pertencem ao interior da malha. Com isso temos a construção das matrizes Bound e Inner, feita através do seguinte código:

```
# 6. Inner e Bound
P = np.ones((npoints), dtype='int')
for i in range(npoints):
    P[i] = i

bound = list(P)
inner = np.zeros((nx-2)*(ny-2), dtype='int')
s = 0
for j in range (1,ny-1):
    for i in range(npoints):
        if j * nx < i < (((j+1) * nx) - 1):
            bound.remove(i)
            inner[s] = i
            s = s + 1
inner = list(inner)
```

5. **Malhas geradas:**

   Serão apresentadas nas imagens abaixo alguns exemplos de malhas diferentes geradas pelo código.

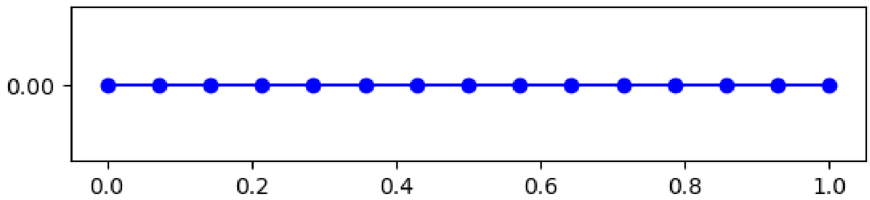

Figura 2: Malha unidimensional com 15 pontos e 14 elementos

