Cálculo Numérico - Relatório EP2

João Rodrigo Windisch Olenscki
NUSP 10773224

Luca Rodrigues Miguel
NUSP 10705655

Julho, 2020

Conteúdo

1 Introdução

Este exercício se propõe a resolver o problema inverso para a obtenção de uma distribuição de temperaturas a partir da equação do calor, aplicando para tanto o Método de Crank-Nicolson e o Método dos Mínimos Quadrados. A metodologia da solução está ilustrada na Seção 2, e os conceitos necessários para compreendê-la são expostos na Seção 3, "Desenvolvimento Matemático". Há uma explicação sobre a estruturação do código bem como convenções aplicadas na Seção 4, enquanto que a análise das Tarefas e Testes solicitados encontram-se, respectivamente, nas Seções 5 e 6. Por fim, as conclusões e comentários finais são expostos na Seção 7. Há, ainda, um apêndice em anexo contendo os resultados de todos os testes requisitados.

2 Metodologia

Inicialmente, é dado um conjunto de pontos em cada qual há uma fonte pontual, cuja função é dada, juntamente com as condições iniciais da barra. A partir destas informações, utilizando o método de Crank-Nicolson, é possível determinar a distribuição das temperaturas ao longo da barra. Feito isso, monta-se a matriz e o sistema normal, a serem resolvidos pelo Método dos Mínimos Quadrados. Pela linearidade das equações, a solução será uma combinação linear das soluções caso cada fonte pontual fosse única. Aproveitando-se o fato de que a matriz quadrada do sistema normal é simétrica, resolve-se o sistema linear por meio da Decomposição de Cholesky, e, a partir da matriz decomposta, torna-se fácil resolver o sistema linear de forma a encontrar os coeficientes de intensidade de cada fonte pontual.

3 Desenvolvimento matemático

3.1 Método de Crank-Nicolson

Como já explicado anteriormente, neste exercício-programa o Método de Crank-Nicolson será utilizado largamente para auxiliar a resolução do problema inverso. Para tanto, é necessário definí-lo:

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

A equação anterior também pode ser re-escrita da seguinte forma:

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

E, com esta equação (2), é possível observar o caráter matricial do método. Assim, segue que:

$$\begin{bmatrix} 1+\lambda & -\frac{\lambda}{2} & 0 & \dots & 0 & 0 \\ -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} & \dots & 0 & 0 \\ 0 & -\frac{\lambda}{2} & 1+\lambda & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1+\lambda & -\frac{\lambda}{2} \\ 0 & 0 & 0 & \dots & -\frac{\lambda}{2} & 1+\lambda \end{bmatrix} \cdot \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ u_3^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} =$$

$$\begin{bmatrix} (1-\lambda)u_1^k + \frac{\lambda}{2}u_2^k + \frac{\lambda}{2}(g_1(t_k) + g_1(t_{k+1})) + \frac{\Delta t}{2}(f(x_1, t_k) + f(x_1, t_{k+1})) \\ \frac{\lambda}{2}u_1^k + (1-\lambda)u_2^k + \frac{\lambda}{2}u_3^k + \frac{\Delta t}{2}(f(x_2, t_k) + f(x_2, t_{k+1})) \\ \frac{\lambda}{2}u_2^k + (1-\lambda)u_3^k + \frac{\lambda}{2}u_4^k + \frac{\Delta t}{2}(f(x_3, t_k) + f(x_3, t_{k+1})) \\ \vdots \\ \frac{\lambda}{2}u_{N-3}^k + (1-\lambda)u_{N-2}^k + \frac{\lambda}{2}u_{N-1}^k + \frac{\Delta t}{2}(f(x_{N-2}, t_k) + f(x_{N-2}, t_{k+1})) \\ \frac{\lambda}{2}u_{N-2}^k + (1-\lambda)u_{N-1}^k + \frac{\lambda}{2}(g_2(t_k) + g_2(t_{k+1})) + \frac{\Delta t}{2}(f(x_{N-1}, t_k) + f(x_{N-1}, t_{k+1})) \end{bmatrix} \tag{3}$$

