Projeto Computacional
Transporte de Calor e Massa
Resolução da Equação do Calor utilizando o Método das Diferenças Finitas

Emanuel Couto Brenag - 190057131

Caso 7

13 de Outubro, 2020

1 Introdução

1.1 A Equação do Calor

A equação geral da difusão de calor, também conhecida como equação do calor foi desenvolvida pela primeira vez por Fourier em 1822. Ela é estudada não só nos cursos de Transferência de Calor, mas também nos cursos de Física Matemática e Equações Diferenciais Parciais, sendo uma das principais equações da Termodinâmica Clássica (Pré-Mecânica Estatística).

A equação do calor, junto com suas variantes, também é importante em muitos campos da ciência e da matemática aplicada. Na teoria da probabilidade, a equação do calor está conectada com o estudo de passeios aleatórios e movimento browniano por meio da equação de Fokker-Planck. A infame equação de Black-Scholes da matemática financeira é uma pequena variante da equação do calor, e a equação de Schrödinger da mecânica quântica pode ser considerada uma equação do calor no tempo imaginário. Na análise de imagens, a equação do calor às vezes é usada para resolver a pixelização e identificar bordas. Seguindo a introdução de métodos de "viscosidade artificial"de Robert Richtmyer e John von Neumann, soluções de equações de calor têm sido úteis na formulação matemática de choques hidrodinâmicos. Soluções da equação do calor também têm recebido muita atenção na literatura de análise numérica, começando na década de 1950 com o trabalho de Jim Douglas, D.W. Peaceman e Henry Rachford Jr.

A equação do calor é dada matematicamente por:

$$\rho C_p \frac{\partial T}{\partial t} = k\nabla^2 T + H$$

onde é T é o campo de Temperatura,ρ é a massa específica, C_p é o calor específico a pressão constante, k é a condutividade térmica do meio e H representa a Geração interna de calor. Para esse caso específico, tomaremos a geração interna como diferente de zero. A partir dessa equação é possível descrever as variações de temperatura em um espaço em função do tempo, mas dependendo da malha a ser analisada, a complexidade aumenta e torna-se necessário o uso de métodos numéricos e computacionais para facilitar a resolução do problema. Para esse trabalho, utilizaremos o Método das Diferenças Finitas.

1.2 O Método das Diferenças Finitas

O Método das Diferenças finitas é a ferramenta numérica que torna mais prático a forma de resolver a equação do calor para problemas de dimensões maiores. A aproximação pelo médodo das diferenças finitas nos permite expressar a EDP do calor em conjuntos de pontos para assim determinar as temperaturas. Para isso, usamos aproximações e Séries de Taylor para transformar a equação diferencial em um sistema linear. Para o caso específico, temos um problema onde há geração interna de energia $H = 10m^3/K$. Para utilizar o método, precisamos encontrar um jeito de desenvolver a equação bidimensional do calor de modo a discretizá-la e encontrar a equação de diferenças finitas em coordenadas cartesianas.

$$k\frac{T^p_{m+1,n} + T^p_{m,n} + T^p_{m-1,n}}{\Delta x^2} + k\frac{T^p_{m,n+1} + T^p_{m,n} + T^p_{m,n-1}}{\Delta y^2} + H = \rho C_p \frac{T^{p+1}_{m,n} - T^p_{m,n}}{\Delta t}$$

Assumimos $\Delta = \Delta x = \Delta y$ e definimos o número de Fourier como:

$$Fo = \frac{k\Delta t}{\rho C_p \Delta^2}$$

E a partir dessas definições e após efetuar algumas simplificações temos a discretização da temperatura para o método das diferenças finitas com geração interna de energia em cada ponto de uma malha dada por:

$$T^{p+1}_{m,n} = F_o(T^p_{m+1,n} + T^p_{m-1,n} + T^p_{m,n+1} + T^p_{m,n-1}) + (1 - 4F_o)T^p_{m,n} + \frac{H\Delta^2}{k}$$

Essa equação torna possível o desenvolvimento de um algoritmo computacional recursivo que descreva a difusão de temperatura no espaço e no tempo para diferentes geometrias.

1.3 Objetivos

- Obter a solução numérica para 4 malhas com quantidades de nós distintas
- Plotar perfis de temperatura da malha mais refinada para 5 diferentes partes do domínio do cálculo
- Computar o tempo computacional em função do número de nós

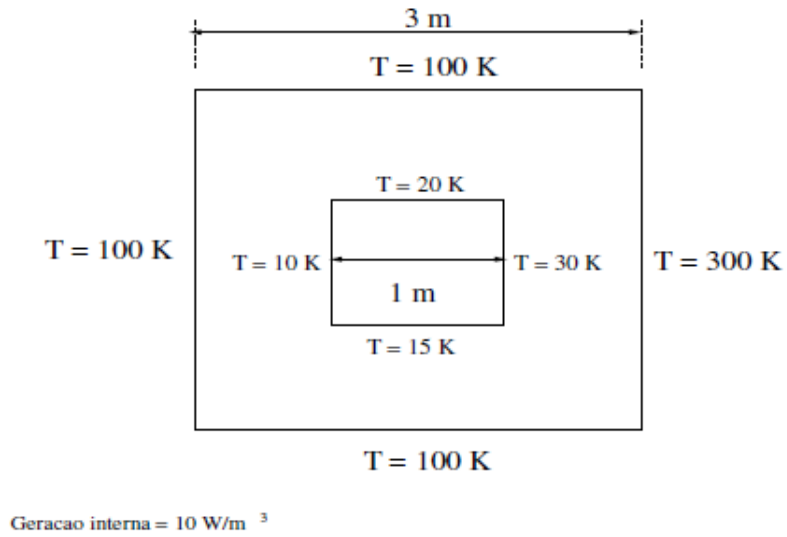

Figura 1 – Caso teste a ser simulado

2 Implementação

O código completo utilizado para gerar as simulações está comentado passo a passo e se encontra ao final desse relatório. Utilizarei essa seção do documento para explicar as partes mais importantes de sua implementação.

