

# Análise Numérica da Equação de Calor - Problema Inverso

Caio Victor Gouveia Freitas e Pier Luigi Nakai Ricchetti

caiofreitas@usp.br pierluigiricchetti@usp.br

*Escola Politécnica da Universidade de São Paulo*

*Instituto de Matemática e Estatística*

*Departamento de Matemática Aplicada*

O presente relatório pertence a um trabalho a ser entregue para a disciplina MAP3121 - Métodos Numéricos e Aplicações - oferecida pelo Instituto de Matemática e Estatística da Universidade de São Paulo - , que tem como motivação física o problema da obtenção das fontes de temperatura aplicadas, no contexto unidimensional ao longo do tempo, dado o estado final. E tem como motivação teórica estudar e desenvolver métodos numéricos para resolução de sistemas de equações sobredeterminados, com o uso do Método dos Mínimos Quadrados.

A linguagem de programação usada para o desenvolvimento das soluções foi o Python 3.6.9 - o programa final pode ser executado em qualquer Python3.x - , em conjunto com as bibliotecas numpy, matplotlib, datetime, time e o módulo random.

<b>Contents</b>	...
<b>I. Acertando o método de Crank-Nicolson</b>	3
A. Preenchimento da matriz	3
B. Equações utilizadas	3
C. Exemplo de resultado da primeira tarefa	3
<b>II. Preparação para o problema inverso</b>	4
A. Produto interno e MMQ	4
B. A matriz de temperaturas em T	4
C. Montagem do problema	4
D. Fatoração $LDL^t$	5
E. Resolução do problema	5
<b>III. Testes realizados e resultados</b>	5
A. Resultados item A	5
B. Resultados item B	6
C. Resultados item C	6
1. N = 128	6
2. N = 256	7
3. N = 512	7
4. N = 1024	8
5. N = 2048	8
D. Resultados item D	9
1. N = 128	9
2. N = 256	10
3. N = 512	10
4. N = 1024	11
5. N = 2048	11
<b>IV. Análises dos resultados</b>	12
<b>V. Considerações finais</b>	12
<b>VI. Código</b>	13

## I. Acertando o método de Crank-Nicolson

Devido à necessidade de se utilizar o método de Crank-Nicolson nesta segunda parte do programa, optou-se por reescrever este método de uma forma mais organizada e bem documentada, uma vez que no relatório anterior foram apontados problemas relacionados à estes tópicos.

Como o foco desta parte do trabalho é a resolução do problema inverso, apenas uma discussão e explicação sucinta será fornecida, assim, preservando o escopo da análise.

### A. Preenchimento da matriz

O método de preenchimento da matriz do problema foi repensado e reprogramado a fim de se manter o maior grau de coerência possível. Para isso, decidiu-se seguir as mesmas notações adotadas pelo enunciado fornecido, ou seja, as linhas da matriz solução deste método variam de  $i = 0, 1, \dots, N$  e as colunas de  $k = 0, 1, \dots, M = N$  de forma que para  $k = 0$  se obtém  $u_0(x)$  (condição inicial) e para  $i = 0$  ou  $i = N$  se obtém  $g_1(t)$  e  $g_2(t)$ , respectivamente.

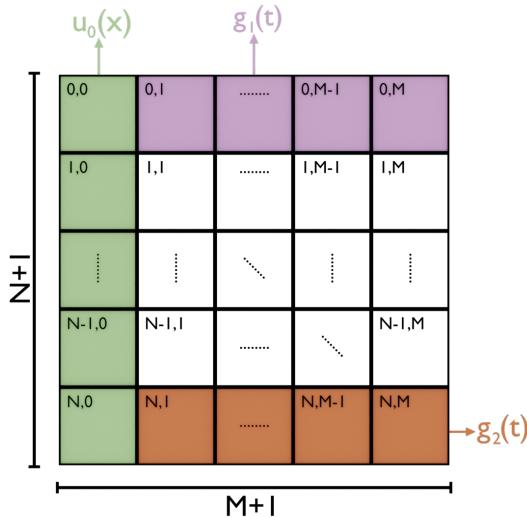


Figure 1: Preenchimento da matriz

### B. Equações utilizadas

Da equação fornecida para o método de Crank-Nicolson, é possível deduzir a seguinte expressão:

$$\begin{aligned} u_i^{k+1} - \frac{\lambda}{2}(u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}) &= u_i^k \\ + \frac{\lambda}{2}(u_{i-1}^k - 2u_i^k + u_{i+1}^k) + \frac{\Delta t}{2}(f(x_i, t_k) + f(x_i, t_{k+1})) & \end{aligned} \quad (1)$$

Variando os parâmetros de  $i$  e  $k$ , obtemos a forma do problema descrito, simplificadamente por  $Ax = b$ , onde temos:

$$\underbrace{\begin{bmatrix} 1 + \lambda & -\frac{\lambda}{2} & 0 & \dots & 0 \\ -\frac{\lambda}{2} & 1 + \lambda & -\frac{\lambda}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{\lambda}{2} & 1 + \lambda & -\frac{\lambda}{2} \\ 0 & \dots & 0 & -\frac{\lambda}{2} & 1 + \lambda \end{bmatrix}}_A \underbrace{\begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ u_{N-1}^{k+1} \\ u_N^{k+1} \end{bmatrix}}_x \quad (2)$$

$$b = \begin{bmatrix} \frac{\Delta t}{2}(f(x_1, t_k) + f(x_1, t_{k+1})) + \frac{\lambda}{2}(g_0(t_{k+1}) + g_0(t^k) + u_2^k) + (1 - \lambda)u_1^k \\ \frac{\Delta t}{2}(f(x_2, t_k) + f(x_2, t_{k+1})) + \frac{\lambda}{2}(u_1^k + u_3^k) + (1 - \lambda)u_2^k \\ \vdots \\ \frac{\Delta t}{2}(f(x_{N-2}, t_k) + f(x_{N-2}, t_{k+1})) + \frac{\lambda}{2}(u_{N-3}^k + u_{N-1}^k) + (1 - \lambda)u_{N-2}^k \\ \frac{\Delta t}{2}(f(x_{N-1}, t_k) + f(x_{N-1}, t_{k+1})) + \frac{\lambda}{2}(g_2(t_{k+1}) + g_2(t^k) + u_{N-2}^k) + (1 - \lambda)u_{N-1}^k \end{bmatrix} \quad (3)$$

### C. Exemplo de resultado da primeira tarefa

Para demonstrar que o método foi aplicado de forma correta, o programa foi rodado para o caso  $N = 320$  e utilizando as equações do item "a" da primeira tarefa, ou seja:

$$\begin{aligned} f(t, x) &= 10\cos(10t)x^2(1-x)^2 - (1 + \sin(10t))(12x^2 - 12x + 2) \\ u_0(x) &= x^2(1-x)^2 \\ g_1(t) &= g_2(t) = 0 \end{aligned}$$

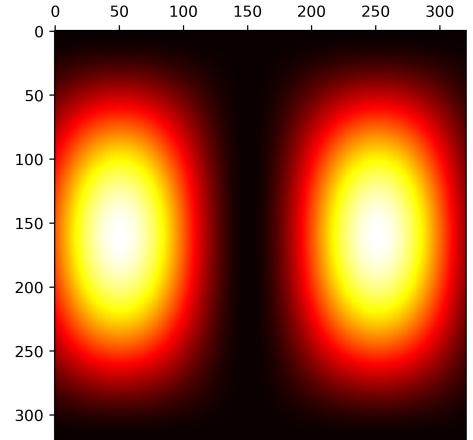


Figure 2: Resultado para o item "a" e  $N=320$

## II. Preparação para o problema inverso

Nesta seção serão descritos os métodos utilizados para a resolução do problema inverso, bem como as suposições e equações utilizadas, deduzidas e adaptadas ao problema dado.