É possível aplicar algumas simplificações ao sistema linear, dado que $M = N$ e que, no exercício proposto é requisitado manter as condições de contorno nulas, ou seja, $g_1(t) = g_2(t) = 0$:

$$\begin{bmatrix} 1+N & -\frac{N}{2} & 0 & \dots & 0 & 0 \\ -\frac{N}{2} & 1+N & -\frac{N}{2} & \dots & 0 & 0 \\ 0 & -\frac{N}{2} & 1+N & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1+N & -\frac{N}{2} \\ 0 & 0 & 0 & \dots & -\frac{N}{2} & 1+N \end{bmatrix} \cdot \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ u_3^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} =$$

$$\begin{bmatrix} (1-N)u_1^k + \frac{N}{2}u_2^k + \frac{\Delta t}{2}(f(x_1,t_k) + f(x_1,t_{k+1})) \\ \frac{N}{2}u_1^k + (1-N)u_2^k + \frac{N}{2}u_3^k + \frac{\Delta t}{2}(f(x_2,t_k) + f(x_2,t_{k+1})) \\ \frac{N}{2}u_2^k + (1-N)u_3^k + \frac{N}{2}u_4^k + \frac{\Delta t}{2}(f(x_3,t_k) + f(x_3,t_{k+1})) \\ \vdots \\ \frac{N}{2}u_{N-3}^k + (1-N)u_{N-2}^k + \frac{N}{2}u_{N-1}^k + \frac{\Delta t}{2}(f(x_{N-2},t_k) + f(x_{N-2},t_{k+1})) \\ \frac{N}{2}u_{N-2}^k + (1-N)u_{N-1}^k + \frac{\Delta t}{2}(f(x_{N-1},t_k) + f(x_{N-1},t_{k+1})) \end{bmatrix} \tag{4}$$

Para cada passo realizado em k é necessário calcular a solução do sistema linear da Equação (4).

3.2 Decomposição de Cholesky

O enunciado do exercício constata que a resolução do Método de Mínimos Quadrados segue da resolução do sistema normal associado a matriz quadrada **A** dos produtos internos entre os vetores $u_k(T)$, $k = 1, ., nf$. Para a resolução deste sistema matricial é necessário que realizar uma decomposição $\mathbf{A} = \mathbf{L}.\mathbf{D}.\mathbf{L^T}$, com **L** sendo uma matriz triangular com a parte inferior preenchida e **D** uma matriz diagonal. Defini-se a matriz **A**, onde a_{ij} representa $< u_i, u_j >$:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1nf} \\ a_{21} & a_{22} & \dots & a_{2nf} \\ a_{31} & a_{32} & \dots & a_{3nf} \\ \vdots & \vdots & \ddots & \vdots \\ a_{nf1} & a_{nf2} & \dots & a_{nfnf} \end{bmatrix}$$

$$L.D.L^T = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ l_{31} & l_{32} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{nf1} & l_{nf2} & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_{nf} \end{bmatrix} \cdot \begin{bmatrix} 1 & l_{21} & \dots & l_{nf1} \\ 0 & 1 & \dots & l_{nf2} \\ 0 & 0 & \dots & l_{nf3} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

$$L.D.L^T = \begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ l_{31} & l_{32} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{nf1} & l_{nf2} & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} d_1 & d_1.l_{21} & \dots & d_1.l_{nf1} \\ 0 & d_2 & \dots & d_2.l_{nf2} \\ 0 & 0 & \dots & d_3.l_{nf3} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_{nf} \end{bmatrix} \tag{5}$$

$$L.D.L^T = \begin{bmatrix} d_1 & l_{21}d_1 & \dots & l_{n_f1}d_1 \\ l_{21}d_1 & l_{21}^2 d_1 + d_2 & \dots & l_{21}l_{n_f1}d_1 + l_{n_f2}d_2 \\ \vdots & \vdots & \ddots & \vdots \\ l_{nf_1}d_1 & l_{21}l_{n_f1}d_1 + l_{n_f2}d_2 & \dots & l_{n_f1}^2 d_1 + l_{n_f2}^2 d_2 + \ldots + d_{n_f} \end{bmatrix} \tag{6}$$

