Universidade Federal do Rio de Janeiro

Estudos sobre geração de malha

Modelos 1D e 2D

Manipulação das matrizes de malha utilizando Python

Aluno Matheus Lira Sartor
Professor Gustavo Rabello

Rio de Janeiro, 18 de outubro de 2021

Conteúdo

1 Introdução **1**

2 Classificação das malhas **2**

3 Manipulação da malha **5**

4 Componentes da Malha **5**
4.1 Vetores de Coordenadas 6
4.2 Matriz de Conectividade 6

5 Gerador de Malha Ordenada 2D **8**
5.1 O Código 8
5.1.1 Parâmetros Principais 8
5.1.2 Algoritmo 9
5.2 Qualidade dos elementos da malha 9
5.2.1 Triangulação de Delaunay 9
5.2.2 Critério de Equilateralidade 10

6 Gerador de Malha Aleatório 2D **11**

7 Gerador de Malha com Contorno 2D **13**

8 GMsh **14**

9 Conclusão **17**

10 Referências **18**

A Apêndice - Códigos **19**
A.1 Gerador de Malhas Ordenadas 19
A.2 Gerador de Malhas Aleatórias 22

B Apêndice - Simulações **26**

1 Introdução

Antes de entender a fundamentação e o propósito das malhas, é preciso compreender a importância delas para a engenharia. Para isso, é necessário obter uma visão mercadológica do valor agregado pelas simulações computacionais a um projeto. Com relação a custos [4], foi estimado que o custo de implementação da simulação em um projeto é considerado baixo, visto que foi atribuído entre 1 a 3% do custo total do projeto. Atualmente estima-se que este custo seja até menor. Sobre o custo do uso de softwares de simulação, estipulou uma relação entre o custo e a fase do sistema, conforme a figura abaixo.

Figura 1: Custos de projeto - Fonte: Duarte(2003)[4]

Para realizar as simulações, existem numerosos métodos matemáticos utilizados que podem ser melhores ou piores dependendo da complexidade do problema. Um dos métodos mais utilizados atualmente é o método dos elementos finitos (ou MEF), que consistentemente se utiliza das malhas para gerar soluções para os mais diversos problemas envolvendo equações diferenciais. Por esse motivo as malhas são tão importantes para a engenharia.

O método dos elementos finitos leva em conta o problema da continuidade física, utilizando funções de interpolações, tendo sua convergência aproximada do real, quanto mais elementos forem levados em consideração na análise [1]. Um importante ponto são os nós, que representa o ponto central da figura [2].

A malha então representa a união das subdivisões dos elementos, logo a eficiência da malha dependerá da adaptação do refinamento feito. Esse refinamento depende de operações aritméticas, que são proporcionais ao elemento finito. A malha apresenta vários tipos de modelamento, em especial o triangular, o quadrilátero, hexaedros, entre outros [3].

Nas seções seguintes, serão explicitados os tipos de malhas, a qualidade das malhas e serão respondidas questões sobre escolha e refinamento de malhas.

2 Classificação das malhas

No que se diz respeito a ordenação dos elementos, pode-se construir malhas das seguintes maneiras:

- malha estruturada
- malha não-estruturada

No caso da malha estruturada, a distribuição dos pontos é regular e uniforme, provocando certa homogeneidade entre os elementos. Entretanto, dependendo da geometria a ser analisada, isso pode não ser algo desejado, pois essa uniformidade pode gerar lacunas na malha gerada ou então até mesmo propor uma solução pobre de detalhes para o sistema.

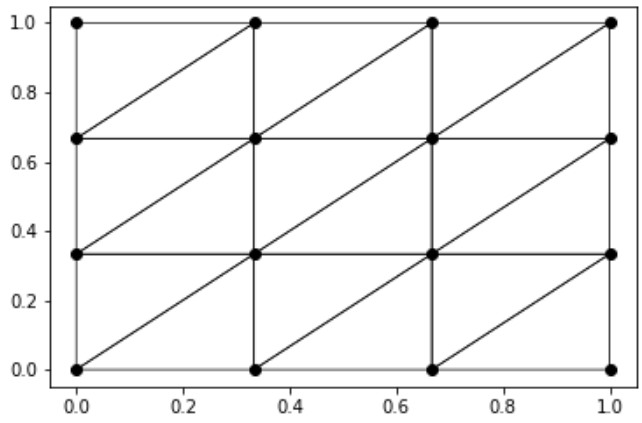

Figura 2: Exemplo de Malha Ordenada

Agora imagine que você queira resolver um problema de análise de tensões e deformações em uma placa com um furo. Sabendo-se que quanto mais elementos houverem presentes em um determinado espaço, maior será a precisão dos resultados da simulação, então fica claro imaginar que será pretendido que haja uma maior concentração de elementos próximos da zona de algum entalhe, visto que essas zonas serão as mais afetadas por algum esforço de

flexão ou torção da peça. Portanto, será necessário um ordenamendo dos elementos que os transforme em uma figura não mais homogênea, tornando desejável então o uso de uma malha não estuturada.

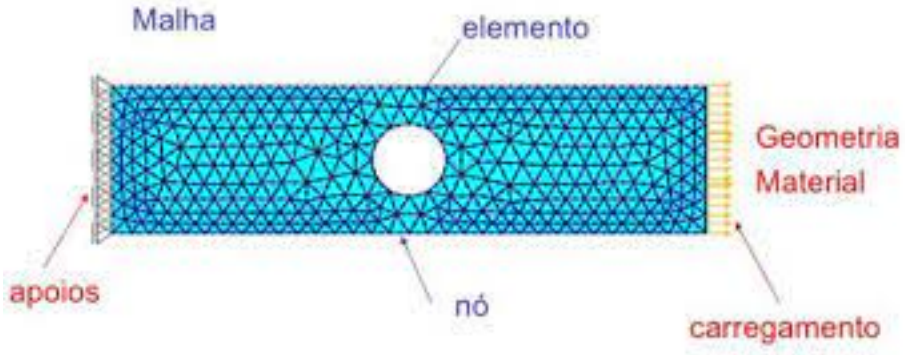

Figura 3: Malha não ordenada para análise de tensões na placa com furo. Fonte - Artigo UNESC. Autores: Serafim, E.S. e Piccinini A.C.[5]

Uma outra abordagem que pode tornar essa visualização mais simples é a resolução de um problema térmico de distribuição de calor em uma barra. Considera-se então que a barra possui apenas uma dimensão em sua direção axial. Desta forma, pode-se explorar diversas formas de distribuição dos nós e elementos para tentar otimizar o enfoque do método.