Para a realização do projeto, optou-se pela linguagem Python, pela imensa possibilidade de bibliotecas prontas que poderia utilizar para a plotagem de gráficos e geração de animações das malhas. Além disso, os códigos rodam um pouco mais lentamente quando comparados com linguagens compiladas como C e C++, mas as facilitações das bibliotecas compensam esse tempo. MATLAB também era uma alternativa, porém a difícil acessibilidade e por ser mais pesado, se tornou inviável.

Para facilitar a geração das matrizes e a manipulação das mesmas, utilizou-se a biblioteca numpy. Para a plotagem de gráficos e animações se utilizou a extensa biblioteca matplotlib e por último, para o cálculo de custo operacional, a biblioteca time.

Além das bibliotecas, utilizaram-se técnicas de desenvolvimento de software como a Programação Orientada a Objetos, com o intuito de facilitar a compreensão e manter o código mais sintetizado e econômico.

2.1 Ferramentas Computacionais

Ao realizar o projeto, algumas ferramentas computacionais foram utilizadas de modo a tornar mais a realização do projeto. Para o desenvolvimento do código, utilizou-se a IDE Visual Studio Code, mas a ferramenta mais interessante para o desenvolvimento do projeto foi o Google Colab, que é uma plataforma Google que permite "emprestar"poder computacional dos servidores da Google, facilitando o desenvolvimento de simulações e tornando o procedimento mais rápido. O Colab foi um grande facilitador do projeto, visto que tornou o processo de geração do vídeo muito mais prático e rápido, além da ferramenta permitir compilar o código por partes, facilitando a maneira de encontrar erros no funcionamento do código.

2.2 Criação da Malha

O primeiro passo na realização das simulações foi estabelecer os valores das constantes utilizadas e também os valores das condições de contorno externas e internas (furo) do caso teste. Para estabelecer as condições iniciais da malha optou-se por montar de uma maneira um pouco diferente. Foram utilizadas algumas funções da biblioteca math e algumas operações com matrizes para gerar a malha inicial. O código utilizado foi o seguinte:

```
# Set Boundary condition

#Need to set the conditions in order to keep the interior nodes Null

T[(lenY-1):, :] = Ttop
T[:1, :] = Tbottom
T[:, (lenX-1):] = Tright
T[:, :1] = Tleft

#Set the boundary conditions for the hole inside the mesh
T[hole_y, hole_x :(hole_x + hole_x_range)] = 20
T[hole_y + hole_y_range , hole_x: (hole_x + hole_x_range)] = 15
T[hole_y : (hole_y + hole_y_range + 1), hole_x] = 10
T[hole_y : (hole_y + hole_y_range + 1), (hole_x + hole_x_range)] = 30
T[hole_y + 1 : (hole_y + hole_y_range), hole_x + 1 : (hole_x + hole_x_range)] = 0
```

Figura 2 – Trecho do código para estabelecer as condições iniciais

2.3 Iterações no Tempo

A parte mais importante para conseguir estimar o modo como o calor flui são as iterações que ocorrem entre cada um dos nós da matriz utilizada. Cada uma dessas iterações representam um instante de tempo, e para cada um desses instantes foi gerado um novo elemento de um array de matrizes. Essa terceira dimensão dessas matrizes funcionava como o tempo. É importante mencionar que a cada iteração era necessário reafirmar as condições de contorno das bordas, visto que essas temperaturas deveriam se manter constante. O código utilizado para fazer essas iterações foi:

```
def iterate(self, iterations):
  for i in range(iterations):
    aux = self.state[len(self.state)-1]
    aux2 = np.zeros((len(aux), len(aux[0])))
    for i in range(1, len(aux)-1, delta):
        for j in range(1, len(aux[0])-1, delta):
            aux2[i][j] = self.factor*(aux[i+1][j] + aux[i-1][j] + aux[i][j+1] + aux[i][j-1]) + (1-4*self.factor)*(aux[i][j]) + self.int_gen

    aux2[(lenY-1):, :] = self.top
    aux2[:1, :] = self.bottom
    aux2[:, (lenX-1):] = self.right
    aux2[:, :1] = self.left

    aux2[hole_y, hole_x :(hole_x + hole_x_range)] = 20
    aux2[hole_y + hole_y_range , hole_x: (hole_x + hole_x_range)] = 15
    aux2[hole_y : (hole_y + hole_y_range + 1), hole_x] = 10
    aux2[hole_y : (hole_y + hole_y_range + 1), (hole_x + hole_x_range)] = 30
    aux2[hole_y + 1 : (hole_y + hole_y_range), hole_x + 1 : (hole_x + hole_x_range)] = 0
    self.state.append(aux2)
```

