Fundação Getulio Vargas
Escola de Matemática Aplicada

Álgebra Linear

Solução Númerica da Equação do Calor

Professor Yuri Fahham Saporito

Cleyton Vinicius

Otávio Augusto

Rio de Janeiro

2022

Sumário

1 Introdução

Esse relatório tem como objetivo mostrar a equação do calor e a sua solução númerica pelo método implícito, com uma implementação em Python.

A resolução de equações diferenciais por métodos numéricos através de computadores é um tópico central na Matemática Moderna, encontrando aplicações na Matemática Aplicada, Engenharia, Física, Economia, Biologia, Medicina, dentre outras inúmeras áreas do conhecimento. As equações diferenciais parciais constituem um campo que desperta particular interesse, uma vez que são de difícil resolução analítica e estão presentes em todas as áreas citadas acima.

Como é de se esperar, dada tamanha relevância do tema, há uma infinidade de métodos e técnicas diferentes, variando em complexidade e eficiência computacional. Neste trabalho apresentaremos e implementaremos o Método Implícito de Euler em conjunto com o Método das Diferenças Finitas, algumas das ferramentas mais simples e conhecidas em nosso contexto.

2 A Equação do Calor

2.1 O que é?

A equação do calor é uma equação diferencial parcial utilizada para a modelagem matemática da difusão do calor em sólidos, pelo que é conhecida também como equação da difusão. A sua teoria foi desenvolvida inicialmente por Joseph Fourier no século XIX, sendo esse o problema que o motivou a criar as Séries de Fourier.

Sua forma mais conhecida é a que modela a condução de calor em um sólido homogêneo, isotrópico e isolado, isto é, sem presença de fontes externas de calor ou trocas de calor com o ambiente, consistindo na seguinte equação:

$$\frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) \tag{1}$$

em que $T = T(x, y, z, t)$ representa o campo de temperaturas (temperatura em cada ponto do espaço, em cada instante de tempo) e α é uma constante positiva chamada coeficiente de difusão térmica.

Resolver a equação significa encontrar uma função $T(x, y, z, t)$ que satisfaça a relação acima. De fato, há uma família de funções que solucionam a equação. É comum que sejam impostas condições sobre a função T procurada, conhecidas como condições iniciais e condições de contorno.

A condição inicial é a temperatura em cada ponto do domínio no espaço no instante inicial de tempo t_0 e representa, fisicamente, o estado inicial do sólido considerado. Por outro lado, condições de contorno são restrições impostas à função nos extremos de seu domínio (como, por exemplo, a temperatura do sólido em seus extremos).

3 Solução Númerica

3.1 Método de Euler

O método de Euler é um dos algoritmos mais simples e mais utilizados para a resolução numérica de equações diferenciais.

Dada uma função derivável $y = y(t)$ definida em um intervalo, realiza-se uma discretização da função: toma-se um ponto t_0 do intervalo, dito valor inicial, e um valor pequeno $h > 0$, chamado de passo.

De fato, dado um ponto x no domínio, temos $y(t+h) = y(t) + h \cdot y'(t) + r(h)$, com $\lim_{h\to 0} r(h) = 0$. Logo, para h suficientemente pequeno, podemos escrever $y(t+h) \approx y(t) + h \cdot y'(t)$. Assim, dado $y_0 = y(t_0)$, escrevemos $y_1 = y(t_0) + h \cdot y'(t_0) \approx y(t_0 + h)$. De uma forma geral, definindo $t_{n+1} = t_n + h$, temos

$$y(t_{n+1}) \approx y_n + h \cdot y'(t_n) \tag{2}$$

Dado um problema de valor inicial

$$y'(t) = f(t, y(t))$$
$$y(t_0) = y_0$$

podemos aplicar o método acima para encontrar numericamente uma aproximação para a solução:

$$y_{n+1} = y_n + h \cdot f(t_n, y_n)$$

onde a sequência $(y_n)_n$ aproxima-se da função nos pontos $t_0 + n \cdot h$. Esta é a forma explícita do método. Há também a forma implícita:

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1})$$

.

A diferença fundamental entre os dois métodos é que no explícito resolve-se $y_{n+1} = F(y_n)$, enquanto que na variante implícita resolve-se $G(y_{n+1}, y_n) = 0$ para encontrar y_{n+1}.

3.2 Método das Diferenças Finitas

O método das diferenças finitas consiste em um artifício para resolução de equações diferenciais cuja base é a aproximação de derivadas por diferenças finitas (i.e., funciona através de uma discretização):

$$y'(t) = \frac{y(t+h) - y(t)}{h} + r(h)$$
$$\lim_{h\to 0} r(h) = 0$$

Seguindo a notação da subseção anterior, temos

$$y'(t_n) \approx \frac{y(t_n + h) - y(t_n)}{h}$$

para h suficientemente pequeno.