A maneira mais simples, sem dúvida é pensar numa malha ordenada, com distribuição uniforme:

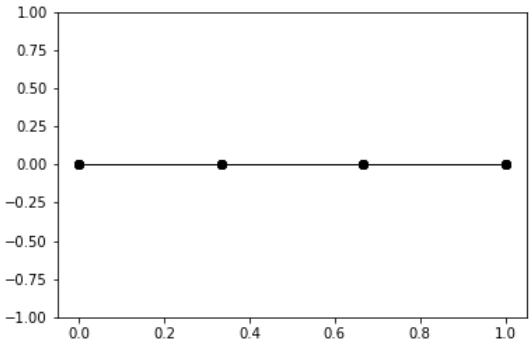

Figura 4: Malha ordenada 1D - Linear Uniforme - 4 nós

Entretanto, pode-se imaginar que a fonte de calor atinge o final da barra, desta forma, é interessante que haja mais pontos próximos a essa região, então pode ser vantajoso utilizar uma malha com outro tipo de distribuição:

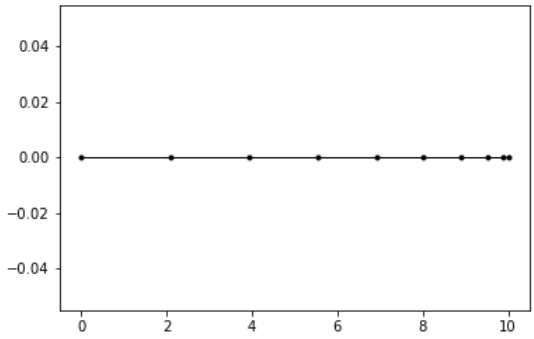

Figura 5: Malha ordenada 1D - distribuição quadrática - 30 nós

Ainda assim, existem outras distribuições que também podem ser convenientes a depender do problema a ser analisado.

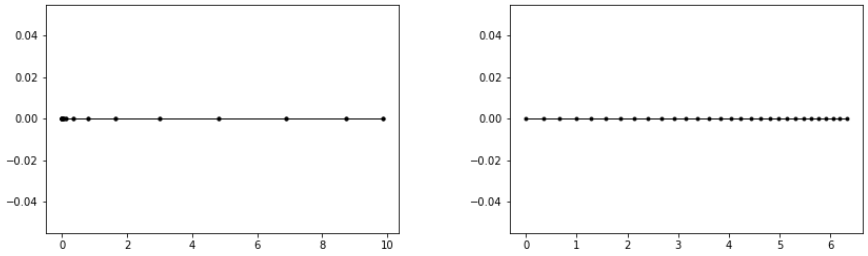

Figura 6: distribuições gaussiana e exponencial - 30 nós

Existem outros fatores também que contribuem para o uso de uma malha não estururada, que serão debatidos posteriormente e que dizem respeito ao refinamento da malha, daí será analisada a qualidade de cada elemento finito individualmente. Pode-se dizer com certa antecipação que a equilateralidade é um bom critério para refinamento de uma malha triangular.

3 Manipulação da malha

Um dos procedimentos que pode ser interessante é a definição do contorno. Por exemplo, caso não seja mais interessante analisar uma placa retangular, pode-se manipular os elementos para que eles não sejam mais em Y constante e passem a obedecer uma função. O exemplo abaixo mostra como isso pode ser feito com uma função senoidal.

$$f(x) = 0.2sin(2\pi x) \tag{1}$$

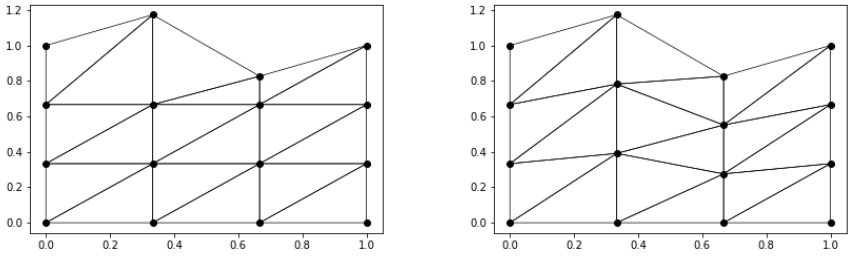

Figura 7: Malha com curva senoidal no topo ou em toda a malha

O conceito de curvatura pode ser de importante valia na manipulação da geometria dos elementos. o mesmo é inversamente proporcional ao raio de curvatura, responsável pelo aspecto curvilíneo da figura. Pode-se traçar um paralelo então, onde pode-se imaginar que qualquer reta é, na verdade, uma curva com raio de curvatura infinito.

Agora que já é conhecido o aspecto visual das malhas e possíveis configurações para as mesmas, pode-se aprofundar o estudo sobre sua natureza computacional e entender melhor seus componentes e o significado de cada um deles.

4 Componentes da Malha

Nesta seção serão abordados os aspectos numéricos dos componentes que integram as malhas. Verá-se quais as utilidades e o que representam cada um deles. De antemão, pode-se imaginar a principio que a criação de uma malha pode parecer algo nebuloso e complexo. Entretanto, a depender da geometria e da organização entre as conexões, pode tornar-se uma tarefa bastante lógica e simplificada. Na explicação a seguir serão utilizados exemplos simples para facilitar o entendimento, entretanto, conforme o estudo fica mais

aprofundado, nota-se que dificilmente a conectividade entre os nós seguirá um padrão tão intuitivo.

4.1 Vetores de Coordenadas

Os vetores representam o conjunto de coordenadas dos nós e são eles que vão determinar como será a geometria da malha. A montagem de uma malha se parece muito com aquele jogo infantil de conectar os pontos para formar um desenho, com o diferencial de que os pontos no caso são conectados de forma computacional. Da mesma forma que uma criança faz, pode-se também conectar os nós de maneira livre, seja formando trangulos, quadrados, retângulos, hexágonos ou as formas que preferir. Para definir como isto será feito existe a matriz de conectividade, mais conhecida como IEN. Ela definirá os pontos de conexão e o formato dos elementos a partir da localização dos pontos.