Figura 3 – Trecho do código responsável pelas iterações

2.4 Geração das Simulações

Um dos maiores obstáculos na realização do código foi a geração das imagens. Definir as informações apresentadas nos gráficos, escolher um padrão de cores adequado e conseguir controlar a quantidade de frames e velocidade do vídeo foram coisas extremamente desafiadoras, tanto na implementação quanto para o computador operar. O código utilizado para gerar o vídeo e as imagens respectivamente foi:

```
frames = 5000
grid = Grid(T, Ttop, Tbottom, Tleft, Tright, 0.25, 0.125)
grid.iterate(frames)

fig = plt.figure()
ax = plt.axes(xlim=(0, 3), ylim=(0, 3))
plt.xlabel(r'x')
plt.ylabel(r'y')
cont = ax.contourf(X, Y, grid.getState(0), colorinterpolation, cmap=colourMap)
cb = fig.colorbar(cont)
anim = animation.FuncAnimation(fig, animate, fargs=(grid,), frames=frames//5, save_count=frames//5)
writermp4 = animation.FFMpegFileWriter(fps=50, bitrate=1800)
anim.save('teste.mp4', writer=writermp4)
# plt.show()

fig = plt.figure()
ax = plt.axes(xlim=(0, 3), ylim=(0, 3))
plt.xlabel(r'x')
plt.ylabel(r'y')
contourplot = plt.contourf(X, Y, grid.getState(3000), colorinterpolation, cmap=colourMap)
cbar = plt.colorbar(contourplot)
plt.show()
```

Figura 4 – Trecho do código responsável pelas Simulações

2.5 Custo Computacional

Para calcular o tempo operacional gasto pelo computador para gerar as malhas de diferentes números de nós, utilizei a função apresentada abaixo de modo que subtraí o tempo em que o código plota o gráfico do tempo em que começa a primeira das iterações e printo o número de segundos gastos para tal. A função recebe como argumento a Matriz, o número de Fourier e a geração interna e printa o tempo gasto.

```
test_cases = [(T, factor, int_gen)]
times = []
grids = []
for test in test_cases:
  start = time.time()
  grid = Grid(test[0], Ttop, Tbottom, Tleft, Tright, test[1], test[2])
  grid.iterate(1000)
  grids.append(grid)
  times.append(time.time()- start)

grids[0].getState(1000)

print(times)
```

Figura 5 – Função do tempo de operação

O gráfico do tempo de execução do programa em função do número de nós é:

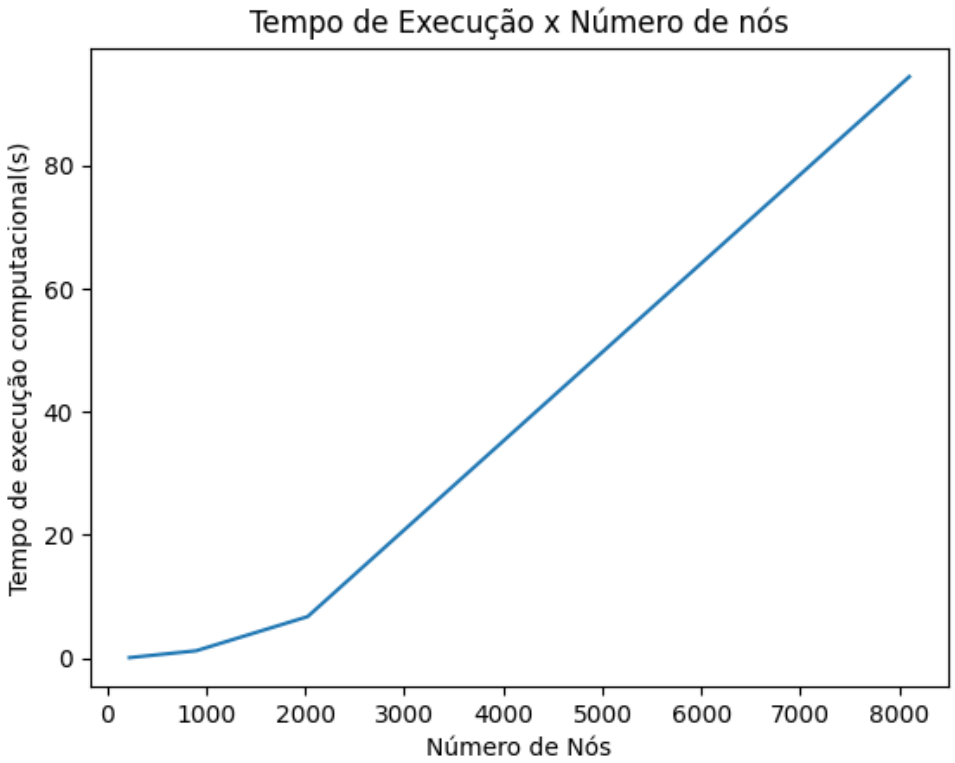

Figura 6 – Gráfico de Custo Operacional por Número de nós

Ao desenvolver e testar o código ficou evidente o quão complexo se tornam os processos com o aumento do número de nós, porém é evidente que existe um acréscimo de caráter mais linear no tempo, visto que tudo que ocorre são operações com matrizes.

3 Simulações

As simulações computacionais em regime transiente foram feitas para um caso específico. Foram testados diversos materiais diferentes, porém o material que gerou as simulações mais estáveis possui as seguintes características:

- $\rho \approx 1800 kg/m^3$
- $C_p \approx 1 kJ/kgK$
- $k \approx 0,81 W/m.K$
- $Fo = 0,25$ (para a malha mais refinada)
- $\Delta t = 0,5 s$

O código é capaz de computar quaisquer valores de número de Fourier e Geração interna, mas escolhi a que achei que as simulações geradas eram mais estáveis e interessantes. Para a escolha da malha mais refinada, foram observados diversos números de nós, porém para malhas maiores que 45x45 as diferenças no estado estacionário da simulação se tornavam cada vez mais imperceptíveis. A malha 90x90 é mais refinada que as anteriores e demorou um pouco mais para ser executada. Ela será utilizada para a análise dos perfis de temperatura posteriormente.