Assim, pode-se fazer $\mathbf{A} = \mathbf{L}.\mathbf{D}.\mathbf{L^T}$ e, portanto:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1nf} \\ a_{21} & a_{22} & \dots & a_{2nf} \\ \vdots & \vdots & \ddots & \vdots \\ a_{nf1} & a_{nf2} & \dots & a_{nfnf} \end{bmatrix} = \begin{bmatrix} d_1 & l_{21}d_1 & \dots & l_{n_f1}d_1 \\ l_{21}d_1 & l_{21}^2d_1 + d_2 & \dots & l_{21}l_{n_f1}d_2 \\ \vdots & \vdots & \ddots & \vdots \\ l_{nf_1}d_1 & l_{21}l_{n_f1} + l_{n_f2}d_2 & \dots & l_{n_f1}^2d_1 + l_{n_f2}^2d_2 + \ldots + d_{n_f} \end{bmatrix}$$

Igualando termo a termo e resolvendo o sistema consequente, encontra-se as expressões para d_i e l_{ij}:

$$d_j = a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_k \tag{7}$$

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_k l_{jk} \right) \times \left(\frac{1}{d_j} \right) \tag{8}$$

Assim, pode-se considerar que a decomposição está bem definida e pode ser facilmente implementada no código. Para encontrar a solução do sistema, deve-se resolver $[L][D][L]^T[x] = [b]$. Primeiro, faz-se $[L]^T[x] = [y]$, ou seja:

$$\begin{bmatrix} 1 & l_{21} & \dots & l_{nf1} \\ 0 & 1 & \dots & l_{nf2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{nf} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{nf} \end{bmatrix} \tag{9}$$

A solução deste sistema é:

$$x_i = y_i - \sum_{k=i+1}^{nf} l_{ki} x_k \tag{10}$$

Depois, a substituição $[D][L]^T[x] = [D][y] = [z]$ é feita. Isto implica que:

$$\begin{bmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_{nf} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{nf} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{nf} \end{bmatrix} \tag{11}$$

Resolvendo em y:

$$y_i = \frac{z_i}{d_i} \tag{12}$$

Por fim, é feita a substituição $[L][D][L]^T = [L][z] = [b]$, ou seja:

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{nf1} & l_{nf2} & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{nf} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{nf} \end{bmatrix} \tag{13}$$

A solução deste novo sistema é, por fim:

$$z_i = b_i - \sum_{k=1}^{i-1} l_{ik} z_k \tag{14}$$

Para encontrar os valores de $[x]$, calcula-se primeiro o vetor $[z]$, seguido do $[y]$ para, por fim, ser possível calcular $[x]$. Seguindo este procedimento, é possível encontrar os coeficientes de a_i pedidos no enunciado.

3.3 Método dos Mínimos Quadrados

No problema em questão deseja-se aproximar u_T a partir de uma sequência de vetores $[u_1, u_2, ..., u_{nf}]$. Para isso, pode-se aplicar o Método dos Mínimos Quadrados. Sendo F um espaço vetorial com produto interno e $f \in F$, deseja-se aproximar f por um conjunto de funções $g_1, g_2, ..., g_{nf}$ que geram um subespaço vetorial G de F. Seja g um elemento de G que tenta aproximar f. O erro quadrático da aproximação é dado por:

$$E(f, g) = \langle f - g, f - g \rangle \tag{15}$$

O elemento g é a melhor aproximação possível quando o vetor $f - g$ é ortogonal a G (pois implica que não há qualquer modificação possível em g que o aproximaria mais de f). Isso implica que, $\forall i | 1 \leq i \leq nf$, vale que $\langle f - g, g_i \rangle = 0$. Mas como $g \in G$, existe uma combinação linear de g_i tal que:

$$g = a_1 g_1 + a_2 g_2 + \cdots + a_{nf} g_{nf} \tag{16}$$

Para encontrarmos os valores de a_i, devemos resolver o seguinte sistema:

$$\begin{bmatrix} \langle g_0, g_0 \rangle & \langle g_0, g_1 \rangle & \dots & \langle g_0, g_{nf} \rangle \\ \langle g_1, g_0 \rangle & \langle g_1, g_1 \rangle & \dots & \langle g_1, g_{nf} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle g_{nf}, g_0 \rangle & \langle g_{nf}, g_1 \rangle & \dots & \langle g_{nf}, g_{nf} \rangle \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{nf} \end{bmatrix} = \begin{bmatrix} \langle f, g_0 \rangle \\ \langle f, g_1 \rangle \\ \vdots \\ \langle f, g_{nf} \rangle \end{bmatrix} \tag{17}$$

Resolvido este sistema, encontra-se a melhor aproximação de f a partir de $g_1, g_2, ..., g_{nf}$. O método descrito acima é o método dos mínimos quadrados, e será utilizado para aproximar u_T a partir de $u_1, u_2, ..., u_{nf}$.