Exemplo: Para a malha da Figura 2, teremos os seguintes vetores para as coordenadas no eixo X e Y:

$$\vec{X} = \begin{pmatrix} 0 & 1/3 & 2/3 & 1 & 0 & 1/3 & 2/3 & 1 & 0 & 1/3 & 2/3 & 1 & 0 & 1/3 & 2/3 & 1 \end{pmatrix}$$

$$\vec{Y} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 1/3 & 2/3 & 2/3 & 2/3 & 2/3 & 1 & 1 & 1 & 1 \end{pmatrix}$$

4.2 Matriz de Conectividade

A matriz de conectividade, ou IEN, é a matriz que vai definir como os nós estarão interligados, ou seja, é ela que definirá o formato dos elementos presentes na malha. O número de linhas da IEN definirá quantos elementos estão presentes na malha, ao passo que o número de colunas representa a quantidade de nós presentes em cada elemento. Veja o exemplo da IEN da malha da Figura 2.

$$\mathbf{IEN} = \begin{pmatrix} 0 & 5 & 4 \\ 1 & 6 & 5 \\ 1 & 6 & 2 \\ 2 & 7 & 6 \\ 2 & 7 & 3 \\ 4 & 9 & 8 \\ 4 & 9 & 5 \\ 5 & 10 & 9 \\ 5 & 10 & 6 \\ 6 & 11 & 10 \\ 6 & 11 & 7 \\ 8 & 13 & 12 \\ 8 & 13 & 9 \\ 9 & 14 & 13 \\ 9 & 14 & 10 \\ 10 & 15 & 14 \\ 10 & 15 & 11 \end{pmatrix}$$

Se contar os nós da esquerda para a direita e de baixo para cima, percebe-se que o primeiro elemento é composto pelos nós 0, 5 e 4; formando o triângulo retângulo destacado na Figura 8 a seguir. Os demais elementos podem ser facilmente visualizados na IEN intuitivamente.

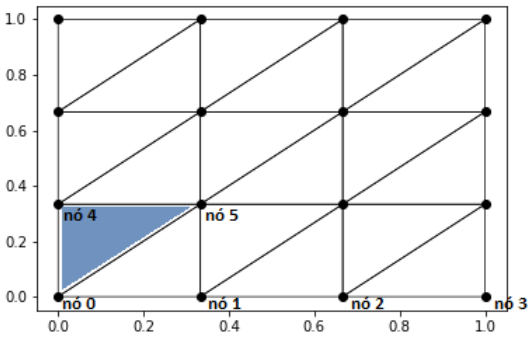

Figura 8: Representação do primeiro elemento da IEN, colorido.

Como afirmado anteriormente, essa disposição sequencial dos nós como encontrados na malha ordenada quase nunca é verdade para os geradores de malha mais sofisticados, em futuros tópicos serão explicados os porquês, mas antes, será esclarecido o processo de criação do gerador de malha ordenada.

5 Gerador de Malha Ordenada 2D

Nesta seção, será explicado o processo lógico por trás do gerador de malha ordenada 2D, que pode ser visto como a primeira versão do gerador de malhas.

5.1 O Código

O processo da geração foi separado em diferentes etapas num código em Python. Foram então criadas diversos métodos para diversificar as responsabilidades em diferentes componentes de código e manter uma maior ordem.

As bibliotecas utilizadas para ajudar na elaboração do código foram as seguintes:

- numpy
- matplotlib
- pathlib
- os

As duas primeiras são responsáveis pela manipulação matemática e visualização dos dados. As outras são responsáveis por organizar as pastas de resultados e direcionar as imagens das para o sistema operacional.

5.1.1 Parâmetros Principais

A função principal, que será responsável pelo retorno dos vetores de coordenadas e da matriz IEN, receberá os seguintes parâmetros e os aplicará num loop a ser explicado em seguida:

- Número de pontos no eixo X
- Número de pontos no eixo Y
- Largura da Placa (eixo X)
- Altura da Placa (eixo Y)
- Modelo da malha (quadrada ou triangular)
- Função para a curva (no topo ou na malha. Default: Y = Y)

5.1.2 Algoritmo

A lógica para formar os vetores de coordenadas é bastante simples, bastando apenas dois loops *for* entrelaçados, onde para cada passo do loop um valor de $\frac{j}{n_x-1}$ é apendido ao vetor de coordenadas $\vec{X}$ ao passo que um valor de $\frac{i}{n_y-1}$ é anexado a $\vec{Y}$. Sendo n_x e n_y o numero de pontos nos eixos X e Y, enquanto i e j são os índices do primeiro e do segundo laço, respectivamente.

Com o numero de pontos nos eixos e o modo (triangular ou retangular) definidos, existe uma função para gerar a IEN automaticamente também.

O método gerador da IEN funciona de maneira similar ao gerador das matrizes de coordenadas, com exceção de que o mesmo percorre apenas um laço de repetição. O processo é o seguinte:

A depender do modelo da malha, o método iniciará um vetor de zeros com a dimensão (nº de elementos x nº de vertices de cada elemento) e então, numa faixa de 0 ao nº de elementos, concatenará, na posição designada, os valores dos vértices correspondentes para cada elemento, retornando a matriz de conectividade em sua forma completa no final.

5.2 Qualidade dos elementos da malha

Uma métrica interessante a se avaliar é a qualidade dos elementos da malha, isto é, o quão capazes esses elementos são de sofrer discretizações espaciais sem acumular erros nas simulações. Os erros devidos às discretizações são inevitáveis, mas devem ser minimizados. Para isso, é necessario estabelecer critérios de avaliação da qualidade desses elementos. Nas malhas triangulares, por exemplo, o elemento que melhor responde à aproximação espacial dos operadores é o triângulo equilátero.

5.2.1 Triangulação de Delaunay

Para o gerador de malhas aleatórias, que será visto no tópico seguinte, utiliza-se o método de geração com a triangulação de Delaunay, que garante que numa determinada geometria, quaisquer que seja, a malha contenha a maior quantidade de triângulos equiláteros possível.