### A. Produto interno e MMQ

A descrição das equações do calor ao longo do tempo pode ser interpretada como um problema típico de malha bidimensional, onde se tem uma resolução dada e expressões que levam em consideração a posição e o tempo para determinar a temperatura em um ponto específico desta malha.

Dadas as temperaturas em um instante  $t$  ao longo de cada  $x_i$  definidos por  $\Delta x \cdot i$ , por conta do tipo de problema tratado, como explicado acima, torna-se necessária uma definição para o produto interno de forma que este atenda à necessidade de minimizar o erro da forma:

$$E_2 = \sqrt{\Delta x \sum_{i=1}^{N-1} (u_T(x_i) - \sum_{k=1}^{nf} a_k u_k(T, x_i))^2} \quad (4)$$

Assim, como descrito no enunciado, este se torna um problema de MMQ, e o produto interno, então, é da forma:

$$\langle u, v \rangle = \sum_{i=1}^{N-1} u(x_i)v(x_i) \quad (5)$$

Dadas duas equações  $u(x_i)$  e  $v(x_i)$  definidas para  $i = 1, 2, \dots, N - 1$ , estas podem ser reescritas na forma de vetores, como se segue:

$$\begin{cases} u(x) = [u(x_1), u(x_2), \dots, u(x_{N-1})] \\ v(x) = [v(x_1), v(x_2), \dots, v(x_{N-1})] \end{cases} \quad (6)$$

Adotando uma notação matricial e fazendo  $u(x) \times v(x)^t$ , obtemos:

$$[u(x_1) \ u(x_2) \ \dots \ u(x_{N-1})] \times \begin{bmatrix} v(x_1) \\ v(x_2) \\ \vdots \\ v(x_{N-1}) \end{bmatrix} \quad (7)$$

$$= u(x_1)v(x_1) + \dots + u(x_{N-1})v(x_{N-1}) \quad (8)$$

É possível notar que a igualdade (8) é equivalente à equação (5) que descreve o produto interno necessário ao problema.

## B. A matriz de temperaturas em T

Para a resolução do problema, seria interessante possuir uma matriz que contivesse os valores de cada  $u_k(T, x)$  arranjados de forma a facilitar a obtenção da matriz normal, necessária para a resolução do problema de MMQ.

Esta matriz foi chamada de "Matriz de temperaturas em T" cuja notação adotada foi  $U_T(x)$  e cujas colunas correspondem às fontes dadas e as linhas as respectivas posições, sendo seus valores àqueles obtidos pelo método de Crank-Nicolson advindos da resolução da equação matricial descrita por (2) e (3) no instante  $t = T$ .

Portanto, seja uma matriz  $U_T(x)$  definida como:

$$U_T(x) = \begin{bmatrix} u_1(T, x_1) & u_2(T, x_1) & \dots & u_{nf}(T, x_1) \\ u_1(T, x_2) & u_2(T, x_2) & \dots & u_{nf}(T, x_2) \\ \vdots & \vdots & \ddots & \vdots \\ u_1(T, x_{N-1}) & u_2(T, x_{N-1}) & \dots & u_{nf}(T, x_{N-1}) \end{bmatrix} \quad (9)$$

O produto  $U_T(x)^t \times U_T(x)$ , pela igualdade obtida de (8), é igual à matriz:

$$N = \begin{bmatrix} \langle u_1, u_1 \rangle & \langle u_1, u_2 \rangle & \dots & \langle u_1, u_{nf} \rangle \\ \langle u_2, u_1 \rangle & \langle u_2, u_2 \rangle & \dots & \langle u_2, u_{nf} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle u_{nf}, u_1 \rangle & \langle u_{nf}, u_2 \rangle & \dots & \langle u_{nf}, u_{nf} \rangle \end{bmatrix} \quad (10)$$

que é chamada de matriz normal.

## C. Montagem do problema

Para a montagem do problema, pode-se considerar uma equação típica  $Nx = b$ , onde  $N$  é a matriz normal (10) e  $x$  e  $b$  da forma:

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{nf} \end{bmatrix} \quad (11)$$

$$b = \begin{bmatrix} \langle u_T, u_1 \rangle \\ \langle u_T, u_2 \rangle \\ \vdots \\ \langle u_T, u_{nf} \rangle \end{bmatrix} \quad (12)$$

A matriz  $b$  (12) pode ser obtida de forma análoga ao método aplicado para a obtenção de (10), ou seja:

$$U_T(x)^t \times u_T(x) = \quad (13)$$

$$\begin{bmatrix} u_1(T, x_1) & u_1(T, x_2) & \dots & u_1(T, x_{N-1}) \\ u_2(T, x_1) & u_2(T, x_2) & \dots & u_2(T, x_{N-1}) \\ \vdots & \vdots & \ddots & \vdots \\ u_{nf}(T, x_1) & u_{nf}(T, x_2) & \dots & u_{nf}(T, x_{N-1}) \end{bmatrix} \times \begin{bmatrix} u_T(x_1) \\ u_T(x_2) \\ \vdots \\ u_T(x_{N-1}) \end{bmatrix} = b \quad (14)$$

se  $u_T(x)$  da forma:

$$\begin{bmatrix} u_T(x_1) \\ u_T(x_2) \\ \vdots \\ u_T(x_{N-1}) \end{bmatrix} \quad (15)$$

Portanto, todos os elementos necessários para a montagem do problema podem ser obtidos, de forma que resta apenas definir uma rotina para o cálculo da fatoração  $LDL^T$  (descrita no próximo tópico) e uma rotina para a obtenção dos valores de (11) (descrita no tópico "E" desta seção).

#### D. Fatoração $LDL^T$

O objetivo da fatoração  $LDL^T$  é transformar uma matriz  $A_{N \times N}$  em um produto  $L_{N \times N} D_{N \times N} L_{N \times N}^T$  com  $L_{N \times N}$  sendo bidiagonal inferior - isto é,  $L_{ij}, j = 0$ , se  $i > j$  ou  $i > j + 1$ . O sistema estará, portanto, escalonado. Trata-se de um algoritmo eficiente para resolver sistemas lineares, porque possui uma complexidade menor do que o cálculo de inversas.