4 Estruturação do Código

O código foi escrito em `Python` e seguiu a metodologia procedural: criou-se várias funções individuais para realizar pequenas tarefas e algumas funções principais agregadoras, para executar as individuais, de forma a facilitar sua manutenção. O código é acompanhado por um arquivo `LEIAME.txt` que contém instruções para seu uso e também outras informações. Além disso, o código foi ricamente comentado: utilizou-se para isto a biblioteca, nativa do Python, `typing` para gerar `type hints` que ajudam a entender o funcionamento das funções. Ademais procurou-se, sempre que possível, seguir as normas estabelecidas pela PEP-8 [2]. Um exemplo de uma declaração de função seria a seguinte:

```
def foo(
        arg1: str,
        arg2: int,
        arg3: Dict = {},
    ) -> Tuple[np.ndarray, np.ndarray]:
    '''
```

```
    uma breve explicação sobre o que a função faz
    @parameters:
    - arg1: explicação
    - arg2: explicação
    - arg3: explicação
    @output:
    - out1: dim = (N x N), explicação
    - out2: dim = (1 x N), explicação
    '''
    #código
```

Esta sintaxe para declarar funções com muitas variáveis é a mais recomendada e provê clareza nas entradas e saídas do código. Cada `input` é declarado separadamente seguido de seu tipo e o `output` também é declarado na definição da função. Quem lê sabe o que entra e o quê sai dela. Além disso, esta definição é seguida por uma `string` de comentário que explica a função e também os parâmetros de entrada e saída. Variáveis do tipo `array` são explicadas e suas dimensões também são definidas.

Quando a função `main` é executada, o programa requer do usuário qual teste deverá ser rodado e, nos casos c) e d), pede também o valor de N desejado. A função `main` chama então a função `run_test` e passa o nome do teste e o valor de N como parâmetros. Esta nova função possui 4 diferentes *scripts* para os 4 testes.

Cada um desses *scripts* segue o seguinte padrão: primeiramente define-se, por `input` ou acessando o arquivo `test.txt` a lista que contém a posição das forçantes $[p_1, ., p_{nf}]$. Em seguida encontra-se, via equação pre-definida, nos casos a) e b), ou também através do arquivo de testes o valor de u_T. Nos testes c) e d) é necessário encontrar os vetores u_k, então é chamada a função `generate_nf_vectors`.

Depois chama-se a função `generate_normal_system`, que gera as matrizes $\mathbf{A}$ e $\mathbf{b}$ do sistema normal do Método dos Mínimos Quadrados. Por fim encontra-se a resposta do sistema normal por meio da função `solve_linear_system` e, para os casos c) e d), calcula-se o erro quadrático associado a solução com a função `calculate_quadratic_error`.

Existem, ainda, diversas funções auxiliares para que o código inteiro consiga ser executado, dentre elas as funções de `plot` e algumas funções de *script* responsáveis pelo chamamento de outras funções.

5 Tarefas

5.1 a)

A Tarefa A requisita a elaboração de um código que, dado um certo conjunto $p_k, k = 1, ., nf$ de pontos, gere o conjunto de vetores u_k resolvendo a equação do calor com condições de contorno nulas e uma forçante do tipo $f(t, x) = r(t)g_h^k(x)$, com $r(t) = 10(1 + \cos(5t))$ e $g_h^k(x)$ um degrau aplicado ao ponto p_k, utilizando, além do mais, o Método de Crank-Nicolson. Escreveu-se, então para a resolução desta tarefa as seguintes funções:

```
def get_u_p(N, T, p):
    M = get_M_parameter(T, N)
    time_array = get_time_array(M, T)
    space_array = get_space_array(N)

    u = np.zeros(N+1)
    u = crank_nicolson(T, M, u, space_array, p)
    u = u[1:-1]
    return u

def generate_unf_vectors(N, T, p_list):
    nf = len(p_list)
    u_vectors = []
    for i in range(nf):
```

```
        ui = get_u_p(N, T, p_list[i])
        u_vectors.append(ui)
    return u_vectors
```

A primeira função é uma função auxiliar que, para cada valor de p_k calcula o vetor u_k em $T = 1$ a partir de condições iniciais e fronteiras nulas. A segunda função recebe quase os mesmos parâmetros da primeira, com a diferença de que p_k foi substituído por uma lista $[p_1, ., p_{nf}]$. Então, para cada p_k nesta lista, calcula-se u_k utilizando a primeira função e então agregam-se todos estes vetores na lista `u_vectors` = $[u_1, ., u_{nf}]$.