A triangulação de Delaunay é muito utilizada por algoritmos que utilizam machine learning para reconhecimento facial. Como a distorção espacial é menor com ela, isso garante uma maior precisão para o algoritmo.

Existem uma série de propriedades e regras matemáticas a serem seguidas para garantir uma triangulação típica de Delaunay, entretanto, este detalhamento fica sugerido como pesquisa ao leitor.

5.2.2 Critério de Equilateralidade

Pode-se afirmar então que um bom critério para a qualidade dos elementos da malha triangular é a equilateralidade de seus triângulos. Pode ser interessante então definir uma métrica que represente esta abstração na visualização da malha. Para medir este aspecto geométrico foi utilizada a seguinte lógica:

1. Calcula-se a Área do elemento utilizando a fórmula de Heron.

$$A = \sqrt{p(p - s_a)(p - s_b)(p - s_c)}; \tag{2}$$

$$p = \frac{s_a + s_b + s_c}{2} \tag{3}$$

   Onde A é a área, p é o semiperímetro e $s_{a,b,c}$ são os comprimentos de cada lado do triângulos, que podem ser medidos a partir da IEN e dos vetores de coordenadas.

2. Calcula-se então a altura de cada lado do triângulo utilizando a seguinte relação:

$$h_{a,b,c} = \frac{2A}{s_{a,b,c}}; \tag{4}$$

3. Sabendo que a relação entre a base e a altura no triângulo equilátero se dá por:

$$h_{eq} = \frac{\sqrt{3}}{2} s_{eq}; \tag{5}$$

   O critério então é calculado da seguinte forma:

$$E_{elemento} = \prod_{i=1}^{3} \frac{2}{\sqrt{3}} \frac{h_i}{s_i} = \frac{8 h_a h_b h_c}{3\sqrt{3} s_a s_b s_c}; \tag{6}$$

   Sendo assim, quanto mais próximo de 1, mais equilátero é o triângulo.

Será utilizado o exemplo da Figura 7 para visualizar uma aplicação prática do critério. Neste caso, quanto mais escuro o tom de azul da figura, mais próximo do triângulo equilátero o elemento está.

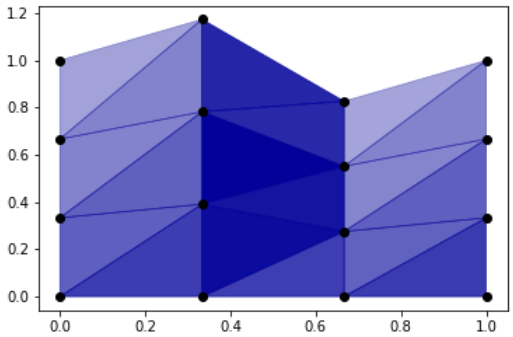

Figura 9: Malha com critério de equilateralidade

Perceba que no exemplo da placa, não é possível montar uma malha somente com triangulos equiláteros, por isso a triangulação de Delaunay se torna tão importante nesse panorama. Existem outras abordagens também para contornar este problema da qualidade da malha, como utilizar uma malha com elementos hibridos.

6 Gerador de Malha Aleatório 2D

Um dos exercícios para a melhor observação da qualidade e da não-estruturação da malha foi construir um gerador de malhas aleatórias, neste caso, foi utilizado o mesmo critério explicado antes e para a geração da IEN foi utilizada a triangulação de Delaunay para a obtenção da melhor configuração possível. Os resultados podem ser visualizados na Figura 10 a seguir.

Perceba que a aleatoriedade dos pontos provoca uma certa dificuldade de otimização dos triângulos, de forma que, mesmo utilizando a triangulação de Delaunay, existem muitos elementos de baixa equilateralidade. Geradores de malha mais modernos utilizam técnicas para otimizar essa distribuição dos pontos e é possível refinar a malha para que ela contemple melhor os aspectos da simulação desejada.

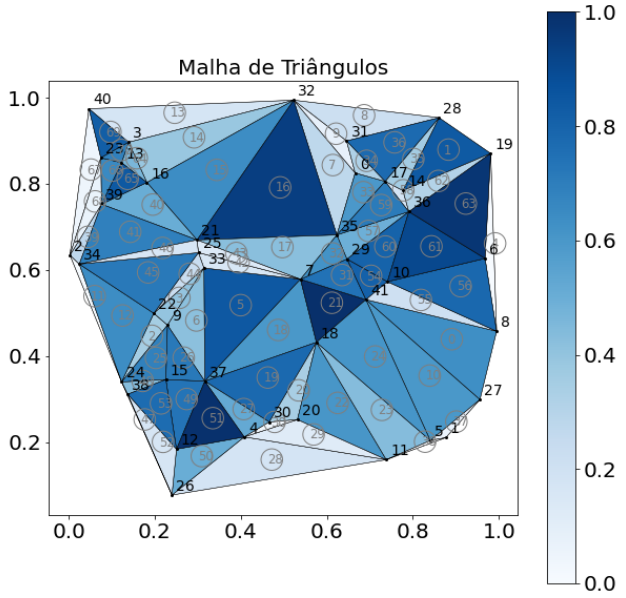

Figura 10: Malha triângular aleatória

Existem softwares CAE com o próprio gerador de malha embutido, como o Abaqus e o ANSYS, e existem outras opções para a geração de malha mais acessíveis, como o GMesh, que é de código livre e gratuito e o openFOAM para simulação com elementos finitos.

7 Gerador de Malha com Contorno 2D

Em problemas de transferência de calor e mecânica dos fluidos, frequentemente lidamos com modelagens matemáticas onde a solução das equações diferenciais dependem de condições iniciais, no caso de regime transiente, e de condições de contorno de maneira geral. Para isso, é preciso identificar as zonas de contorno e gerar matrizes de conectividade para cada contorno.

O método utilizado para a geração das matrizes de conectividade do contorno se utilizou do seguinte raciocínio:

1. Cada aresta interior à malha é compartilhada entre dois elementos, então bastou verificar a quantidade de elementos que tal vetor está contido e, caso o vetor seja considerado isolado, ou seja, pertencente a apenas um elemento, então este é um dos vetores de contorno.

