Algoritmo de geração de malhas estruturadas.

Gabriel Felipe de Oliveira Antão
Departamento de Engenharia Mecânica, UFRJ.

Resumo

Este relatório apresenta a implementação de um algoritmo para geração de malhas estruturadas simples com elementos triangulares. O objetivo é perturbar, com uma função senoidal, a posição dos nós regularmente dispostos ao longo de eixo x e y de uma malha bidimensional, gerar sua matriz de conectividade e então a visualização do resultado. O código foi implementado em Python com auxílio das bibliotecas NumPy e Matplotlib.

I. Introdução

Uma das etapas do Método de Elementos Finitos é a geração da malha que representa a discretização do domínio físico do problema em questão. Softwares especializados implementam algoritmos para geração automática dessas malhas a partir de um modelo previamente criado em um software de CAD. Este trabalho apresenta a implementação de um algoritmo para geração de uma malha bidimensional simples com nós regularmente espaçados formando um domínio retangular.

II. Descrição do algoritmo

A partir das dimensões L_x e L_y e com número de nós n_x e n_y, nas direções do eixo x e do eixo y, respectivamente, é possível gerar todos os nós de uma malha retangular. A proposta do algoritmo implementado é perturbar a posição dos nós da fronteira da malha com uma *função de perturbação* arbitrária e acomodar os demais nós de acordo com uma regra simples aqui denominada de *regra de acomodação*. A regra de acomodação dos nós é um polinômio de primeiro grau que calcula a nova posição dos pontos na forma

$$y_n(x) = \frac{y}{L_y} f(x) + y$$

em que y é a posição atual do nó na direção y, $f(x)$ o valor da função de perturbação dos nós calculada no ponto x de interesse e y_n é a nova posição dos nós em y. A posição dos nós em x se preserva, uma vez que se aplica a perturbação na direção do eixo y. O termo y/L_y gera um fator de escala que ajusta a disposição na direção do eixo y do formato da função de perturbação até o deslocamento nulo dos pontos em que $y = 0$. A regra de acomodação utilizada reajusta os nós, portanto, alterando-os linearmente a curvatura da função de perturbação até um valor de uma função constante $y = 0$. A função que calcula a nova posição dos nós retorna cada um dos pontos em forma de uma matriz $n_t \times 2$ em que $n_t = n_x \times n_y$; as colunas representam as coordenadas x e y e as linhas cada um dos nós.

Uma função foi utilizada para para geração dos elementos triangulares numerando os nós sequencialmente, iniciando de 0 até n_t, de baixo para cima, da esquerda para direita. A função gera uma matriz da forma $n_e \times 3$ em que os

números do nós estão dispostos nas colunas e os elementos nas linhas; $n_e = 2 * (n_x - 1) * (n_y - 1)$. Com a informação da posição dos nós descritos anteriormente e da matriz de conectividade, é possível gerar a nova malha de interesse.

III. Resultados

A função escolhida para exemplificar o funcionamento do código é

$$f(x) = A \sin\left(\frac{2\pi}{\lambda}x - \phi\right)$$

e uma malha com nós igualmente espaçados foi utilizada para testar o código. Os parâmetros utilizados para geração da malha estão resumidos na Tabela 1.

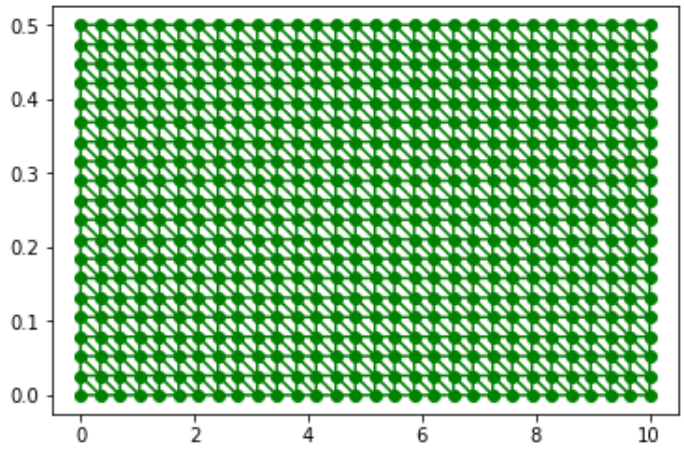

Figura 1: *Malha original.*

O que se observa é que os nós seguem o formato da função de perturbação tendendo ao valor da função constante $y = 0$ quando a função se aproxima do eixo x, conforme esperado. Outras funções de perturbações geram um comportamento semelhante. Este exemplo usa pontos igualmente espaçados no eixo x, mas isso não é um requisito para o funcionamento do código.