A solução para a fatoração  $LDL^T$  está presente em pseudocódigo no livro *Numerical Analysis - Burden and Faires*

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,N} \\ a_{2,1} & a_{2,2} & \dots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \dots & a_{N,N} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{2,1} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{N,1} & l_{N,2} & \dots & 1 \end{bmatrix}}_{L} \times \underbrace{\begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_N \end{bmatrix}}_D \times \underbrace{\begin{bmatrix} 1 & l_{2,1} & \dots & l_{N,1} \\ 0 & 1 & \dots & l_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}}_{L^T} \quad (16)$$

#### E. Resolução do problema

A partir da decomposição  $LDL^T$ , partimos para a resolução do nosso sistema, que passa a ser

$$A_{N,N} \times x_{N,1} = b_{N,1} \implies L_{N,N} D_{N,N} \underbrace{L_{N,N}^T \times x_{N,1}}_y = b_{N,1} \quad (17)$$

Definidas as variáveis auxiliares  $y = L^T \times x$  e  $z = D \times y$ , temos que, para achar nossa icôgniga  $\mathbf{x}$ , precisamos primeiro resolver o sistema para  $\mathbf{z}$ , então para  $\mathbf{y}$  e afinal para  $\mathbf{x}$ , como ilustrado abaixo

$$L_{N,N} \times \mathbf{z} = b_{N,1} \Rightarrow D_{N,N} \times \mathbf{y} = z_{N,1} \Rightarrow L_{N,N}^T \times \mathbf{x} = y_{N,1} \quad (18)$$

Onde cada um dos sistemas pode ser resolvido com complexidade  $O(n^2)$  ou  $O(n)$  - no caso da matriz diagonal-. Para resolução do sistema  $L \times \mathbf{z} = b$ , temos o seguinte algoritmo

$$\begin{cases} z_1 = b_1 \\ z_2 = b_2 - z_1 L_{2,1} \\ z_3 = b_3 - z_1 L_{3,1} - z_2 L_{3,2} \\ \vdots \\ z_N = b_N - \sum_{i=1}^{N-1} z_i L_{N,i} \end{cases} \quad (19)$$

Uma vez encontrados os valores  $z_i$ , encontramos  $y_i$  com

$$y_i = \frac{z_i}{D_{i,i}} \quad (20)$$

Enfim, encontramos nosso vetor  $\mathbf{x}$  de modo análogo ao encontrado em (21), mas fazendo de uma matriz triangular superior, e não inferior

$$\begin{cases} z_N = y_N \\ z_{N-1} = y_{N-1} - y_N L_{N-1,N}^T \\ \vdots \\ z_1 = y_1 - \sum_{i=1}^{N-1} y_{N-i} L_{1,N-i}^T \end{cases} \quad (21)$$

### III. Testes realizados e resultados

Nesta seção serão apresentados os resultados dos testes pedidos no enunciado deste trabalho.

Além disso, o programa também devolve cada  $a_k$  achado, o valor numérico de cada erro quadrático, uma estimativa de tempo para o processo total e o tempo que o processo levou de fato (no terminal, o programa também imprime a porcentagem de conclusão do processo, no entanto, este fora omitido a fim de se manter a clareza do documento). Estas saídas estão colocadas após as imagens de cada "N" escolhido.

#### A. Resultados item A

Para este item, apenas uma fonte foi fornecida,  $N = 128$  e a solução é trivial, já que, por definição, neste item,  $u_T(x_i) = 7u_1(T, x_i)$ .

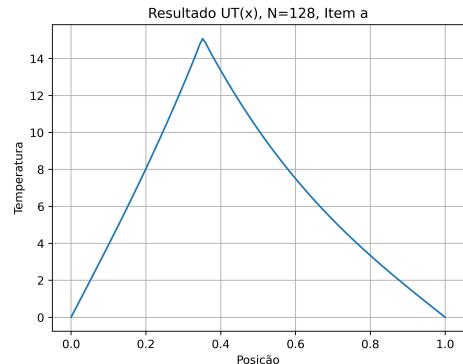


Figure 3: Temperatura em função da posição para  $N=128$

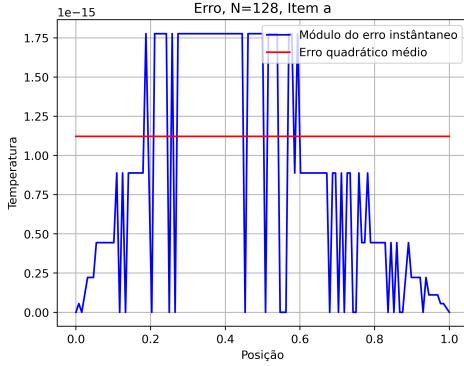


Figure 4: Erros na temperatura em função da posição para N=128

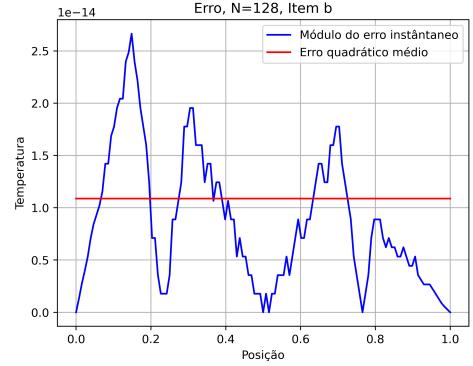


Figure 6: Erros na temperatura em função da posição para N=128

Valores devolvidos pelo programa:

Valores devolvidos pelo programa:

$$\begin{aligned} \text{Estimativa de tempo} &= 0 : 00 : 01.400300 \\ \text{O processo demorou} &= 0 : 00 : 03.008025 \\ a_k &= [6.99999999] \\ \text{Erro quadrático} &= 1.12169157 \cdot 10^{-15} \end{aligned}$$

$$\begin{aligned} \text{Estimativa de tempo} &= 0 : 00 : 06.751200 \\ \text{O processo demorou} &= 0 : 00 : 08.338955 \end{aligned}$$

$$a_k = \begin{bmatrix} 2.3 \\ 3.7 \\ 0.3 \\ 4.2 \end{bmatrix}$$

$$\text{Erro quadrático} = 1.08709073 \cdot 10^{-14}$$

### C. Resultados item C

Para este item e o item d), o programa lê um arquivo chamado "teste.txt", extraído do *edisciplinas* e resolve o problema inverso de acordo com o parâmetro "N" colocado pelo usuário.

#### B. Resultados item B

Neste item, foram fornecidas 4 fontes com  $N = 128$  novamente e a solução também já era conhecida.