2. Adiciona-se então cada nó deste vetor à matriz de conectividade do contorno, repete o procedimento para todos os elementos da IEN original e *voilà!*

O resultado obtido aplicando o contorno à malha aleatória pode ser visualizado abaixo:

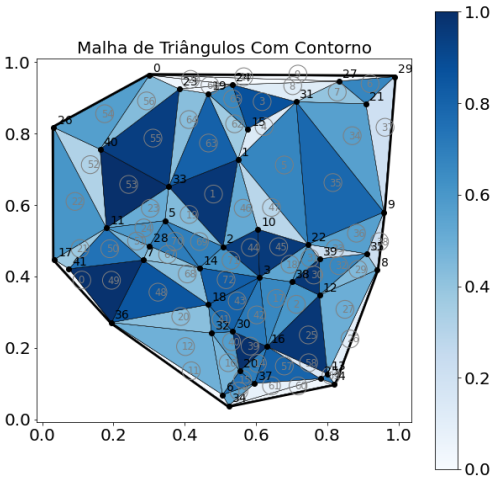

Figura 11: Malha triângular aleatória com contorno

O contorno pode ser observado pela maior espessura da linha no gráfico.

8 GMsh

Uma alternativa mais simples de gerar malhas pode ser o software GMsh, desenvolvido por Christophe Geuzaine e Jean-François Remacle e lançado sob a GNU General Public License. É um software livre e de fácil uso. O mesmo possui 4 módulos distintos, para descrição geométrica, geração de malha, resolução e pós processamento. Neste documento o maior enfoque é nos módulos de descrição geométrica e geração de malha.

Basicamente, assim que o usuário iniciar o aplicativo, será aberta uma janela, onde você pode começar a definir sua geometria.

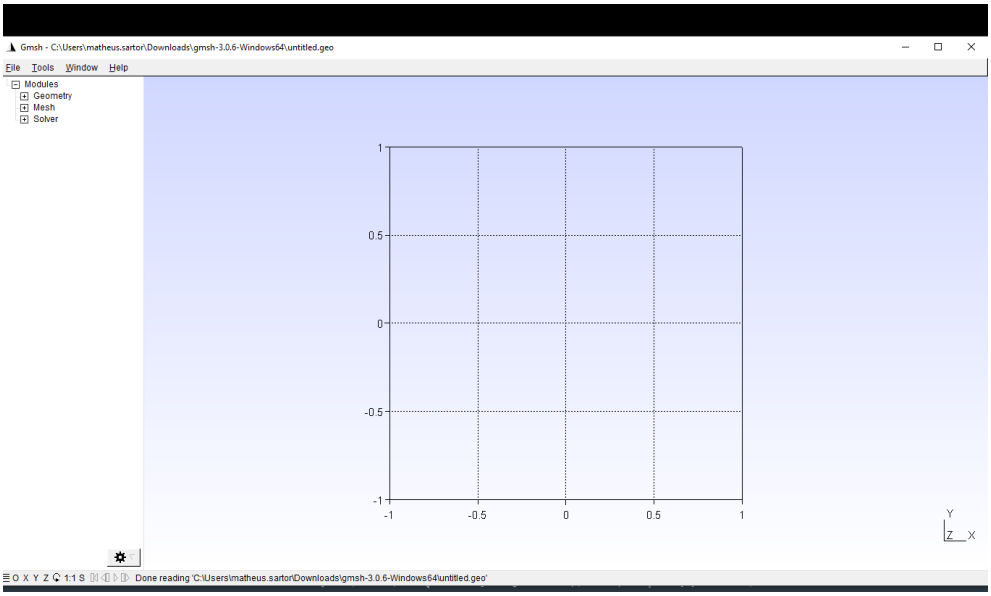

Figura 12: Tela inicial do software GMsh

Nesta janela, expandindo as opções na seguinte sequência *Geometry* $\longrightarrow$ *Elementary entities* $\longrightarrow$ *Add* o usuário poderá visualizar as entidades possíveis a se adicionar à geometria desejada, uma boa escolha é escolher os pontos, clicando em *Point* e ligando-o com linhas, clicando em *Straight line*, a seguinte geometria foi gerada utilizando pontos e linhas, simulando uma placa mãe fictícia.

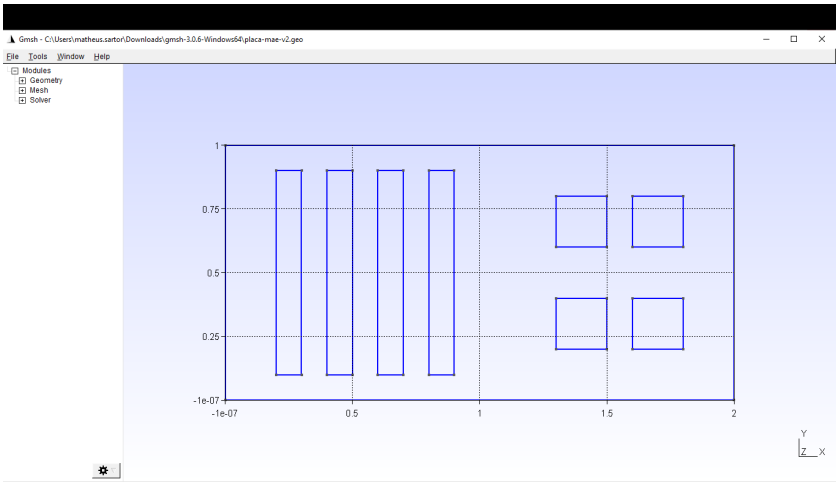

Figura 13: Placa Mãe Fictícia para simulação

Após ajustar a geometria da maneira desejada, para obter bons resultados, é preciso que o software caracterize as diferentes superficies. Para isso, existe o comando *Plane Surface*, para configurar uma superfície basta clicar no comando e clicar na linha de algum limite externo da superfície desejada e depois clicar em cada buraco, caso haja algum. Para finalizar a configuração, basta apertar a tecla "e"ou caso queira abortar o comando, use a tecla "q", isso vale para todos os comandos da aplicação. Quando a superfície estiver demarcada, será possível ver um tracejado percorrendo esta entidade.