5.2 b)

A Tarefa B exige que, dado um vetor $u_T(x_i), i = 1, ., N-1$ que representa a solução da equação de calor, construa-se o sistema normal dado pela equação (17). Para a resolução desta tarefa têm-se a função `create_normal_system`:

```
def create_normal_system(u_vectors, u_T):
    nf = len(u_vectors)
    A = np.zeros([nf, nf])
    b = np.zeros(nf)
    for i in range(nf):
        b[i] = np.inner(u_vectors[i], u_T)
        for j in range(nf):
            inner_product = np.inner(u_vectors[i], u_vectors[j])
            A[i][j] = inner_product
            A[j][i] = inner_product

    return A, b
```

Este código recebe a lista de vetores $[u_1, ., u_{nf}]$ gerados na tarefa anterior e o vetor u_T e calcula a matriz quadrada $\mathbf{A}$ e a matriz de resultados $\mathbf{b}$.

5.3 c)

A Tarefa C pede uma rotina para que se decomponha a matriz $\mathbf{A}$ em $\mathbf{L}.\mathbf{D}.\mathbf{L^T}$ e que, dada esta decomposição, desenvolva-se outra rotina que resolva o sistema linear do tipo $\mathbf{A}.\mathbf{x} = \mathbf{b}$, ambas baseadas em [1, p. 422]:

```
def perform_ldlt_transformation(A):
    nf = A.shape[0] # registra-se a dimensão da matriz A
    L = np.eye(nf) # inicia-se a matriz L com a diagonal preenchida por 1s
    D = np.zeros([nf, nf]) # inicia-se a matriz D com zeros
    for i in range(nf): # para i = 0,..., nf - 1
        D[i, i] = A[i, i]
        for j in range(i): # para j = 0,..., i - 1
            L[i, j] = A[i, j]
            for k in range(j): # para k = 0,..., j - 1
                L[i, j] -= L[i, k]*L[j, k]*D[k, k]
            L[i, j] /= D[j, j]
            D[i, i] -= L[i, j]*L[i, j]*D[j, j]
    return L, D
```

Observe que foi possível implementar o código requerido com o mínimo de iterações possíveis, aproveitando-se o máximo de cada iteração para executar partes diferentes do algoritmo (i.e. calcular L e D por partes e simultaneamente). Em seguida, elaborou-se a rotina de resolução do sistema linear através da função `solve_linear_system`:

```
def solve_linear_system(A, u):
    nf = u.shape[0] # determina-se dimensao nf
    L, D = perform_ldlt_transformation(A) # aplica-se a decomp A = L.D.L'

    z = np.zeros(nf) # inicia-se a matriz z com zeros
    for i in range(nf): # para i = 0, ..., nf - 1
        bckwrd_sum = 0 # cria-se uma variável auxiliar para guardar o somatório em k
        for k in range(i): para k = 0, ..., i - 1
            bckwrd_sum += L[i, k]*z[k]
        z[i] = u[i] - bckwrd_sum

    y = np.zeros(nf) # inicia-se y com zeros
    for i in range(nf): # para i = 0, ..., nf
        y[i] = z[i]/D[i,i]

    x = np.zeros(nf) # inicia-se x com zeros
    for i in range(nf-1, -1, -1): # para i = nf - 1, ..., 0
        bckwrd_sum = 0 # cria-se uma variável auxiliar para guardar o valor do somatório em k
        for k in range(i+1, nf): # para k = i + 1, ..., nf - 1
            bckwrd_sum += L[k, i]*x[k]
        x[i] = y[i] - bckwrd_sum
    return x
```

6 Testes

O enunciado pede que execute-se 4 testes, os dois primeiros para a conferência das funções implementadas e com resultados triviais e os outros dois que executavam o código e de fato resolviam o problema inverso da distribuição de temperaturas. Os valores numéricos das intensidades a_i e do erro quadrático E_2 para cada teste podem ser encontrados no Apêndice A.