##### 1. $N = 128$

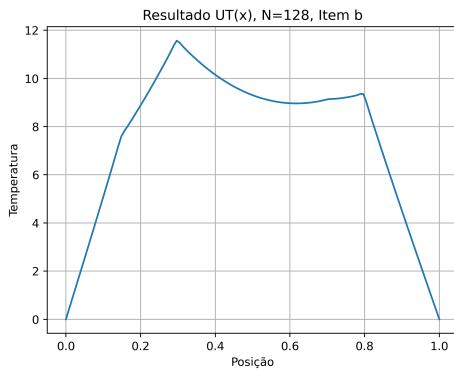


Figure 5: Temperatura em função da posição para N=128

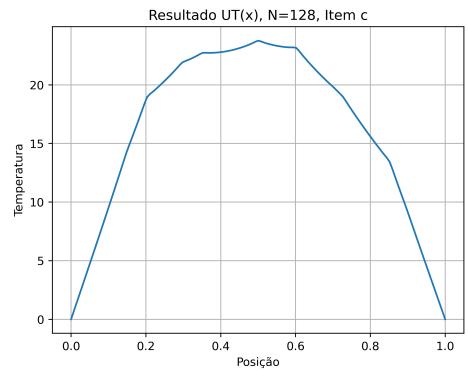
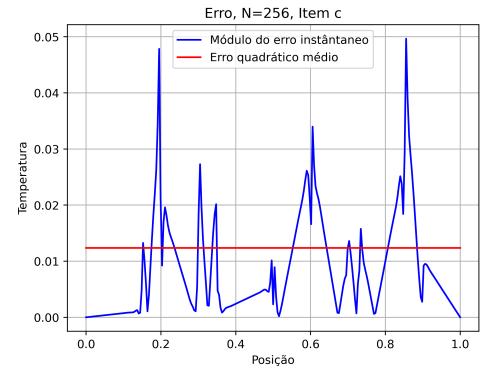
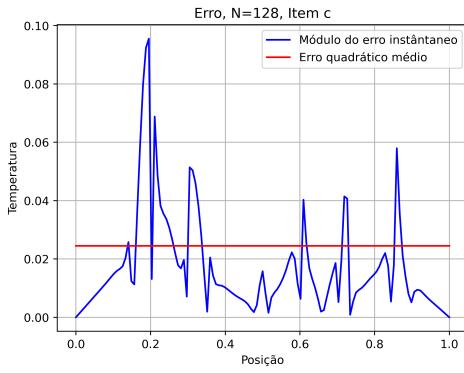


Figure 7: Temperatura em função da posição para N=128



Valores devolvidos pelo programa:

$$\text{Estimativa de tempo} = 0 : 00 : 16.502800$$

$$O processo demorou = 0 : 00 : 18.856604$$

$$a_k = \begin{bmatrix} 1.20912318 \\ 4.83925872 \\ 1.88724086 \\ 1.58339993 \\ 2.21450405 \\ 3.12129478 \\ 0.37734029 \\ 1.49234829 \\ 3.9751388 \\ 0.40414515 \end{bmatrix}$$

$$\text{Erro quadrático} = 0.024453403799693515$$

$$\text{Estimativa de tempo} = 0 : 02 : 09.522700$$

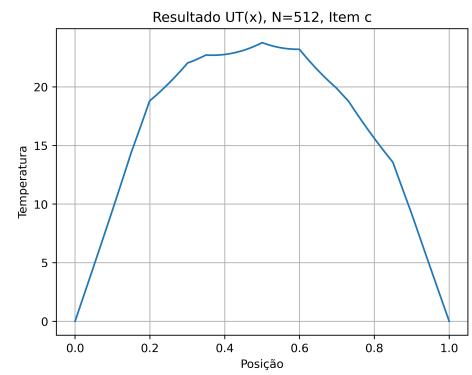
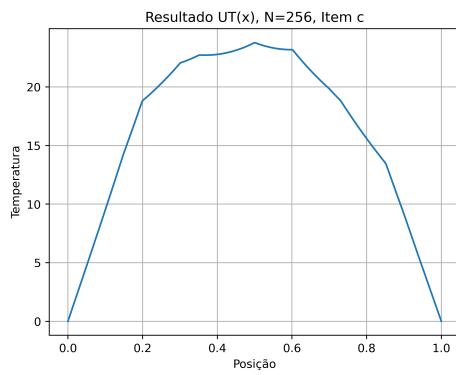
$$O processo demorou = 0 : 02 : 11.812606$$

$$a_k = \begin{bmatrix} 0.90450103 \\ 5.07757264 \\ 2.1008536 \\ 1.41415569 \\ 2.22924501 \\ 3.10461386 \\ 0.5094526 \\ 1.38650879 \\ 3.94987865 \\ 0.41489313 \end{bmatrix}$$

$$\text{Erro quadrático} = 0.012363464048875568$$

2.  $N = 256$

3.  $N = 512$



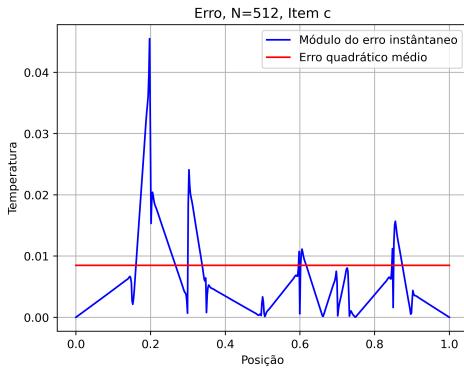


Figure 12: Erros na temperatura em função da posição para N=512

Valores devolvidos pelo programa:

$$\text{Estimativa de tempo} = 0 : 17 : 22.481900$$

$$O processo demorou = 0 : 17 : 04.604402$$

$$a_k = \begin{bmatrix} 0.92868838 \\ 5.05370784 \\ 2.04370105 \\ 1.46767067 \\ 2.19676333 \\ 3.09113117 \\ 0.63758752 \\ 1.27168722 \\ 3.87809487 \\ 0.53055678 \end{bmatrix}$$

$$\text{Erro quadrtico} = 0.008476628330819478$$

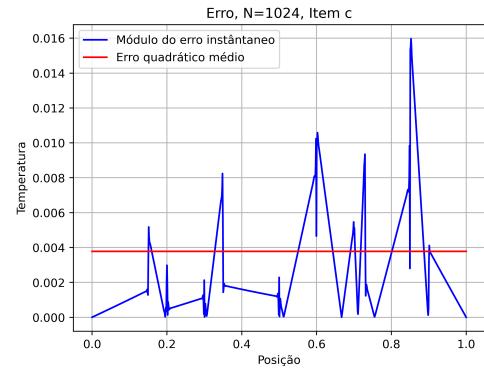


Figure 14: Erros na temperatura em função da posição para N=1024

Valores devolvidos pelo programa:

$$\text{Estimativa de tempo} = 2 : 16 : 57.533700$$

$$O processo demorou = 2 : 15 : 28.854252$$

$$a_k = \begin{bmatrix} 1.00728132 \\ 4.99244301 \\ 1.98587673 \\ 1.51325847 \\ 2.19269284 \\ 3.09515288 \\ 0.65232665 \\ 1.25378989 \\ 3.87966706 \\ 0.52973663 \end{bmatrix}$$