3.1 Simulações para diferentes tamanhos de malha

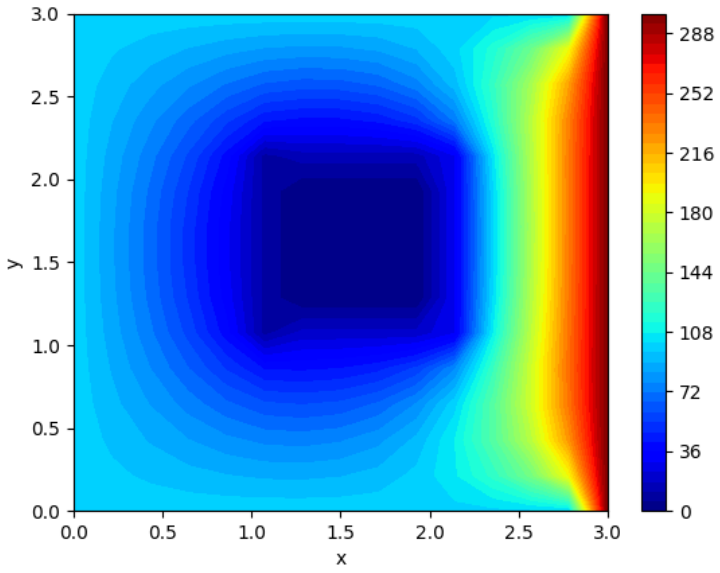

Figura 7 – Malha com 225 nós (15x15)

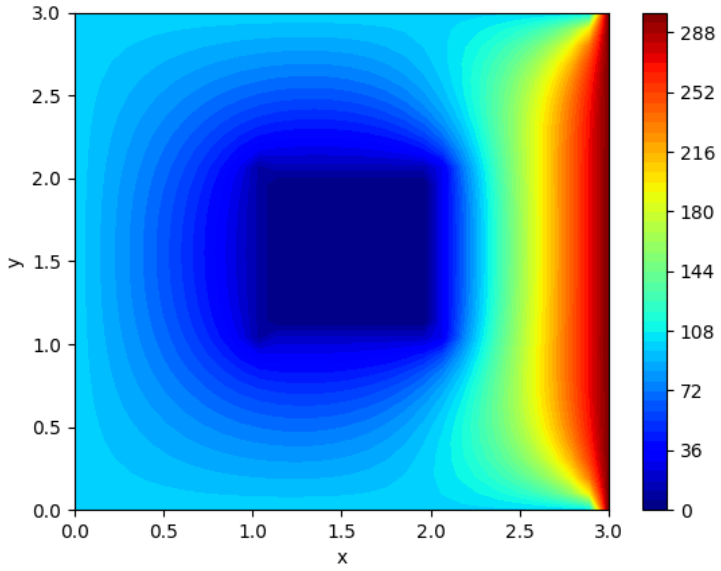

Figura 8 – Malha com 900 nós (30x30)

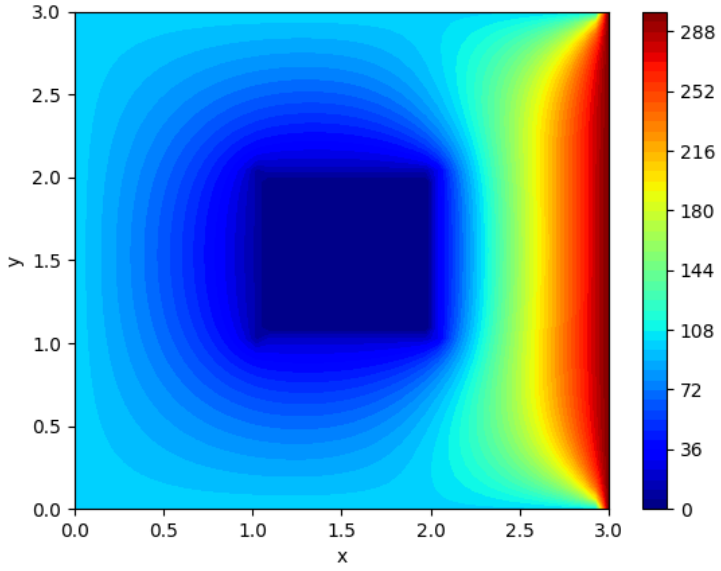

Figura 9 – Malha com 2025 nós (45x45)

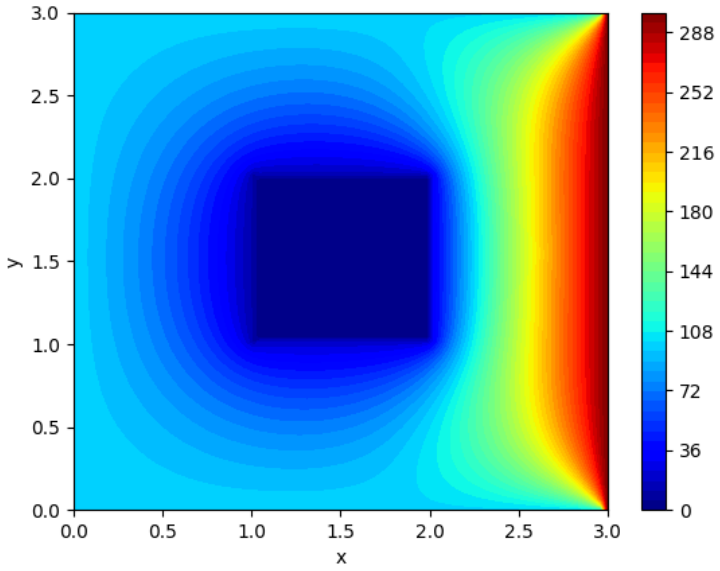

Figura 10 – Malha com 8100 nós (90x90)