No mesmo espírito, temos

$$y''(t_n) \approx \frac{1}{h}\left(\frac{y(t_n+h)-y(t_n)}{h} - \frac{y(t_n)-y(t_n-h)}{h}\right)$$

ou, finalmente,

$$y''(t_n) \approx \frac{y(t_n+h)-2y(t_n)+y(t_n-h)}{h^2}$$

.

3.3 As Condições do Problema

É dada, como condição inicial, uma distribuição arbitrária de calor sobre uma barra, denotada $T(x,0)$.

Será considerada a versão mais simples do problema:

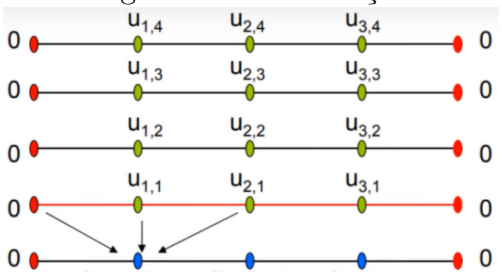
$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

isto é, o caso unidimensional sem fonte externa de calor. Consideraremos também a condição de contorno de Dirichlet: $T(0,t) = T(L,t) = 0$, $\forall t$, onde 0 e L representam os pontos extremos da barra.

3.4 O Algoritmo

O algoritmo utilizado para resolver númericamente a equação baseia-se nos dois tópicos discutidos acima. O domínio da função será discretizado, i.e., transformado em um grid com nx pontos no espaço e nt momentos no tempo.

Figura 1: Discretização



Denotamos por Δx a diferença (constante) entre os pontos no espaço e Δt o incremento temporal (note que $\Delta x = \frac{L}{nx-1}$). Além disso, denotaremos

$$x_i = i\Delta x,\ 0 \leq i \leq nx-1$$

os pontos do espaço discretizado e

$$t^n = t_0 + n\Delta t,\ 0 \leq n \leq nt$$

onde t_0 é o instante de tempo inicial. Por fim, escrevemos $T_i^n = T(x_i, t^n)$, sendo estes os pontos sobre os quais trabalharemos.

O método implícito de Euler para a equação diferencial $y' = f(y,t)$ é

$$y^{n+1} = y^n + \Delta t f(y^{n+1}, t)$$

. Em nosso caso, aplicando tal método em junção com as diferenças finitas, temos

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \alpha \frac{T_{i-1}^{n+1} - 2T_i^{n+1} + T_{i+1}^{n+1}}{\Delta x^2} \tag{3}$$

$$\Leftrightarrow T_i^{n+1} = T_i^n + \frac{\alpha \Delta t}{\Delta x^2}(T_{i-1}^{n+1} - 2T_i^{n+1} + T_{i+1}^{n+1}) \tag{4}$$

Uma vez que estamos impondo as condições de contorno de Dirichlet $T_0^m = T_{nx-1}^m \forall m$, podemos considerar a matriz-coluna $T^m = [T_i^m]$, $1 \leq i \leq nx - 2$. A equação (4) nos dá, portanto,

$$T^{n+1} = T^n + AT^{n+1} \iff T^n = (I - A)T^{n+1}$$

onde A é a matriz tridiagonal simétrica de tamanho $nx - 2$ que representa a aproximação da segunda derivada por diferenças finitas (decorrente do segundo membro na equação (3))

$$A = \alpha \frac{\Delta t}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & \dots \\ 1 & -2 & 1 & \dots \\ 0 & 1 & -2 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Temos, finalmente,

$$T^{n+1} = (I - A)^{-1} T^n$$

que nos permite calcular recursivamente o estado previsto pelo modelo em qualquer instante de tempo $t_0 + k\Delta t$.

3.5 Estabilidade

A matriz A descrita anteriormente pertente à classe das matrizes de Toeplitz, caracterizadas por terem diagonais constantes. São matrizes da forma

$$\begin{bmatrix} a & b & 0 & \dots & 0 & 0 & 0 \\ c & a & b & \dots & 0 & 0 & 0 \\ 0 & c & a & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a & b & 0 \\ 0 & 0 & 0 & \dots & c & a & b \\ 0 & 0 & 0 & \dots & 0 & c & a \end{bmatrix}$$

Quando $bc \neq 0$, a matriz acima é diagonalizável. Há, nesse caso, uma fórmula fechada para seus autovalores, que são todos distintos:

$$\lambda_k = a + 2\sqrt{bc} \cdot cos\left(\frac{\pi k}{m+1}\right),\ 0 \leq k \leq m$$

sendo m o tamanho da matriz.

Em nosso caso, temos

$$\lambda_k = \frac{\alpha \Delta t}{\Delta x^2}\left(-2 + 2 \cdot cos\left(\frac{\pi k}{nx-1}\right)\right),\ 0 \leq k \leq nx-2$$

Seja $z^m = (z_1, z_2, \ldots, z_{nx-2})$ vetor das coordenadas de T^m na base dos autovetores. Diagonalizando A, temos

$$(I - \Lambda)z^{n+1} = z^n \iff (I - \Lambda)^{n+1} z^{n+1} = z^0$$

sendo Λ a matriz diagonal dos autovalores.