$$\text{Erro quadrtico} = 0.0037793104632986565$$

4. N = 1024

5. N = 2048

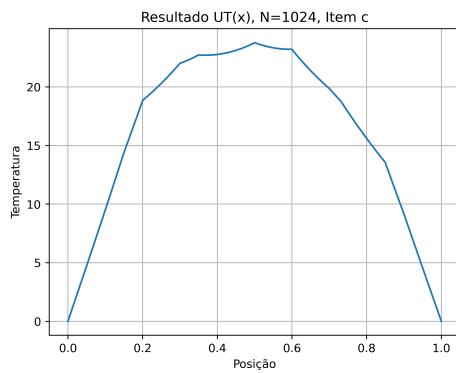


Figure 13: Temperatura em função da posição para N=1024

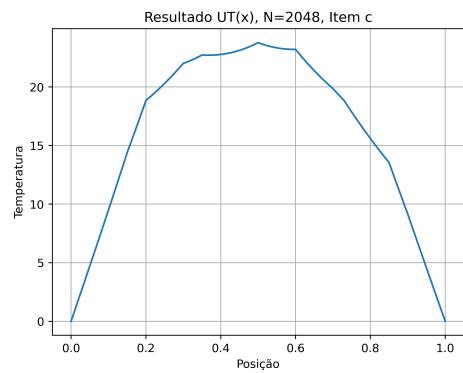


Figure 15: Temperatura em função da posição para N=2048

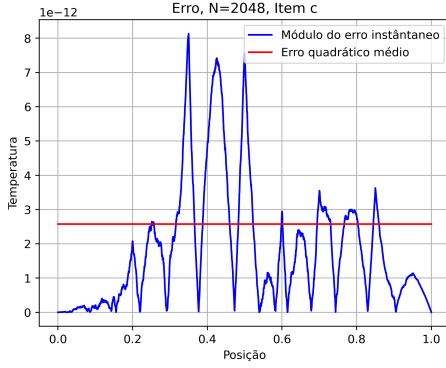


Figure 16: Erros na temperatura em função da posição para N=2048

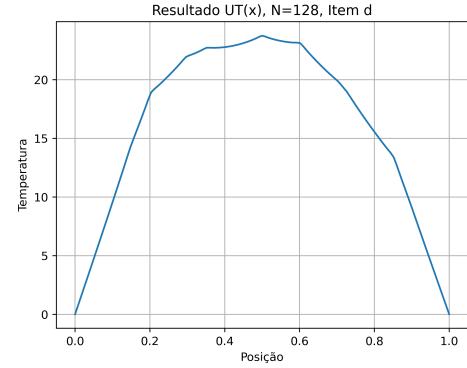


Figure 17: Temperatura em função da posição para N=128

Valores devolvidos pelo programa:

$$\text{Estimativa de tempo} = 18 : 20 : 45.022900$$

$$O processo demorou = 18 : 22 : 15.227700$$

$$a_k = \begin{bmatrix} 1. \\ 5. \\ 2. \\ 1.5 \\ 2.2 \\ 3.1 \\ 0.6 \\ 1.3 \\ 3.9 \\ 0.5 \end{bmatrix}$$

$$\text{Erro quadrático} = 2.5753999631557192.10^{-12}$$

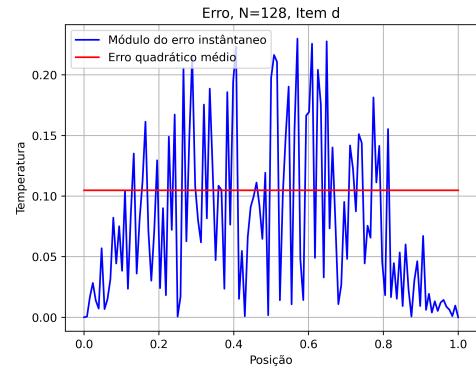


Figure 18: Erros na temperatura em função da posição para N=128

Valores devolvidos pelo programa:

#### D. Resultados item D

Neste item, um ruído foi adicionado nas medições de cada  $u_T(x_i)$ , de forma que:

$$u_T(x_i)' = u_T(x_i)[1 + 0.02(r_i - 0,5)], \quad r_i \in [0.0, 1.0[ \quad (22)$$

onde  $r_i$  é gerado para cada  $i$  e é aleatório no intervalo descrito acima.

$$\text{Estimativa de tempo} = 0 : 00 : 10.702500$$

$$O processo demorou = 0 : 00 : 15.410525$$

$$a_k = \begin{bmatrix} 1.34880004 \\ 4.64450455 \\ 2.05176667 \\ 1.49968481 \\ 2.23141206 \\ 2.98409256 \\ 0.9674803 \\ 1.02430143 \\ 3.97670087 \\ 0.37058409 \end{bmatrix}$$

$$\text{Erro quadrático} = 0.10482872068684804$$

2.  $N = 256$

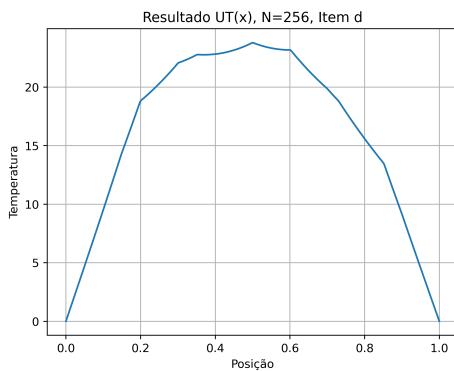


Figure 19: Temperatura em função da posição para  $N=256$

3.  $N = 512$

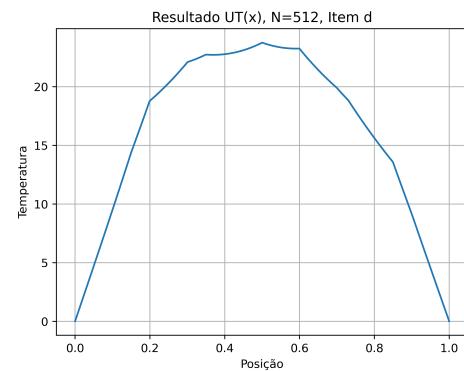


Figure 21: Temperatura em função da posição para  $N=512$

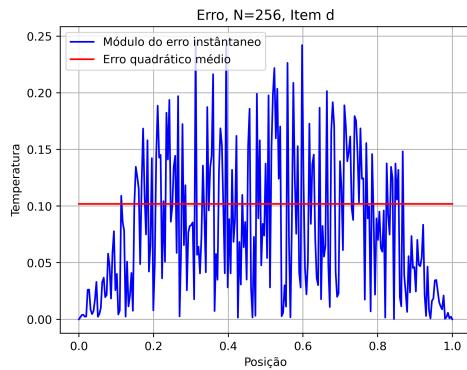


Figure 20: Erros na temperatura em função da posição para  $N=256$

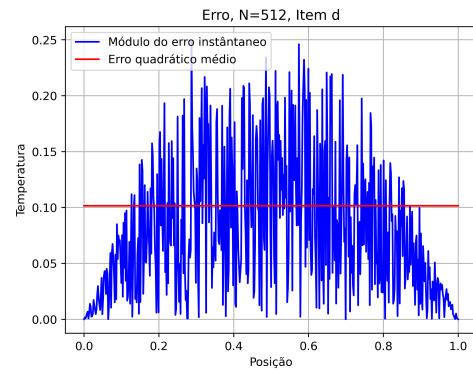


Figure 22: Erros na temperatura em função da posição para  $N=512$

Valores devolvidos pelo programa:

$$\text{Estimativa de tempo} = 0 : 01 : 25.317100$$

$$O processo demorou = 0 : 01 : 50.918412$$

$$a_k = \begin{bmatrix} 0.96852459 \\ 5.00967934 \\ 2.06780444 \\ 1.48812001 \\ 2.22976563 \\ 3.06136316 \\ 0.63735767 \\ 1.26083424 \\ 4.01021261 \\ 0.37175372 \end{bmatrix}$$