3.2 Perfis de Temperatura

Os perfis de temperatura foram escolhidos com o propósito de verificar a temperatura da malha em regime permanente em algumas linhas da mesma. Cada um dos perfis mostra a propagação de calor para uma situação, sendo elas: proximidade das condições de contorno e o comportamento do campo de temperatura ao passar pelo furo adiabático da malha.

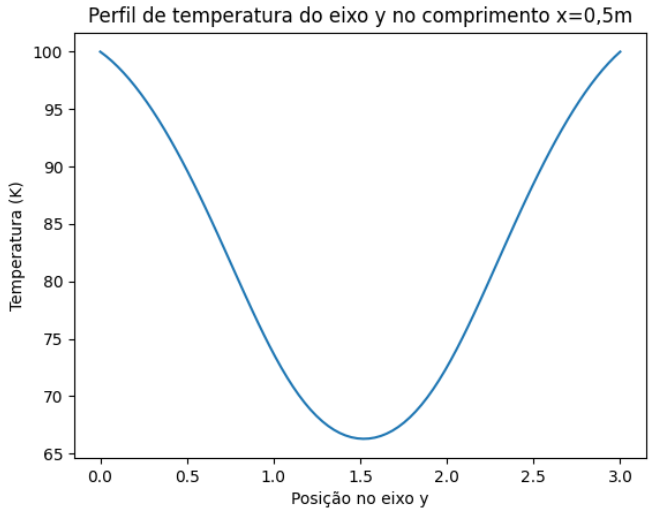

Figura 11 – Perfil de temperatura do eixo y em x = 0,5m

Nesse primeiro perfil temos a temperatura de toda uma linha no eixo y onde x está em 0,5m. Essa

linha cruza a malha de uma parede de 100K até outra parede de 100K, sem passar pelo buraco, porém pela influência das condições de contorno de temperatura da extremidade esquerda do buraco, podemos ver que ao cruzar o trecho onde temos o furo a temperatura cai, visto que lá as temperaturas são menores. Esse efeito também pode ser explicado pelo fato do trecho ser mais distante de ambas as bordas por estar no centro, e distante também da borda de 300K.

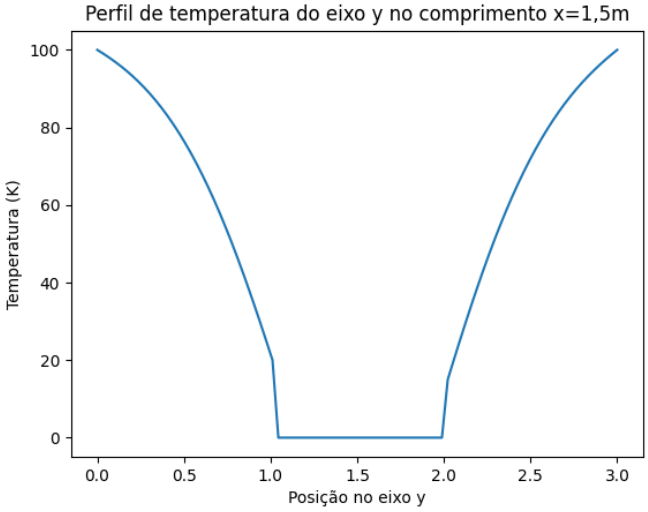

Figura 12 – Perfil de temperatura do eixo y em x = 1,5m

Já nesse segundo perfil, que cruza o eixo y inteiro, tendo seu ponto fixo em x = 1,5m, podemos observar que esse ponto no eixo x passa exatamente pelo centro do furo da malha. Pode-se perceber uma descida estável na temperatura, provocada pelas condições de contorno que regem as temperaturas nas paredes do buraco. Além disso, percebe-se que em todo esse trecho adiabático, as temperaturas se encontram dentro do que é estabelecido pelo roteiro experimental (T=0K).

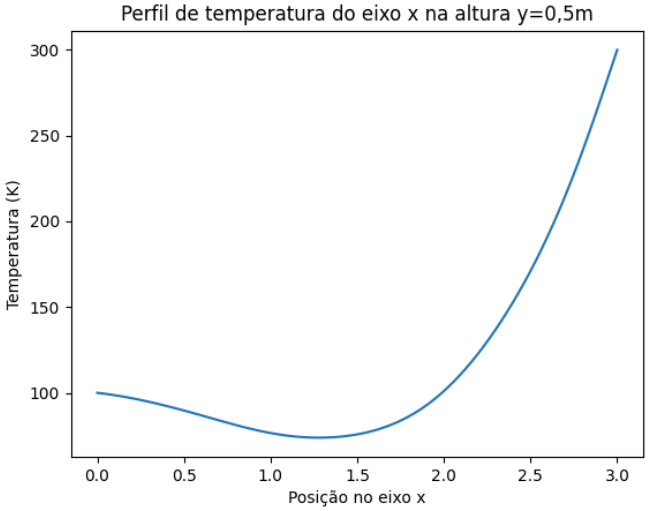

Figura 13 – Perfil de temperatura do eixo x em y = 0,5m

Para esse terceiro perfil, acontece o que é esperado: uma variação menos brusca da extremidade esquerda até o meio da malha, que tem temperaturas entre 10-20K, e depois a temperatura sobe de maneira muito rápida pela influência do fluxo de calor gerado pela borda que está aquecida a 300K.