6.1 a)

O teste a tinha como único objetivo testar as funções implementadas requeria a resolução de um sistema normal de dimensões 1×1, definindo, ainda, $u_T = 7u_1$. O código, para este teste, retornou o resultado esperado, $a_1 = 6.999999999999999$, que é a resposta trivial para a aproximação $u_T = a_1 u_1$, dado que iniciamos com uma restrição linear entre u_T e a_1. Além disto, a pequena diferença observada nos resultados é imputável aos erros de arredondamento do Python.

6.2 b)

Novamente, o teste visa checar o funcionamento das funções implementadas, porém desta vez considerando mais de uma fonte pontual, cada uma com uma posição diferente. Assim como no teste a), os resultados encontrados estão extremamente próximos dos valores apresentados no enunciado, diferindo apenas por erros de arredondamento do Python.

6.3 c)

Os valores de a_i para cada valor de N estão resumidos na Figura 1 a seguir:

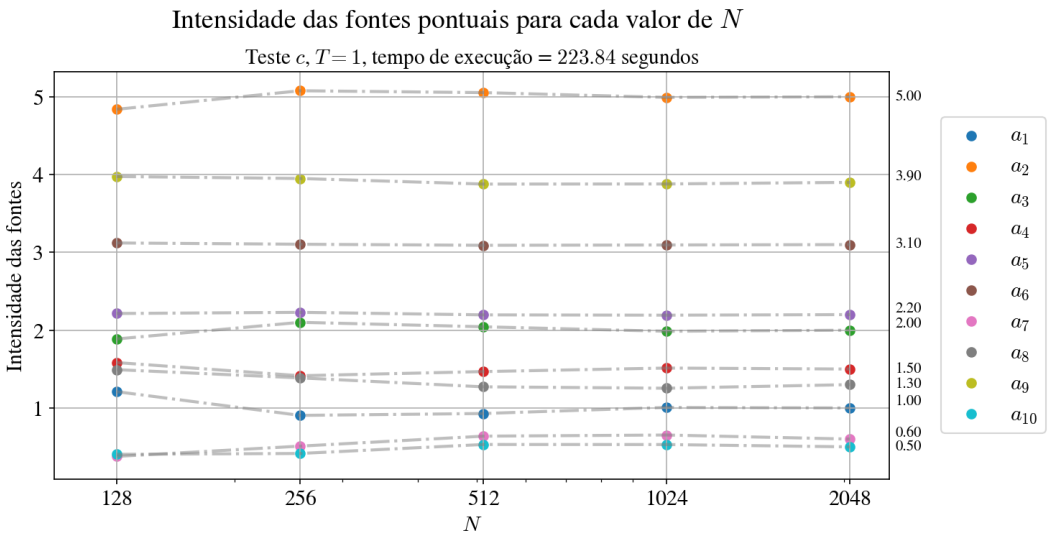

Figura 1: Intensidade das fontes pontuais para cada valor de N para o teste c

De maneira geral, conforme aumenta-se N, os valores começam a convergir para certos valores, o que é corroborado pelo comportamento do erro quadrático. Isto é esperado, pois ao aumentar o valor de N fornece-se a temperatura em mais pontos da barra, o que aprimora a precisão da solução.

As soluções obtidas variando-se o valor de N estão representadas nos gráficos a seguir:

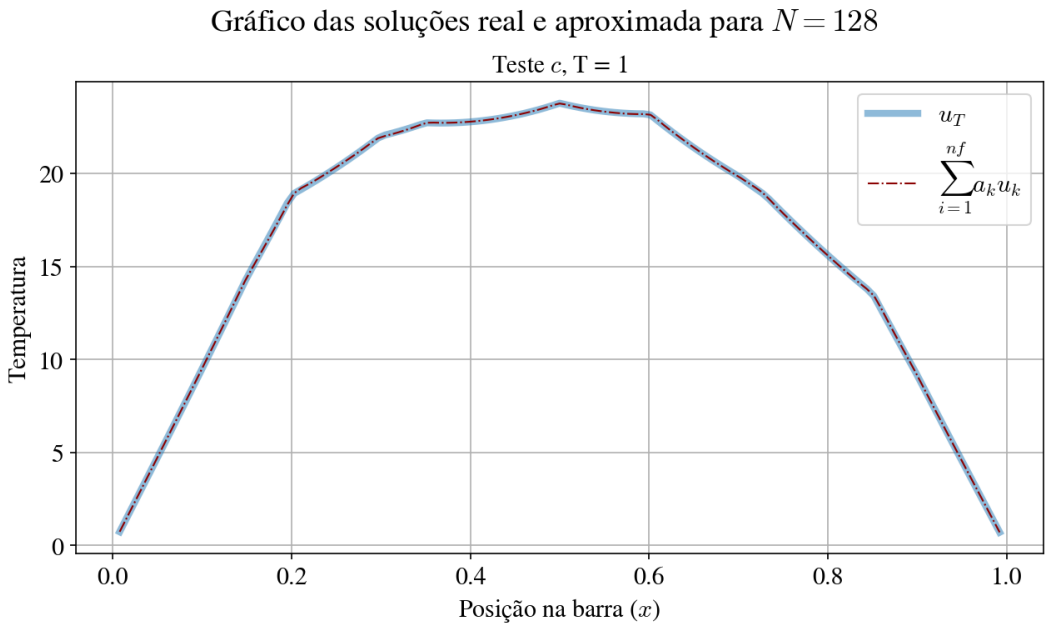

Figura 2: Gráfico de comparação entre as soluções real e aproximada no teste c para $N = 128$

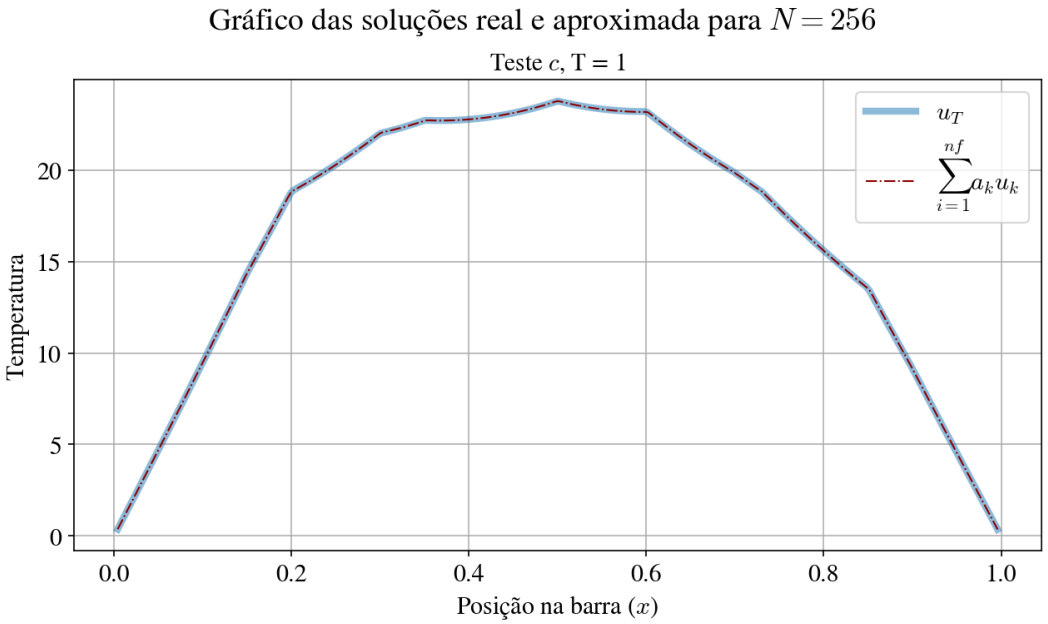

Figura 3: Gráfico de comparação entre as soluções real e aproximada no teste c para $N = 256$