$$Erro quadrático = 0.1019080782652278$$

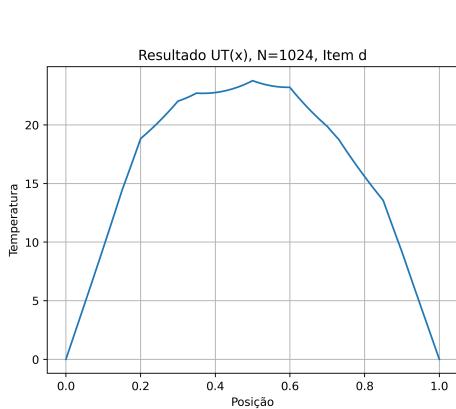
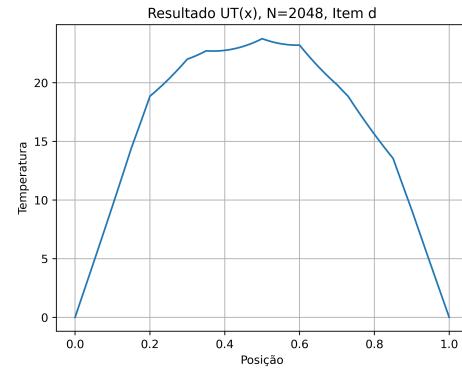
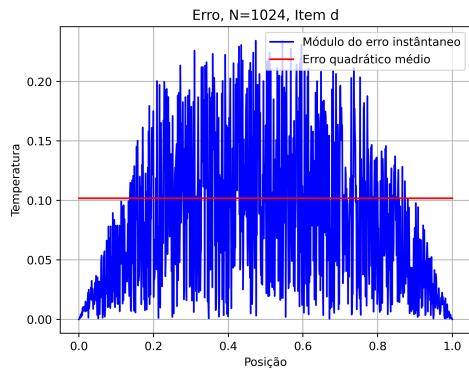
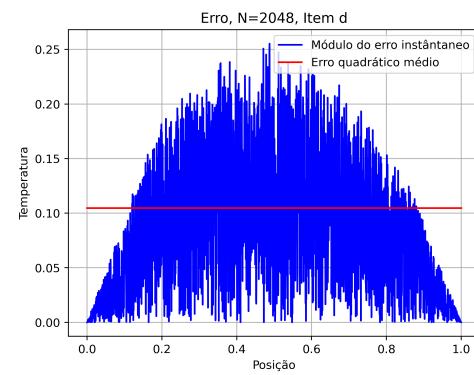
Valores devolvidos pelo programa:

$$\text{Estimativa de tempo} = 0 : 16 : 16.975500$$

$$O processo demorou = 0 : 18 : 12.561626$$

$$a_k = \begin{bmatrix} 0.98496874 \\ 4.92910002 \\ 2.18691505 \\ 1.41427549 \\ 2.12898024 \\ 3.1455941 \\ 0.64460352 \\ 1.2699693 \\ 3.89607826 \\ 0.49769425 \end{bmatrix}$$

$$Erro quadrático = 0.10157969001779545$$

4.  $N = 1024$ 5.  $N = 2048$ Figure 23: Temperatura em função da posição para  $N=1024$ Figure 25: Temperatura em função da posição para  $N=2048$ Figure 24: Erros na temperatura em função da posição para  $N=1024$ Figure 26: Erros na temperatura em função da posição para  $N=2048$ 

Valores devolvidos pelo programa:

*Estimativa de tempo = 16 : 03 : 51.384900**O processo demorou = 13 : 31 : 14.112660*

Valores devolvidos pelo programa:

*Estimativa de tempo = 1 : 43 : 34.734600**O processo demorou = 1 : 44 : 11.822709*

$$a_k = \begin{bmatrix} 1.01426541 \\ 4.9607457 \\ 2.05210024 \\ 1.4614231 \\ 2.20234157 \\ 3.08271436 \\ 0.77538737 \\ 1.11913868 \\ 3.88572519 \\ 0.55102714 \end{bmatrix}$$

*Erro quadrático = 0.10178250022292325*

$$a_k = \begin{bmatrix} 0.98511735 \\ 5.01750402 \\ 2.00745169 \\ 1.49394439 \\ 2.17448183 \\ 3.13243787 \\ 0.51450742 \\ 1.37188367 \\ 3.90672087 \\ 0.49262659 \end{bmatrix}$$

*Erro quadrático = 0.10463582087944213*

#### IV. Análises dos resultados

Os resultados obtidos nos itens a) e b) foram, como esperado, bem precisos, com erros quadráticos na ordem de  $10^{-15}$  e  $10^{-14}$  respectivamente. Este comportamento se deve ao fato de que o número de operações necessárias para se chegar à solução é baixo quando comparado com os testes realizados nos itens subsequentes. Quanto à diferença na ordem de grandeza entre os dois itens, isto se deve ao fato de que a resolução da malha foi mantida em  $N = 128$  e o número de operações, no entanto, aumentou uma vez que, no item b), foram consideradas 3 fontes a mais.

Nos itens c) e d), para uma mesma resolução de malha dos itens anteriores, isto é, para  $N = 128$ , notou-se um aumento significativo no erro quadrático, de  $10^{-15}$  e  $10^{-14}$  para uma ordem de aproximadamente  $10^{-2}$  e  $10^{-1}$ . Como comentado anteriormente, isto se deve ao fato de que a resolução permaneceu a mesma mas o número de fontes aumentou, ou seja, o número de operações necessárias aumentou.

Além disso, percebe-se uma diferença nos erros quadráticos dos itens c) e d), de forma que esta aumenta juntamente com a resolução da malha. A explicação para este fenômeno está na adição do ruído dado pela expressão (22).

Uma análise rápida desta expressão permite concluir que:

$$F_r^{max}(i) = \max[1 + 0.02(r_i - 0,5)] \cong 1.01 \quad (23)$$

onde  $F_r^{max}(i)$  é o fator de ruído máximo que multiplica  $u_T(x_i)$  de acordo com a expressão (22) e o sinal de aproximado é devido ao intervalo de  $r_i$ , já que foi considerado 1.0 como o valor máximo.

Este fator indica que poderá existir uma variação de até  $\pm 1\%$  do valor de cada temperatura dada. De acordo com os dados fornecidos pelo arquivo de texto, essa variação pode adotar um valor máximo de  $\approx 0.24$  em módulo. Essa variação máxima pode ser confirmada através dos gráficos dos erros geradas pelo programa para o item d, de forma que fica evidente que o erro máximo instantâneo sempre está muito próximo de 0.25. Por conta disso, o erro quadrático para os casos do item d) sempre possuem um valor muito próximo entre eles, da ordem de  $\approx 0.10$ , o que, novamente, faz sentido dado (23).

Desta forma, para as resoluções de malha escolhidas, as aproximações não melhoram para o caso com presença de ruído, uma vez que, para a menor resolução considerada, isto é, para  $N = 128$ , o erro quadrático sem ruído (item c) já é menor do que o erro médio obtido para a solução com ruído (item d).

Portanto, conforme a resolução da malha aumenta, o erro do item c tende à zero, enquanto que o erro do item d tende a permanecer em uma mesma média probabilística, dada a equação do ruído (22), de forma que a discrepância entre os dois itens aumenta juntamente com a resolução da malha, como descrito anteriormente.

#### V. Considerações finais

Durante a execução deste trabalho, ficou evidente a importância do conhecimento dos efeitos dos ruídos nas medições. Estes podem representar imprecisões em instrumentos de medição bem como imprecisões humanas no manuseio de equipamentos.

Durante a modelagem de um problema complexo como este, essa pequena imprecisão pode causar uma grande variação em torno da solução procurada, de forma que, como visto, uma variação de apenas 1% nas medições já pode causar um erro quadrático da ordem de  $\approx 0.1$ .