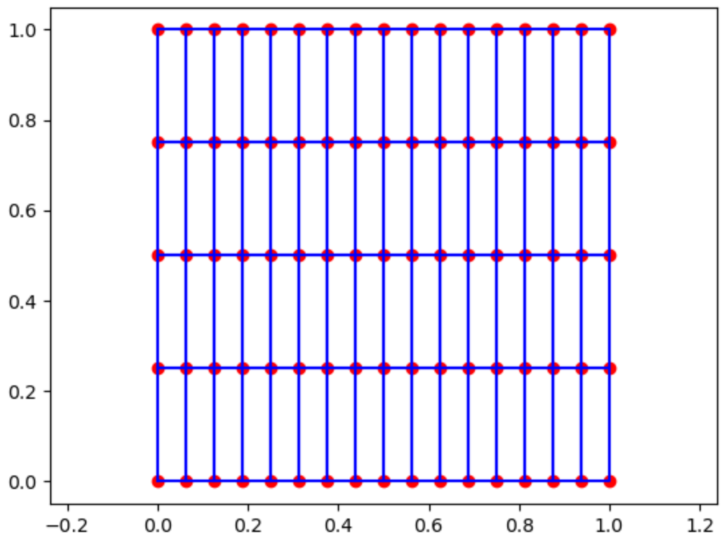

Figura 3: Malha bidimensional quadrangular com 17 pontos em x e 5 pontos em y

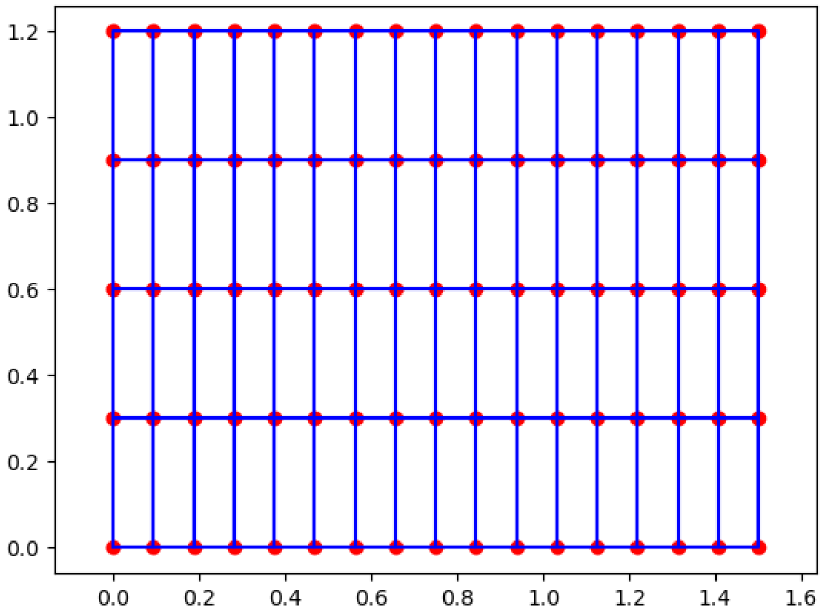

Figura 4: Mesma malha anterior com comprimentos laterais variados

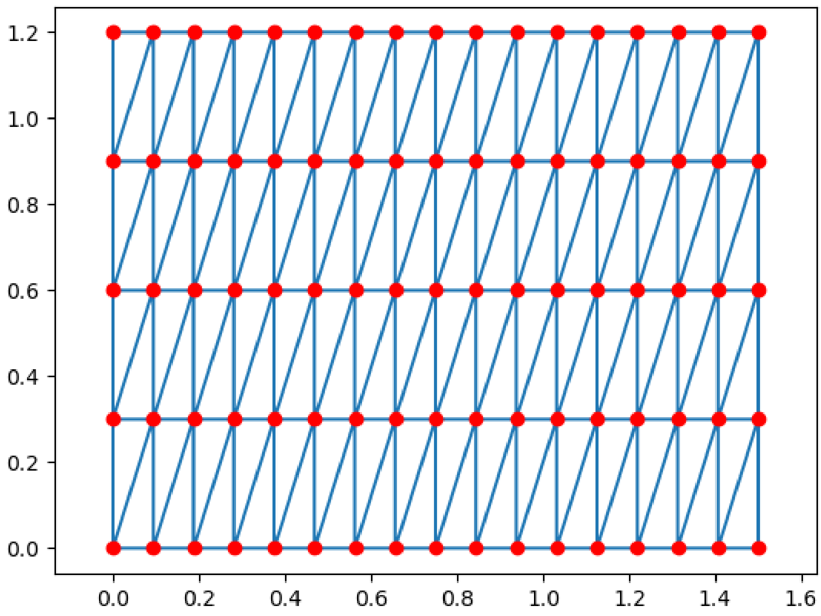

Figura 5: Mesma malha anterior com elementos triangulares

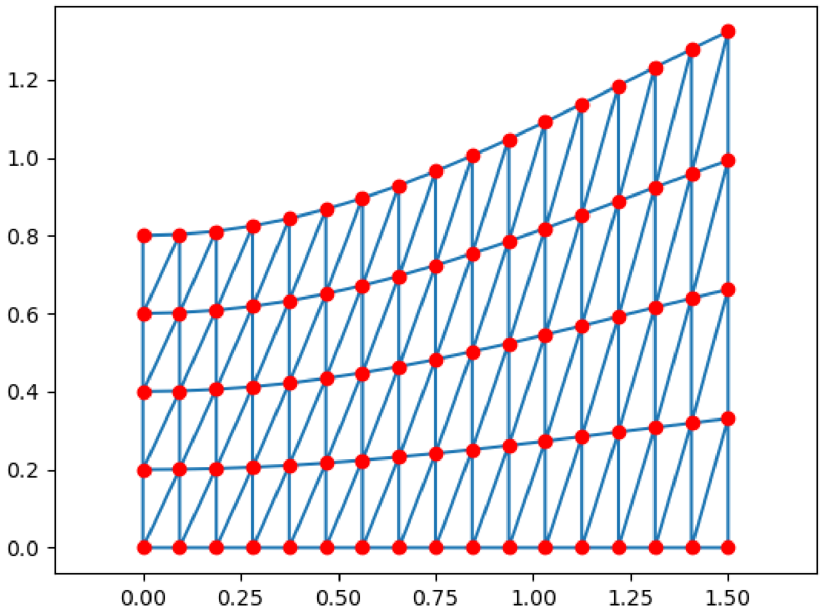

Figura 6: Mesma malha anterior com perturbação

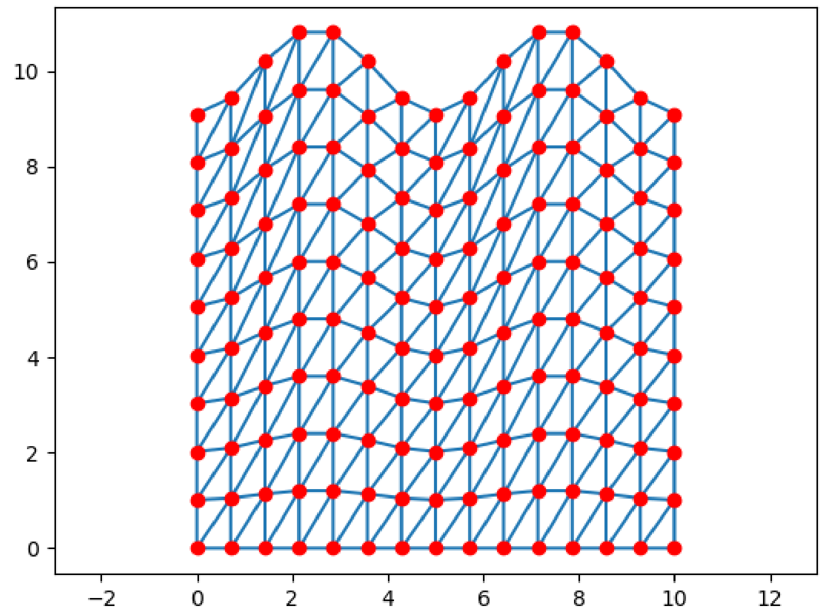

Figura 7: Malha com perturbação

Anexo 1 – Código da malha unidimensional

Esse código foi usado para os casos de geração de malha unidimensional, e os valores de entrada podem ser alterados.