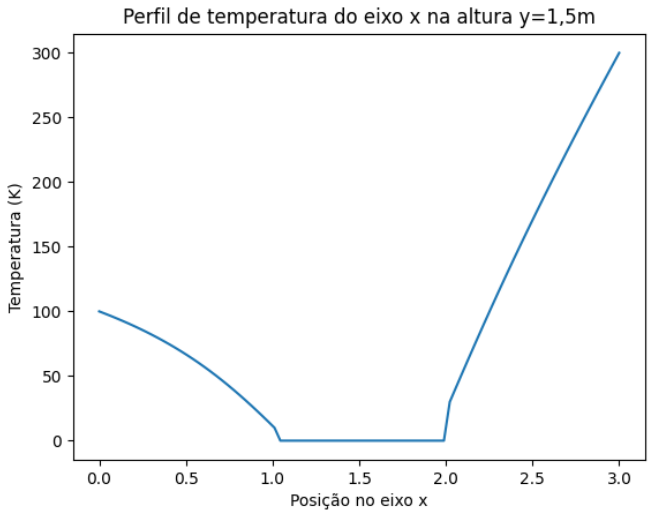

Figura 14 – Perfil de temperatura do eixo x em y = 1,5m

Assim como o terceiro perfil, esse quarto perfil tem uma variação menor da esquerda até o meio e uma subida extremamente brusca do meio até a borda direita, que é 3 vezes mais quente. O fato que diferencia essas duas bordas é que para percorrer esse caminho na altura y=1,5m, passa-se pelo buraco onde a temperatura é de 0K, sendo perceptível que a temperatura chega a 0 nesse momento e tem um degrau na saída do buraco, que pode ser explicado pela condição de contorno das bordas do mesmo.

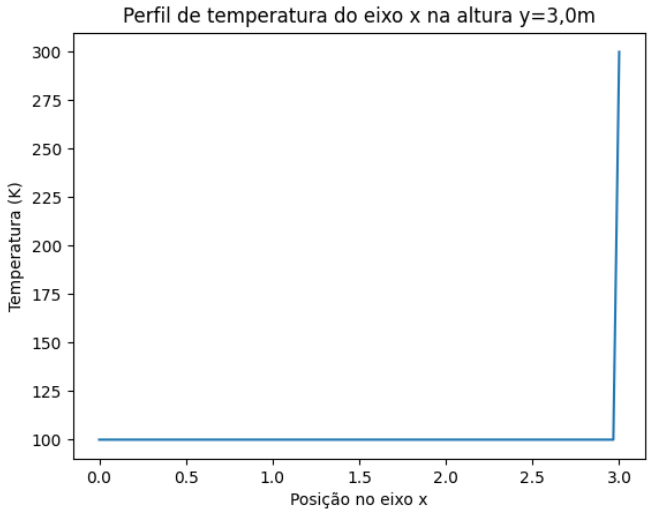

Figura 15 – Perfil de temperatura do eixo x em y = 3m

Esse quinto e último perfil de temperatura foi utilizado para verificar como o programa interpreta as

condições de contorno em bordas de temperaturas muito diferentes. Cruzando o eixo x na altura máxima de y=3m, a borda tem temperatura constante de 100K, até chegar na extremidade onde a borda tem temperatura de 300K, que prevalece no nó, por isso ocorre essa subida linear, como um gráfico de uma função delta de Dirac.

4 Comparações

Foram plotadas simulações para a mesma malha, com mesmo Número de Fourier e quantidade de nós com o intuito de observar as mudanças que valores diferentes de geração interna de energia proporcionam.

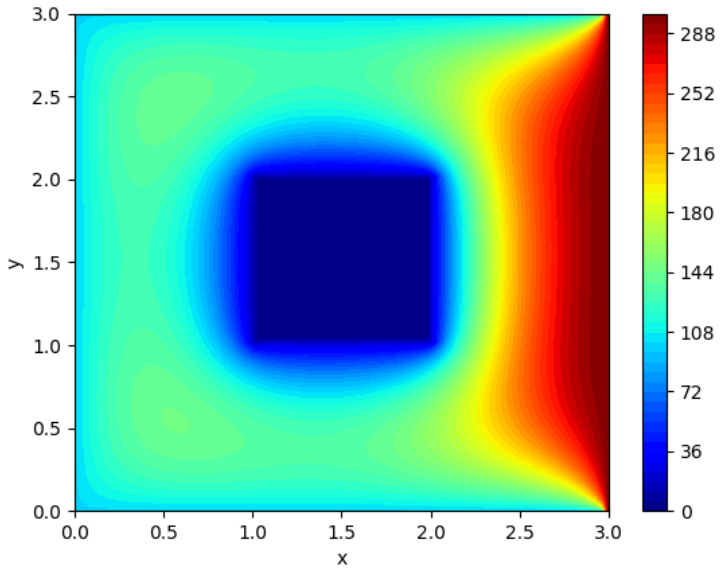

Figura 16 – Mapa de Calor com 10x mais geração interna

Essa primeira figura representa uma malha com uma única alteração do valor de H, que variou de $10W/m^3$ para $100W/m^3$. Essa variação gera uma cor mais esverdeada na malha, que mostra que com essa grande geração interna, as temperaturas mais ao centro da distância entre a borda e o buraco são superiores às bordas de 100K. A temperatura volta a descer quando se aproxima do buraco, onde pode-se observar uma mudança extremamente brusca na temperatura, onde o tom de azul escurece bruscamente.