Sabendo desta variação, fica evidente que uma resolução de malha elevada não melhora os resultados obtidos o que leva a uma otimização de tempo muito eficiente, pois, como visto, de  $N = 1024$  para  $N = 2048$  há uma diferença de tempo que pode ultrapassar 14h, ou seja, em outras palavras, um mesmo resultado poderia ser obtido cerca de  $7\times$  mais rápido, se para  $N = 1024$  já fosse a menor resolução precisa. Para o caso tratado neste EP, a diferença de tempo entre uma resolução de malha de  $N = 128$  e outra de  $N = 2048$  permitiria a obtenção do resultado cerca de  $5760\times$  mais rápido, reforçando, assim, a necessidade do conhecimento das imprecisões e de seus efeitos na análise considerada.

## VI. Código

```

import numpy as np
import matplotlib.pyplot as plt
import datetime
import time
import random
,,
Fun o da Fonte
,,

def funcao_fonte(x_pt, t, ponto):
    global item

    if (x_pt >= (ponto - h/2) and x_pt <= (ponto + h/2)):

        r_t = 10*(1 + np.cos(5*t))
        return r_t/delta_x

    else:
        return 0

,,
Condi o inicial (t=0)
,,

def cond_ini (x):
    return 0

,,
Condicoes de contorno
,,

def g1(t): # condi o de contorno , x=0
    return 0

def g2(t): # condicao de contorno , x=1
    return 0

,,
Funcao que gera resultado da decomposicao LDLT
para a matriz
tridiagonal presente no m todo de Crank-
Nicolson
,,
def decomporLDL( ):
    # L e D Vetores
    D = np.zeros(N-1)
    L = np.zeros(N-2)

    # Condicao inicial para prosseguir com os
    # calculos
    D[0] = 1 + 2*
    # C lculo dos valores dos vetores L e D
    for i in range (1, len(D)):
        D[i] = 1+2* - (((- /D[i-1])**2)*D[i
            -1])
,,
for i in range (0, len(L)):
    L[i] = - /D[i]

# Criacao das matrizes L e D a partir dos
vetores
D_matriz = np.zeros((N-1, N-1))
L_matriz = np.zeros((N-1, N-1))

for i in range (0, N-1):
    D_matriz[i][i] = D[i]
    L_matriz[i][i] = 1 # A matriz L
    deve ter 1s na diagonal principal

for i in range (0, N-2):
    L_matriz[i+1][i] = L[i] #Colocando a
    subdiagonal de L

LT_matriz = np.transpose(L_matriz)

return (L_matriz, D_matriz, LT_matriz)

,,
Resolu o de um sistema Ax = b, onde A pode
ser:
- "lower": triangular inferior
- "upper": triangular superior
- "diagonal": diagonal
,,
def solve(A, b, m_type):
    if A.shape[0] != A.shape[1]:
        print("Erro, matriz nao quadrada!")
        return -1
    solution = np.zeros((A.shape[0]))
    if m_type == "lower":
        solution[0] = b[0]/A[0][0]

        for i in range(1, A.shape[0]):
            summ = 0
            for j in range(1, i + 1):
                summ += A[i][i-j]*solution[i-j]
            ]
            solution[i] = (b[i] - summ)

        return solution

    elif m_type == "upper":
        solution[A.shape[0] - 1] = b[A.shape
            [0]-1]/A[A.shape[0] - 1][A.shape
            [0] - 1]
        for i in range(2, A.shape[0]+1):
            j = A.shape[0] - i
            summ = 0
            for k in range(1, i):
                summ += A[j][j+k]*solution[j+k]
            ]
            solution[j] = (b[j] - summ)

        return solution

    elif m_type == "diagonal":
        for i in range(A.shape[0]):
            solution[i] = b[i]/A[i][i]
        return solution

,,
Decomposi o LDLT de uma matriz A gen rica
,,

def decomposeLDLT(A):
    L = np.diag(np.ones(len(A)))

```

```

D = np.diag(np.ones(len(A)))
v = np.ones(len(A))
for i in range(len(A)):
    for j in range(i):
        v[j] = L[i][j]*D[j][j]
    summ = 0
    for k in range(i):
        summ += L[i][k]*v[k]
    D[i][i] = A[i][i] - summ

    for j in range(i+1, len(A)):
        summ = 0
        for k in range(i):
            summ += L[j][k]*v[k]
        L[j][i] = (A[j][i] - summ)/D[i][i]

return L, D
"""

Resolucao Crank-Nicolson
"""

#----- Funcao que resolve por Crank-
#Nicholson -----
def Crank (pontos): # Retorna a matriz com os
    UTk nas colunas

    pixels_tot = (N-1)*(N-1)*len(pontos) + len
    (pontos)*(N-1)
    porc = int(pixels_tot/100)
    cont_porc = 0
    cont_pixels = 0
    Temps_T = np.zeros((N-1, len(pontos)))
    contou = False

    for fonte in range (len(pontos)):

        #----- Matrizes
        #----- iniciais
        #----- 

        matriz_final = np.zeros((N+1,N+1))

        # Problema do tipo Ax = b...
        x = np.zeros ((N-1, 1))
        b = np.zeros ((N-1))
        x_linha = np.zeros ((N-1))

        #----- Condicoes iniciais
        #----- 

        for i in range (N+1):
            matriz_final[i][0] = cond_ini(i*
                delta_x)

        for k in range (1, N+1):
            matriz_final[0][k] = g1(k*delta_t)

        for k in range (1, N+1):
            matriz_final[N][k] = g2(k*delta_t)

        #----- Inicio do calculo
        #----- 

        for k in range (0, N):
            for i in range (1, N):
                if (i==1):
                    b[i-1] = (delta_t/2)*(
                        funcao_fonte(delta_x*i
                        , delta_t*k, pontos[
                        fonte])+funcao_fonte(
                        delta_x*i, delta_t*(k
                        +1), pontos[fonte]))+(
                        lamb/2)*(g1(delta_t*(k
                        +1))+g1(delta_t*k)+
                        matriz_final [2][k]) +
                        (1-lamb)*matriz_final
                        [1][k]
                elif (i==N-1):
                    b[i-1] = (delta_t/2)*(
                        funcao_fonte(delta_x*i
                        , delta_t*k, pontos[
                        fonte])+funcao_fonte(
                        delta_x*i, delta_t*(k
                        +1), pontos[fonte]))+(
                        lamb/2)*(g2(delta_t*(k
                        +1))+g2(delta_t*k)+
                        matriz_final [N-2][k]) +
                        (1-lamb)*
                        matriz_final [i][k]
                else:
                    b[i-1] = (delta_t/2)*(
                        funcao_fonte(delta_x*i
                        , delta_t*k, pontos[
                        fonte])+funcao_fonte(
                        delta_x*i, delta_t*(k
                        +1), pontos[fonte]))+(
                        lamb/2)*(matriz_final [
                        i-1][k] + matriz_final
                        [i+1][k]) + (1-lamb)*
                        matriz_final [i][k]

            cont_pixels += 1

            if (cont_pixels%porc == 0):
                if (contou == False):
                    C = (datetime.
                        datetime.now() -
                        A)*100
                    print("Estimativa
                    de tempo = {}\
                    n".format(C))
                    contou = True
                cont_porc += 1
                print(cont_porc, "%"
                    concluido")

            y = solve(L_matriz, b, "lower")
            z = solve(D_matriz, y, "diagonal")
            x_linha = solve(LT_matriz, z, "
                upper")

            for j in range (0, N-1):
                x[j][0] = x_linha[j]

            for i in range (1, N):
                matriz_final[i][k+1] = x[i
                    -1][0]

        #----- Preenchendo com uT(xi)
        #----- 