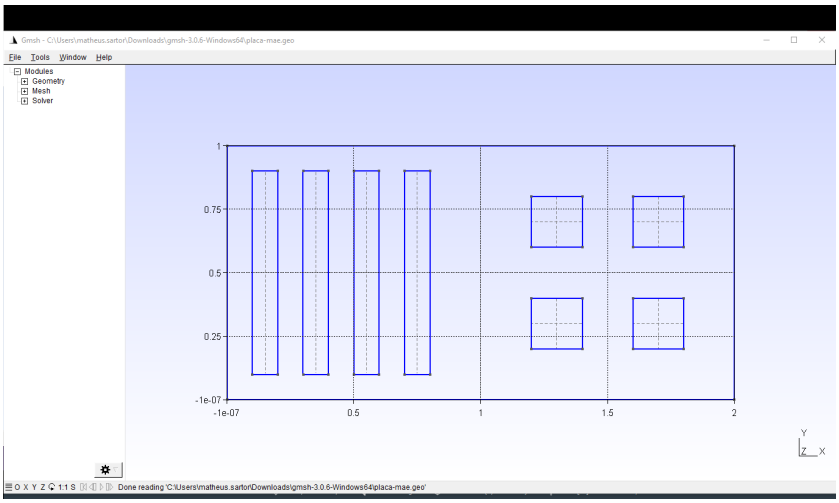

Figura 14: Placa Mãe com superfícies definidas

Após ter definido as superfícies, caso seja necessário, o usuário poderá verificar as nuances da geometria construída ao selecionar a opção *Edit script* no módulo *Geometry*. É interessante se familiarizar com a sintaxe do código em extensão .geo e pode ser extremamente útil para eventuais ajustes. Então, com as superfícies e o formato definidos, pode-se criar a malha 2D. O processo é muito simples, basta utilizar a opção 2D do módulo *Mesh* e o gerador irá gerar uma malha automaticamente, as configurações de visualização da malha também poderão ser modificadas apenas com um duplo clique na tela. Para este exemplo, foi utilizado uma geometria um pouco mais complexa, inspirada na placa mãe Asus Prime A320M-K/BR Amd AM4 DDR4 mATX.

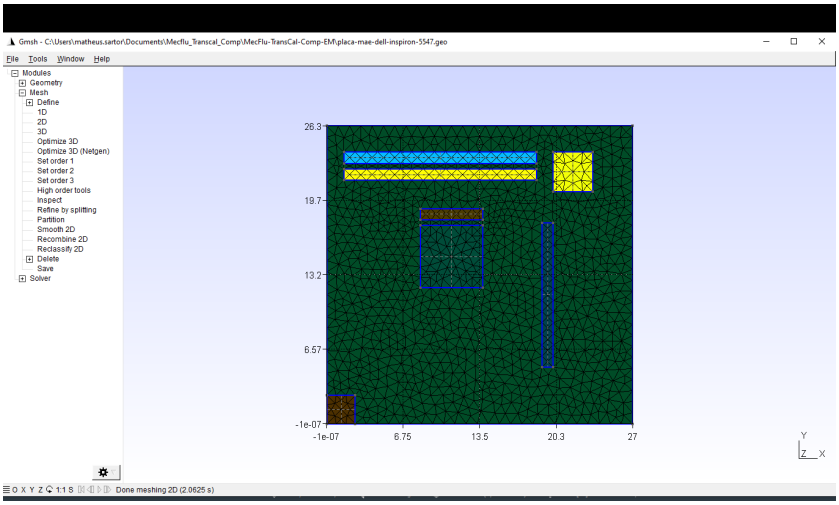

Figura 15: Mesh da Placa Mãe modelo Asus Prime

A partir daí, pode-se salvar o arquivo *.msh* indo no menu superior em *File* $\longrightarrow$ *Save Mesh* ou pressionando as teclas *"Ctrl+Shift+S"*. Este arquivo poderá ser lido no Python através da biblioteca *meshio*. (É interessante que o usuário possua versões compatíveis do meshio e do GMsh para executar um bom trabalho. Abaixo encontra-se um exemplo de código realizado para leitura de uma malha no Python utilizando as versões 3.2.15 do meshio e 3.0.6 do GMsh).

```
import meshio

msh = meshio.read('placa-mae-dell-inspiron-5547.msh')
X = msh.points[:,0]
Y = msh.points[:,1]
IEN = msh.cells['triangle']
npoints = len(X)
```

```
ne = IEN.shape[0]
regions = msh.cell_data['triangle']['gmsh:geometrical']
```

Desta forma, a partir da manipulação das matrizes é possível fazer o que quiser com as malhas no que se diz respeito ao método dos elementos finitos.

9 Conclusão

Existe uma complexidade matemática muito grande por trás da geração e manipulação de malhas e, apesar de buscar uma abordagem simplificada para o problema, ainda assim é uma tarefa difícil construir um gerador de qualidade. As soluções prontas como o GMsh certamente atenderão melhor às expectativas para simulações, visto que foram empregados anos de estudos e desenvolvimento por experts na área. Entretanto, foi bastante divertido e desafiador construir do zero um gerador simples e funcional para simulações não muito pretensiosas.

10 Referências

[1] BARROS, Felício Bruzzi. Métodos Sem Malha e Método dos Elementos Finitos Generalizados em Análise Não-Linear de Estruturas. 2002. 222 f. Tese (Doutorado) - Curso de Engenharia, Universidade de São Paulo, São Carlos, 2002.

[2] COSTA, Celme Torres F. Introdução Aos Métodos Numéricos. Universidade Federal Do Ceará, Campus Cariri 2010. 107p. Apostila.

[3] OWEN, S.J., A survey of unstructured mesh generation technology, in: Proc. 7th Internat. Meshing Roundtable, Dearborn, MI, USA, October 1998.

[4] DUARTE, R.N. Simulação Computacional: analise de uma célula de manufatura em lotes do setor de autopeças. Universidade Federal de Itajubá. Dissertação em engenharia de Produção. 2003. 187pg.