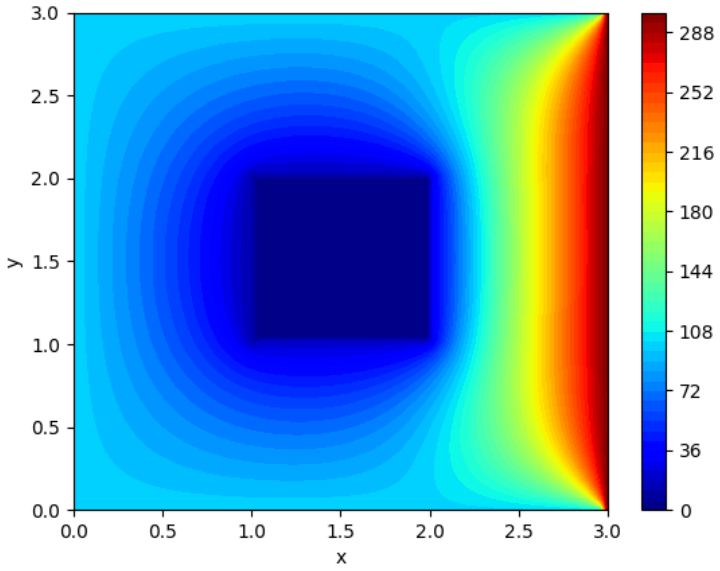

Figura 17 – Mapa de Calor sem geração interna

Ao comparar a malha sem geração interna de calor, podemos observar que as temperaturas se propagam de maneira muito parecida com a malha com a geração interna de $10W/m^3$, porém na malha com geração podemos observar que em cada um dos pontos as temperaturas são levemente superiores, com diferenças muito pequenas. Pode-se observar que próximo as bordas, os tons de azul permanecem claro por mais tempo ao se aproximar do meio da malha.

5 Conclusão

A partir do desenvolvimento computacional e de uma análise dos resultados obtidos pelas simulações, foi possível observar o modo como o calor flui em uma malha com diferentes temperaturas em cada uma de suas extremidades. Além disso, foi possível identificar o impacto causado por diferentes valores de geração interna. Pode-se concluir que o experimento foi um sucesso, visto que as comparações dos perfis e dos casos com diferentes gerações entregaram resultados que são tangíveis com a realidade.

6 Bibliografia

As imagens utilizadas foram retiradas do roteiro do trabaalho computacional, localizado no site do Professor Rafael Gabler.

- INCROPERA, Frank P. Fundamentos de Transferência de Calor e de Massa, LTC 8. Ed., 2019.
- VASCONCELOS, Marcelo Augusto. Uma Aplicação dos Métodos das Diferenças Finitas e Elementos Finitos a problemas térmicos. Orientador: Lineu José Pedroso. 2018. Monografia de Projeto Final em Estruturas, Universidade de Brasília, 2018.