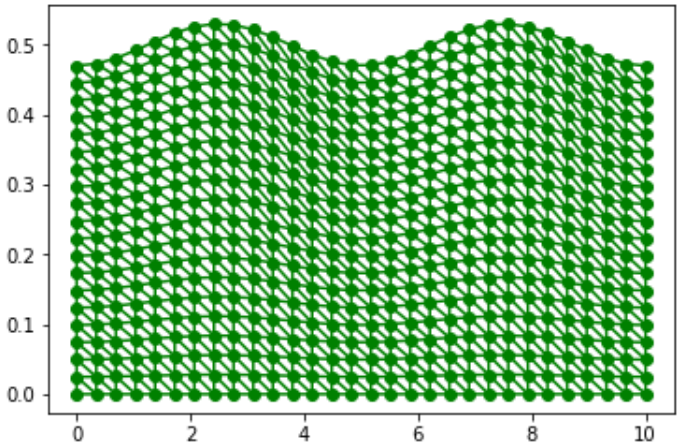

Figura 2: *Malha alterada pela função de perturbação.*

IV. Conclusão

De um algoritmo simples é possível gerar os nós de uma malha estruturada e perturbar a posição dos pontos para obter uma nova malha, desta vez com os pontos acomodados de acordo com uma regra de interesse. O mesmo algoritmo pode ser utilizado para pontos que não estejam necessariamente igualmente espaçados e para funções arbitrárias que perturbem a posição dos pontos da malha. O código apresentado no Apêndice poderá ser reutilizado para geração de malhas bidimensionais, desde que estas atendam o requisito de serem estruturadas, regulares e uniformes.

Referências

[1] Anjos, G. R. *Computação Científica para Engenheiros*. 2019.

[2] Rao, S. S. *The Finite Element Method in Engineering*. 5 ed. Burlington: Elsevier, 2011.

Tabela 1: *Parâmetros utilizados para exemplo.*

A	ϕ	λ	k	L_x	L_y	n_x	n_y
0.03	$2\pi/4$	$10/2$	$2\pi/\lambda$	10.0	0.5	30	20

Apêndice

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Universidade Federal do Rio de Janeiro
Departamento de Engenharia Mecanica

Mecanica dos fluidos e transferencia de calor computacional (EEK-597)

Solucao da tarefa proposta em:
Gustavo Anjos. 'Computacao cientifica para engenheiros'. Cap 3.3, pag 40.

autor: Gabriel Antao [gabrielantao@poli.ufrj.br]

1) Implementa algoritmo para gerar pontos de uma malha com uma perturbacao em um
extremidade. Os valores da coordenada y dos nos sao recalculados com um fator
de escala linear.
2) Implementa geracao de matriz de conectividade.
"""

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


def apply_disturbance(x, y):
    """ Gera o vetor com os pontos da malha 'amortecida'. """
    all_nodes = np.empty((n_t, 2))
    for i, node_coord in enumerate(itertools.product(x, y)):
        all_nodes[i] = [node_coord[0],
                        node_coord[1]/L_y * func(node_coord[0]) + node_coord[1]]
    return all_nodes

def triangulate(n_x, n_y):
    """ Retorna a matriz de conectividade. """
    all_nodes = []
```

```
    for f in range(0, n_x * n_y - n_y, n_y):
        for y in range(n_y - 1):
            all_nodes.append([f + y, f + y + 1, f + y + n_y])
            all_nodes.append([f + y + 1, f + y + n_y + 1, f + y + n_y])
    return np.array(all_nodes)

def plot_curves(x, y):
    """ Visualizacao das curvas 'amortecidas'. """
    default_shape = func(x) #calcula so uma vez
    for y in [node_y/L_y * default_shape + node_y for node_y in y]:
        plt.plot(x, y)

def plot_mesh(nodes, triangulations):
    """ Visualizacao da malha. """
    plt.triplot(nodes[:, 0], nodes[:, 1], triangulations, "go-")
    plt.show()

if __name__ == "__main__":
    L_x = 10.0                          # comprimento em x
    L_y = 0.5                           # comprimento em y
    n_x = 30                            # numero de nos em x
    n_y = 20                            # numero de nos em y
    n_t = n_x * n_y                     # numero total de nos
    n_e = 2 * (n_x - 1) * (n_y - 1)     # numero total de elementos
    A = 0.03
    phi = 2 * np.pi / 4
    lambda_ = 10.0 / 2
    k = 2 * np.pi / lambda_
    func = lambda x: A * np.sin(k * x - phi)
    # criacao dos pontos iniciais
    x_init = np.linspace(0, L_x, n_x)
    y_init = np.linspace(0, L_y, n_y)
    # geracao dos triangulos
    triangles = triangulate(n_x, n_y)
    # malha original
    nodes = np.array([par for par in itertools.product(x_init, y_init)])
    plot_mesh(nodes, triangles)
    # malha perturbada
    nodes = apply_disturbance(x_init, y_init)
    plot_mesh(nodes, triangles)
```