<u>Código Python:</u>

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

# 1. Entrada do usuario
L = 1
nx = 15   # Numero de pontos
dx = L / (nx - 1)
ne = nx - 1

X = np.zeros((nx, 1), dtype='float')
for i in range(0, nx):
   X[i] = dx * i

IEN = np.zeros((ne, 2), dtype='int')
for e in range(0, ne):
   IEN[e, 0] = e
   IEN[e, 1] = e + 1

print(IEN)
Y = np.zeros((nx, 1), dtype='float')
plt.plot(X, Y, 'bo-')  # o "-" é para traçar a reta
plt.show()
```

Anexo 2 – Código da malha bidimensional

Esse código foi usado para os casos de geração de malha bidimensional, e os valores de entrada podem ser alterados.

<u>Código Python:</u>

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

# 1. Entrada do usuario
Lx = 1.2
Ly = 1.5
nx = 17  # Numero de pontos no eixo x
ny = 5  # Numero de pontos no eixo y
tipo = 4  # Numero de lados do elemento (3 ou 4)

# 2. Variáveis
if tipo == 4:
    f = 1
if tipo == 3:
    f = 2

ne = f * (nx - 1) * (ny - 1)  # Número de elementos
npoints = nx * ny  # Número de pontos
dx = Lx / (nx - 1)  # Comprimento de cada elemento em x
dy = Ly / (ny - 1)  # Comprimento de cada elemento em y

# 3. Criação das Listas
X = np.zeros((npoints), dtype='float')  # Matriz de coordenada x de cada ponto da malha
Y = np.zeros((npoints), dtype='float')  # Matriz de coordenada y de cada ponto da malha
IEN = np.zeros((ne, tipo), dtype='int')

# 4. Preenchimento das listas X e Y
for i in range(0, npoints):
    X[i] = Lx * i / (nx - 1)
    if i > (nx - 1):
        X[i] = X[i - nx]

for i in range(0, npoints):
    Y[i] = 0.0
    #  Malha com perturbação:
    if i >= nx:
        A = 0.1   #  "A = 0.0" é a malha sem perturbacao
        fi = 2 * np.pi / 4.0
        Y[i] = (Y[i - nx] + dy) + A * np.sin((2 * np.pi / 5.0) * X[i] - fi)

# 5. Matriz IEN
# 5.1. Malha de quadriláteros
if tipo == 4:
```

```python
    s = -1
    for e in range(0, ne):
        if e % (nx - 1) == 0:
            s = s + 1
        IEN[e] = [e + s, e + 1 + s, e + nx + 1 + s, e + nx + s]

# 5.2. Malha de triângulos
if tipo == 3:
    i = 1
    for a in range(ny - 1):
        for b in range(nx - 1):
            IEN[i] = [nx * a + b, nx + 1 + b + nx * a, nx * a + b + 1]
            IEN[i - 1] = [nx * a + b, nx + nx * a + b, nx + nx * a + b + 1]
            i += 2

# 6. Inner e Bound
P = np.ones((npoints), dtype='int')
for i in range(npoints):
    P[i] = i

bound = list(P)
inner = np.zeros((nx-2)*(ny-2), dtype='int')
s = 0
for j in range (1,ny-1):
    for i in range(npoints):
        if j * nx < i < (((j+1) * nx) - 1):
            bound.remove(i)
            inner[s] = i
            s = s + 1
inner = list(inner)

# 7. Plot
plt.plot(X, Y, 'ro')
if tipo == 3:
    plt.triplot(X, Y, IEN)

if tipo == 4: #OBS: Tambem funciona para tipo == 3
# def PlotMesh(tipo):
    for i in range(0, len(IEN)):
        cx = np.zeros((tipo + 1), dtype='float')
        cy = np.zeros((tipo + 1), dtype='float')
        for l in range(0, tipo):
            cx[l] = X[IEN[i][l]]
            cy[l] = Y[IEN[i][l]]
        cx[tipo] = cx[0]
        cy[tipo] = cy[0]
        plt.plot(cx, cy, 'b-')

plt.axes().set_aspect('equal', 'datalim')
plt.show()
# plt.savefig("2dmesh")
```