[5] SERAFIM, E.S. PICCININI A.C. Análise numérica da rigidez rotacional em uma ligação viga-pilar com dupla cantoneira de alma. Universidade do Extremo Sul Catarinense - UNESC. Artigo em engenharia civil. 2019. 19pg.

[6] ANJOS, G.R. COMPUTAÇÃO CIENTÍFICA PARA ENGENHEIROS - Universidade Federal do Rio de Janeiro - UFRJ. Apostila do curso de Transferência de Calor e Mecânica dos Fluidos Computacional. 2020. 101 pg.

A Apêndice - Códigos

A.1 Gerador de Malhas Ordenadas

```
import os
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
import matplotlib

def square(x):
    return 1-x**2
def gaussian(x,m,dp):
    return
        (((1/(np.sqrt(2*np.pi)*dp))*np.e**((-1/2)*((x-m)/dp)**2))/((1/(np.sqrt(2*np.pi)
def exponencial(x):
    return 1-(np.e**x)/np.e


def plotMalha1D(n_points=100,x_start=0,x_end= 10.,mode="linear"):
    x = np.linspace(x_start,x_end,n_points)
    y = np.zeros(x.shape)
    xx = []
    for i in range(0,n_points):
        relative_linear_point = i/(n_points-1)
        if(mode == "quad"):
            relative_point = square(relative_linear_point)
        if(mode == "gaussian"):
            relative_point = gaussian(relative_linear_point, .5, .1)
        if(mode == "exponencial"):
            relative_point = exponencial(relative_linear_point)
        absolute_point = relative_point*x_end
        xx.append(absolute_point)

    plt.plot(xx,y,'.')
    if(mode == "linear"):
        return plt.plot(x,y,'.')
    if(mode == "quad"):
        x =
            1/(1/(np.sqrt(2*np.pi)*2))*np.e**((-1/2)*((x-(x_end/2))/2)**2)
        return plt.plot(x,y,'.')

# matriz de conectividade IEN
```

```
def IENcreate(ne):
    IEN = np.zeros((ne,2))
    for e in range(0,ne):
        IEN[e] = [e,e+1]
    print(IEN)



def getXY(n_pointsx=4,n_pointsy=4,Lx = 1, Ly = 1,
    mode="square",function="None",coef=.2):
    x = []
    y = []
    for i in range(0,n_pointsy):
        for j in range(0,n_pointsx):
            x.append(j/(n_pointsx-1))
            y.append(i/(n_pointsy-1))
    IEN = IEN2D(n_pointsx,n_pointsy,mode)
    if(function=="sin"):
        y[-n_pointsx::] +=
            y[-n_pointsx::]*np.sin(np.multiply(x[:n_pointsx:],2*np.pi))*coef
        y[-2*n_pointsx:-n_pointsx:] +=
            y[-2*n_pointsx:-n_pointsx:]*np.sin(np.multiply(x[:n_pointsx:],2*np.pi))*coe
        y[-3*n_pointsx:-2*n_pointsx:] +=
            y[-3*n_pointsx:-2*n_pointsx:]*np.sin(np.multiply(x[:n_pointsx:],2*np.pi))*c
        y[-4*n_pointsx:-3*n_pointsx:] +=
            y[-4*n_pointsx:-3*n_pointsx:]*np.sin(np.multiply(x[:n_pointsx:],2*np.pi))*c
    x_array = np.asarray(x)
    x_array = (np.transpose(x_array.reshape(n_pointsx*n_pointsy)))
    y_array = np.asarray(y)
    y_array = (np.transpose(y_array.reshape(n_pointsx*n_pointsy)))
    IEN_array = np.asarray(IEN)
    return x_array,y_array,IEN_array

def plotMalhaQuadrada(X,Y,IEN):
    name = 'Malha Ordenada'
    Path(os.path.join('results', name)).mkdir(parents=True,
        exist_ok=True)
    ne = IEN.shape[0]
    crit_list = []
    for e in range(0,ne):
        [v1,v2,v3] = IEN[e]
        x1 = X[v1]; x2 = X[v2]; x3 = X[v3]
        y1 = Y[v1]; y2 = Y[v2]; y3 = Y[v3]
```

```
            a = np.sqrt((x3-x2)**2+(y3-y2)**2)
            b = np.sqrt((x3-x1)**2+(y3-y1)**2)
            c = np.sqrt((x2-x1)**2+(y2-y1)**2)
            s = (a + b + c)/2
            A = np.sqrt(s*(s-a)*(s-b)*(s-c)) # Formula de Heron
            ha = 2*A/a
            hb = 2*A/b
            hc = 2*A/c
            crit = ((ha*hb*hc)/(a*b*c))/((np.sqrt(3)/2)**3)
            crit_list.append((0,0,.6,crit))
        xy = np.stack((X, Y), axis=-1)
        verts = xy[IEN]
        ax=plt.gca()
        pc =
            matplotlib.collections.PolyCollection(verts,edgecolors=('black',),
                                                  facecolor='None',
                                                  linewidths=(0.7,))
        pc.set_color(crit_list)
        ax.add_collection(pc)
        plt.plot(X,Y,'ko')
        plt.savefig(os.path.join('results', name,
            'malha_senoidal_colorida.png'))

    def IEN2D(nx,ny, mode="square"):
        if(mode=="square"):
            ne = (nx-1)*(ny-1)
            IEN = np.zeros((ne,4),dtype=int)
            i = 0
            for e in range(0,ne):
                if(e!=0 and ((e-1)%(nx-1)) == (nx - 2)):
                    IEN[e] = [i+1,i+2,i+nx+2,i+nx+1]
                    i += 2
                else:
                    IEN[e] = [i,i+1,i+nx+1,i+nx]
                    i += 1
        elif(mode=="triangle"):
            ne = (nx-1)*(ny-1)*2
            IEN = np.zeros((ne,3),dtype=int)
            i = 0
            for e in range(0,ne,2):
                if(e!=0 and (((e/2) -1)%((nx-1))) == (nx - 2)):
                    IEN[e] = [i+1,i+nx+2,i+nx+1]
                    IEN[e+1] = [i+1,i+nx+2,i+2]
```

```
                i += 2
            else:
                IEN[e] = [i,i+nx+1,i+nx]
                IEN[e+1] = [i,i+nx+1,i+1]
                i += 1
    return IEN
if __name__ == '__main__':
    X,Y,IEN = getXY(mode="triangle", function="sin")
    plotMalhaQuadrada(X,Y,IEN)
```