Neste novo sistema de coordenadas, nosso sistema se reduz a um conjunto de $nx-2$ equações independentes. Temos, assim,

$$z_k^{n+1} = \left(\frac{1}{1-\lambda_k}\right)^{n+1} z_k^0$$

.

Como $\lambda_k < 0$ (já que $cos\left(\frac{\pi k}{nx-1}\right) < 1$ para $1 \leq k \leq nx-2$), temos $\frac{1}{1-\lambda_k} < 1$, de forma que as coordenadas z_k permanecem limitadas. Isto garante que o algoritimo é incondicionalmente estável. Veja também que $\lim_{n\to\infty} z_k^n = z_k^0 \cdot \lim_{n\to\infty}(\frac{1}{1-\lambda_k})^n = 0$, isto é, a função tende a aproximar-se da função nula conforme o tempo passa.

4 Implementação em Python

Começamos importando algumas bibliotecas úteis.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from IPython.display import HTML
```

Aqui temos o código da função que cria a matriz tridiagonal A.

```
from scipy.sparse import diags

def mat_tridiag(nx, dx, dt, alpha):
    """
    Constroi a matriz tridiagonal para calcular a segunda derivada discreta
    Parametros
    ----------
    nx : int
        Numero de pontos na barra
    dx : float
```

```
        Espacamento entre os pontos
    Return : numpy.ndarray
        matriz para computar a segunda derivada discreta
    """
    diagonais = [[1.], [-2.], [1.]]

    offsets = [-1, 0, 1] # diagonais acima, na diagonal, abaixo

    d2mat = diags(diagonais, offsets, shape=(nx-2,nx-2)).toarray() # matriz tridiagonal

    return alpha*dt*d2mat / dx**2 # matriz tridiagonal dividida pelo quadrado do espacamento
```

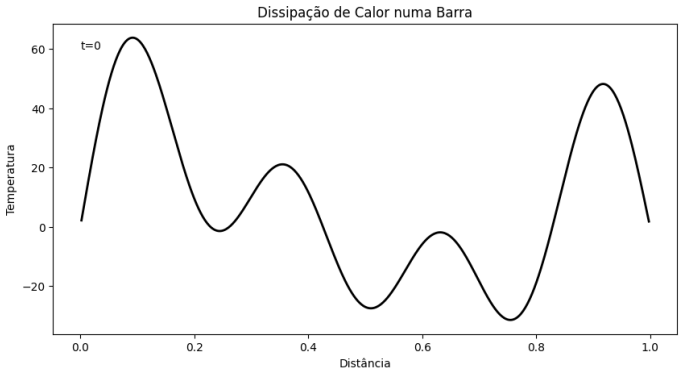
Aqui computamos o array T que contém nas suas linhas a temperatura em cada ponto da barra no tempo t.

```
    # Parametros
alpha = 0.05                    # coeficiente de difusao
lx = 1.                         # tamanho da barra em x
ti = 0.0                        # tempo inicial
tf = 7.0                        # tempo final

# Discretizacao
nx = 513                        # numero de pontos na barra
dx = lx / (nx-1)                # espaco entre os pontos
x = np.linspace(0., lx, nx)     # coordenadas dos pontos

T0 =(20*np.sin(3*np.pi*x) + 25*np.sin(7*np.pi*x)
    + 15*np.sin(2*np.pi*x) + 18*np.sin(5*np.pi*x)) # condicao inicial

fourier = 10                # numero de termos da serie de Fourier
dt = fourier*dx**2/alpha    # passo de tempo
nt = int((tf-ti)/dt)        # numero de passos de tempo

# matriz d^2 / dx^2
A = mat_tridiag(nx, dx, dt, alpha)

# I-A
M = np.eye(nx-2) - A
Minv = np.linalg.inv(M)

T = np.empty((nt+1, nx)) # array para armazenar a solucao
T[0] = T0.copy()         # coloca a condicao inicial no array

T[:,0] = 0
T[:,-1] = 0

for i in range(nt):
    T[i+1, 1:-1] = np.dot(Minv, T[i, 1:-1])
```

Não é díficil ver que no tempo t arbitrariamente grande todo calor irá se dissipar, deixando a temperatura da barra uniformemente igual a 0.

Figura 2: Uma equação inicial qualquer

$$(20\sin(3\pi x) + 25\sin(7\pi x) + 15\sin(2\pi x) + 18\sin(5\pi x))$$



5 Conclusão

Neste trabalho discutimos brevemente a equação do calor, a matématica por trás do método implícito que é um dos muitos métodos numéricos existentes para sua resolução e fizemos a implementação em python, com resultado final uma animação[1] do tempo inicial até o final com a evolução da temperatura através da barra.

6 Referências Bibliográficas

[1] Knaepen, B.; Velizhanina, Y., *Numerical methods for partial differential equations*. GitHub, 2020. Disponível em https://github.com/aquaULB/solving_pde_mooc.

[1] Animação citada