```

```

-----
for i in range (1, N):
    Temps_T[i-1][fonte] = matriz_final
    [i][N]

return Temps_T

,,
Montando o problema
,,

def Montar (Temps_T, UTs): # Retorna N e b do
    problema Nx = b, se UTs     um vetor coluna

    Normal = np.transpose(Temps_T).dot(Temps_T)
    )
    b = np.transpose(Temps_T).dot(UTs)

    return (Normal, b)

,,
Leitura do teste.txt
,,

def Ler (nome): # 'points' contem os valores
    dos pontos de aplicacao da fonte e '
    content' os UTs

    file = open(nome, 'r+')
    content = []
    points = []
    cont = 0
    for line in file:
        if (cont == 0):
            points = line.split()
        else:
            content.append(line.strip())
        cont+=1

    file.close()

    for i in range (len(content)):
        content[i] = float(content[i])

    for i in range (len(points)):
        points[i] = float(points[i])

    return (content, points)

,,
Erro quadratico
,,

def Erro_2 (aks, UTs, Temps_T, nf):

    soma_N = 0
    soma_K = 0

    for i in range (0, N-1):
        for k in range (nf):
            soma_K = soma_K + aks[k]*Temps_T[i]
            [k]

    soma_N = soma_N + (UTs[i] - soma_K)**2
    soma_K = 0

    Erro = np.sqrt(delta_x*soma_N)

    return (Erro)

,,
Cria gráficos
,,

def Cria_graficos (aks, UTs, Temps_T, nf,
    Erro_2):

    soma_K = 0
    label_x = np.zeros((N+1))
    resultado = np.zeros((N+1))
    Erro = np.zeros((N+1))
    Erro_2_graf = np.full((N+1), Erro_2)

    for i in range (N+1):
        label_x[i] = delta_x*i

    for i in range (1, N):
        for k in range (nf):
            soma_K = soma_K + aks[k]*Temps_T[i]
            [-1][k]

        resultado[i] = soma_K
        soma_K = 0

    plt.plot(label_x, resultado)
    plt.title("Resultado UT(x), N=" + str(N) +
        ", Item " + item)
    plt.xlabel("Posição")
    plt.ylabel("Temperatura")
    plt.grid(True)
    plt.savefig("Result_item-{}-N-{}.png".
        format(item, str(N)), dpi=400)
    #plt.show()
    plt.close()

    for i in range (1, N):
        Erro[i] = abs(resultado[i]-UTs[i-1])

    plt.plot(label_x, Erro, '-b', label =
        'Máximo do erro instantâneo')
    plt.plot(label_x, Erro_2_graf, '-r', label =
        'Erro quadrático')
    plt.title("Erro, N=" + str(N) + ", Item " +
        item)
    plt.xlabel("Posição")
    plt.ylabel("Temperatura")
    plt.grid(True)
    plt.legend()
    plt.savefig("Erro_item-{}-N-{}.png".format
        (item, str(N)), dpi=400)
    #plt.show()
    plt.close()

,,
Itens Pedidos
,,

item = input("Digite o item a ser resolvido (a
, b, c, d): ")

if (item == 'a'):
```

```

#-----
#----- Parametros iniciais -----
#-----

N = 128
M = N
lamb = N
delta_t = 1/N
delta_x = 1/N
L_matriz, D_matriz, LT_matriz =
    decomporLDL(lamb/2)
h = delta_x
Pontos = np.array([0.35])
UTs = np.zeros((N-1, 1))
A = datetime.datetime.now()

#----- Inicio da resolucao -----
#-----



Temps_T = Crank(Pontos)

for i in range (1, N):
    UTs[i-1][0] = 7*Temps_T[i-1][0]

Normal, b = Montar(Temps_T, UTs)
NL, ND = decomposeLDLt(Normal)
y = solve(NL, b, "lower")
z = solve(ND, y, "diagonal")
aks = solve(NL.T, z, "upper")
Erro = Erro_2(aks, UTs, Temps_T, len(
    Pontos))

print("Os valores 'ak' sao: ")
print(aks)
print("Com decomposicao LDLt: {}".format(
    aks))
print("O erro quadratico eh: ", Erro)

Cria_graficos(aks, UTs, Temps_T, len(
    Pontos), Erro)
print("O processo demorou ", datetime.
    datetime.now()-A)

elif (item == 'c'):

#----- Parametros iniciais -----
#-----



N = int(input("Digite o valor de N: "))
M = N
lamb = N
delta_t = 1/N
delta_x = 1/N
L_matriz, D_matriz, LT_matriz =
    decomporLDL(lamb/2)
h = delta_x
todos_UTs, Pontos = Ler("teste.txt")
UTs = []
A = datetime.datetime.now()

#----- Inicio da resolucao -----
#-----



fator = 2048/N
cont = 1

for i in range (1, 2048):
    if (i*fator == 0):
        UTs.append(todos_UTs[i])
    cont+=1

Temps_T = Crank(Pontos)
Normal, B = Montar(Temps_T, UTs)

NL, ND = decomposeLDLt(Normal)
y = solve(NL, B, "lower")
z = solve(ND, y, "diagonal")
aks = solve(NL.T, z, "upper")
#aks = np.linalg.inv(Normal).dot(B)
Erro = Erro_2(aks, UTs, Temps_T, len(
    Pontos))

```

```

    pontos))

print("Os valores 'ak' s o: ")
print(aks)
print("O erro quadratico eh: ", Erro)

Cria_graficos(aks, UTs, Temps_T, len(
    pontos), Erro)
print("O processo demorou ", datetime.
    datetime.now()-A)

elif (item == 'd'):

#-----
    Parametros iniciais
-----
#-----


N = int(input("Digite o valor de N: "))
M = N
lamb = N
delta_t = 1/N
delta_x = 1/N
L_matriz, D_matriz, LT_matriz =
    decomporLDL(lamb/2)
h = delta_x
todos_UTs, pontos = Ler("teste.txt")
UTs = []
A = datetime.datetime.now()

#-----
    Inicio
    da resolucao
-----
#-----


fator = 2048/N
cont = 1

for i in range (1, 2048):
    if (i%fator == 0):
        UTs.append(todos_UTs[i]*(1 + 2*(random.random() -0.5)*0.01))
    cont+=1

Temps_T = Crank(pontos)
Normal, B = Montar(Temps_T, UTs)

NL, ND = decomposeLDLt(Normal)
y = solve(NL, B, "lower")
z = solve(ND, y, "diagonal")
aks = solve(NL.T, z, "upper")
Erro = Erro_2(aks, UTs, Temps_T, len(
    pontos))

print("Os valores 'ak' s o: ")
print(aks)
print("O erro quadratico eh: ", Erro)

Cria_graficos(aks, UTs, Temps_T, len(
    pontos), Erro)
print("O processo demorou ", datetime.
    datetime.now()-A)

else:
    print("Voce digitou um item n o v lido")

```