A.2 Gerador de Malhas Aleatórias

```
import os
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
import matplotlib

def getXYRandom(nx=6,ny=7,Lx = 1, Ly = 1):
    X = np.random.uniform(0.0,Lx,(nx*ny,1))
    X = (np.transpose(X.reshape(nx*ny)))

    Y = np.random.uniform(0.0,Ly,(nx*ny,1))
    Y = (np.transpose(Y.reshape(nx*ny)))
    Tri = matplotlib.tri.Triangulation(X,Y)
    IEN = Tri.triangles
    print(X,Y,IEN)
    return X,Y,IEN

def plotMalhaTri(X,Y,IEN):
    name = 'MalhaTriangularComContorno'
    Path(os.path.join('results', name)).mkdir(parents=True,
        exist_ok=True)
    borda = verificaBorda(X, Y, IEN)
    print(borda)
    ne = IEN.shape[0]
    npoints = X.shape[0]
    crit_list = []
    A_sum = 0
    for e in range(0,ne):
        [v1,v2,v3] = IEN[e]
        x1 = X[v1]; x2 = X[v2]; x3 = X[v3]
```

```
    y1 = Y[v1]; y2 = Y[v2]; y3 = Y[v3]
    a = np.sqrt((x3-x2)**2+(y3-y2)**2)
    b = np.sqrt((x3-x1)**2+(y3-y1)**2)
    c = np.sqrt((x2-x1)**2+(y2-y1)**2)
    s = (a + b + c)/2
    A = np.sqrt(s*(s-a)*(s-b)*(s-c)) # Formula de Heron
    A_sum += A
    ha = 2*A/a
    hb = 2*A/b
    hc = 2*A/c
    crit = ((ha*hb*hc)/(a*b*c))/((np.sqrt(3)/2)**3)
    crit_list.append(crit)

fig, ax = plt.subplots()
fig.set_size_inches(10, 10)
xy = np.stack((X, Y), axis=-1)
verts = xy[borda]
ax=plt.gca()
pc =
    matplotlib.collections.PolyCollection(verts,edgecolors=('black',),
                                          facecolor = 'none',
                                            linewidths=(5))
ax.add_collection(pc)
ax.set_title("Malha de Triangulos Com Contorno")
plt.plot(X,Y,'ko')
ax = plt.triplot(X,Y,IEN,color='k',linewidth=0.5)
ax = plt.plot(X,Y,'ko',markersize=2)
for i in range(0,npoints):
   plt.text(X[i]+0.01,Y[i]+0.01,str(i),size =
       'x-small',color='k')
for e in range(0,ne):
    [v1,v2,v3] = IEN[e]
    xm = ( X[v1]+X[v2]+X[v3] )/3.0
    ym = ( Y[v1]+Y[v2]+Y[v3] )/3.0
    draw_circle = plt.Circle((xm+0.01, ym+0.01), 0.025,
        fill=False, color='gray')
    plt.text(xm,ym,str(e),size='xx-small',color='gray')
    plt.gcf().gca().add_artist(draw_circle)
plt.gca().set_aspect('equal')
plt.tripcolor(Tri, crit_list,
    edgecolors='k',cmap='Blues',vmin=0,vmax=1)
plt.colorbar()
```

```
    plt.savefig(os.path.join('results', name,
        'Malha_de_Triangulos_com_Contorno.png'))

def verificaBorda(X,Y,IEN):
    IENBounds = []
    arestas_compartilhadas = []
    arestas_isoladas = []
    for i in range(0,len(IEN)):
        [v1,v2,v3] = IEN[i]
        a1 = [v1,v2]
        a2 = [v2,v3]
        a3 = [v3,v1]
        for j in range(0,len(IEN)):
            if(i != j):
                [w1,w2,w3] = IEN[j]
                a11 = [w1,w2]
                a22 = [w2,w3]
                a33 = [w3,w1]
                a11_ = [w2,w1]
                a22_ = [w3,w2]
                a33_ = [w1,w3]
                a = [a11,a22,a33,a11_,a22_,a33_]
                if(a1 in a):
                    arestas_compartilhadas.append(a1)
                if(a2 in a):
                    arestas_compartilhadas.append(a2)
                if(a3 in a):
                    arestas_compartilhadas.append(a3)
        if(a1 not in arestas_compartilhadas):
            arestas_isoladas.append(a1)
        if(a2 not in arestas_compartilhadas):
            arestas_isoladas.append(a2)
        if(a3 not in arestas_compartilhadas):
            arestas_isoladas.append(a3)
    IENBounds = organizadorIEN(arestas_isoladas)
    return IENBounds

def organizadorIEN(lista):
    ## Percorre a lista de arestas isoladas, organizando pelos
        vertices
    print(lista)
    for i in range(0,len(lista)-1):
        for j in range(0,len(lista)):
```

```
            if(lista[i][1] == lista[j][0]):
                lista_org = lista[i+1]
                lista[i+1] = lista[j]
                lista[j] = lista_org
    ## Separa os contornos diferentes (ex.: externo e
       interno(buracos))
    split = []
    n = 0
    for m in range(0,len(lista)):
        if(lista[m-1][1]!=lista[m][0]):
            split.append(lista[i:m])
            i = m
    split.append(lista[n:])
    print(split)
    ## Armazena numa IEN os diferentes contornos,
    ## identificando tanto os externos quanto internos
    IENs = []
    for b in range(0,len(split)):
        contorno = split[b]
        print(contorno)
        IEN = np.zeros((len(contorno),1),dtype=int)
        IEN[0] = contorno[0][0]
        for k in range(0,len(contorno)-1):
            IEN[k+1] = contorno[k][1]
        IEN = IEN.reshape((1,len(contorno)))
        IENs.append(IEN)
    return IENs


if __name__ == '__main__':
    X,Y,Tri = getXYRandom()
    plotMalhaTri(X,Y,Tri)
```

B Apêndice - Simulações

Nesta seção seguem o resultado de algumas simulações feitas com malhas geradas pelo gerador criado com código Python.