7 Código

```
   # -*- coding: utf-8 -*-
#Numerical Solution for the Heat Equation using Finite Difference Method
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
import math
import time

# Set Dimension and delta
lenX = lenY = 90 #we set it rectangular
delta = 1
#these are not the real values of delta and the length. The size of the mesh is constant,
lenX is the number of knots per line and column and the real value of delta is
defined by 3/lenX
#Some tricks i used to fill the gaps inside the matrix and define the 1x1 hole in the center
knots = lenX/delta

hole_x = math.floor(knots/3)
hole_y = math.floor(knots/3)

hole_x_range = round(knots/3)
hole_y_range = round(knots/3)

#Define the thermal conductivity of the specific material
k= 80
#Define the thermal diffusivity of the specific material
alpha= 0.25

#define the internal generation constant
H= 10
int_gen= H*(3/lenX)**2/k

#define the time interval for the fourier number (delta_t)
delta_t=1
#define the fourier number
factor = alpha*delta_t/(delta**2)



# Boundary condition for the meshgrid

Ttop = 100
Tbottom = 100
Tleft = 100
```

```
Tright = 300

# Initial guess of interior grid/ fill the matrix with 0
Tguess = 0

# Set colour interpolation and colour map
colorinterpolation = 50
colourMap = plt.cm.jet #you can try: colourMap = plt.cm.coolwarm

# Set meshgrid
X, Y = np.meshgrid(np.linspace(0, 3, lenX), np.linspace(0, 3, lenY))

# Set array size and set the interior value with Tguess
T = np.empty((lenX, lenY))
T.fill(Tguess)

# Set Boundary condition

#Need to set the conditions in order to keep the interior nodes Null

T[(lenY-1):, :] = Ttop
T[:1, :] = Tbottom
T[:, (lenX-1):] = Tright
T[:, :1] = Tleft

#Set the boundary conditions for the hole inside the mesh
T[hole_y, hole_x :(hole_x + hole_x_range)] = 20
T[hole_y + hole_y_range , hole_x: (hole_x + hole_x_range)] = 15
T[hole_y : (hole_y + hole_y_range + 1), hole_x] = 10
T[hole_y : (hole_y + hole_y_range + 1), (hole_x + hole_x_range)] = 30
T[hole_y + 1 : (hole_y + hole_y_range), hole_x + 1 : (hole_x + hole_x_range)] = 0

#class to define the iterations and avoid the matrix in t=t+1 interacting with
the matrix in t=t
class Grid:
  def __init__(self, matrix, Ttop, Tbottom, Tleft, Tright, factor, int_gen):
    self.state = [[]]
    self.state[0] = matrix
    self.top = Ttop
    self.bottom = Tbottom
    self.left = Tleft
    self.right = Tright
    self.factor = factor
    self.int_gen = int_gen
  def iterate(self, iterations):
    for i in range(iterations):
      aux = self.state[len(self.state)-1]
      aux2 = np.zeros((len(aux), len(aux[0])))
```

```
      for i in range(1, len(aux)-1, delta):
          for j in range(1, len(aux[0])-1, delta):
              aux2[i][j] = self.factor*(aux[i+1][j] + aux[i-1][j] + aux[i][j+1]
              + aux[i][j-1]) + (1-4*self.factor)*(aux[i][j]) + self.int_gen

#just to make sure that the boundary conditions stay still
      aux2[(lenY-1):, :] = self.top
      aux2[:1, :] = self.bottom
      aux2[:, (lenX-1):] = self.right
      aux2[:, :1] = self.left

      aux2[hole_y, hole_x :(hole_x + hole_x_range)] = 20
      aux2[hole_y + hole_y_range , hole_x: (hole_x + hole_x_range)] = 15
      aux2[hole_y : (hole_y + hole_y_range + 1), hole_x] = 10
      aux2[hole_y : (hole_y + hole_y_range + 1), (hole_x + hole_x_range)] = 30
      aux2[hole_y + 1 : (hole_y + hole_y_range), hole_x + 1 : (hole_x + hole_x_range)] = 0
      self.state.append(aux2)

  def getState(self, t):
    return self.state[t]

#this function is to define some conditions for the animation
def animate(t, grid):
  cont = ax.contourf(X, Y, grid.getState(t*5), colorinterpolation, cmap=colourMap)


#this function recieves T, Fourier and the internal generation in order to calculate
the time to run the iterations
test_cases = [(T, factor, int_gen)]
times = []
grids = []
for test in test_cases:
  start = time.time()
  grid = Grid(test[0], Ttop, Tbottom, Tleft, Tright, test[1], test[2])
  grid.iterate(5000)
  grids.append(grid)
  times.append(time.time()- start)

grids[0].getState(5000)

print(times)

#this function generates the .mp4 file since the initial state

frames = 5000
grid = Grid(T, Ttop, Tbottom, Tleft, Tright, 0.25, 0.125)
grid.iterate(frames)
```

```
fig = plt.figure()
ax = plt.axes(xlim=(0, 3), ylim=(0, 3))
plt.xlabel(r'x')
plt.ylabel(r'y')
cont = ax.contourf(X, Y, grid.getState(0), colorinterpolation, cmap=colourMap)
cb = fig.colorbar(cont)
anim = animation.FuncAnimation(fig, animate, fargs=(grid,), frames=frames//5, save_count=frames//5)
writermp4 = animation.FFMpegFileWriter(fps=50, bitrate=1800)
anim.save('teste.mp4', writer=writermp4)
# plt.show()
#plot the image

fig = plt.figure()
ax = plt.axes(xlim=(0, 3), ylim=(0, 3))
plt.xlabel(r'x')
plt.ylabel(r'y')
contourplot = plt.contourf(X, Y, grid.getState(5000), colorinterpolation, cmap=colourMap)
cbar = plt.colorbar(contourplot)
plt.show()

#to plot the temperature profiles
def column(matrix, i):
  return [row[i] for row in matrix]
Temperatura= column(grid.getState(3000),15)
plt.plot(np.linspace(0,3,len(Temperatura)),Temperatura)
plt.title('Perfil de temperatura do eixo y no comprimento x=0,5m')
plt.xlabel(' Posio  no eixo y')
plt.ylabel('Temperatura (K)')
plt.show()

def column(matrix, i):
  return [row[i] for row in matrix]
Temperatura= column(grid.getState(3000),45)
plt.plot(np.linspace(0,3,len(Temperatura)),Temperatura)
plt.title('Perfil de temperatura do eixo y no comprimento x=1,5m')
plt.xlabel(' Posio  no eixo y')
plt.ylabel('Temperatura (K)')
plt.show()


Temperatura= grid.getState(3000)[45][:]
plt.plot(np.linspace(0,3,len(Temperatura)),Temperatura)
plt.title('Perfil de temperatura do eixo x na altura y=1,5m')
plt.xlabel(' Posio  no eixo x')
plt.ylabel('Temperatura (K)')
plt.show()

Temperatura= grid.getState(3000)[89][:]
```

```
plt.plot(np.linspace(0,3,len(Temperatura)),Temperatura)
plt.title('Perfil de temperatura do eixo x na altura y=3,0m')
plt.xlabel(' Posio  no eixo x')
plt.ylabel('Temperatura (K)')
plt.show()

Temperatura= grid.getState(3000)[15][:]
plt.plot(np.linspace(0,3,len(Temperatura)),Temperatura)
plt.title('Perfil de temperatura do eixo x na altura y=0,5m')
plt.xlabel(' Posio  no eixo x')
plt.ylabel('Temperatura (K)')
plt.show()
```