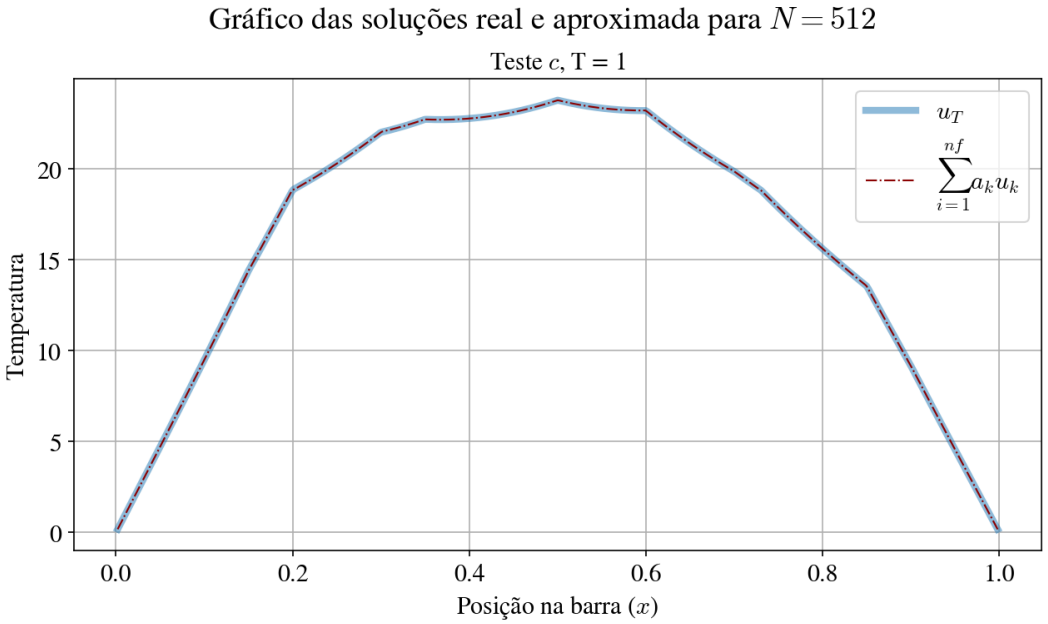

Figura 4: Gráfico de comparação entre as soluções real e aproximada no teste c para $N = 512$

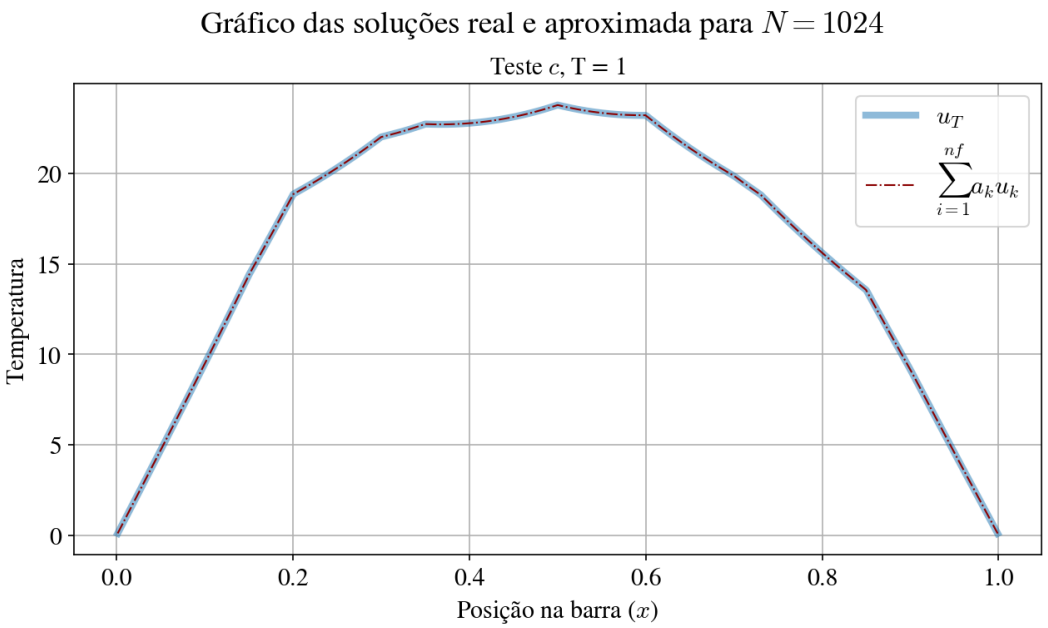

Figura 5: Gráfico de comparação entre as soluções real e aproximada no teste c para $N = 1024$

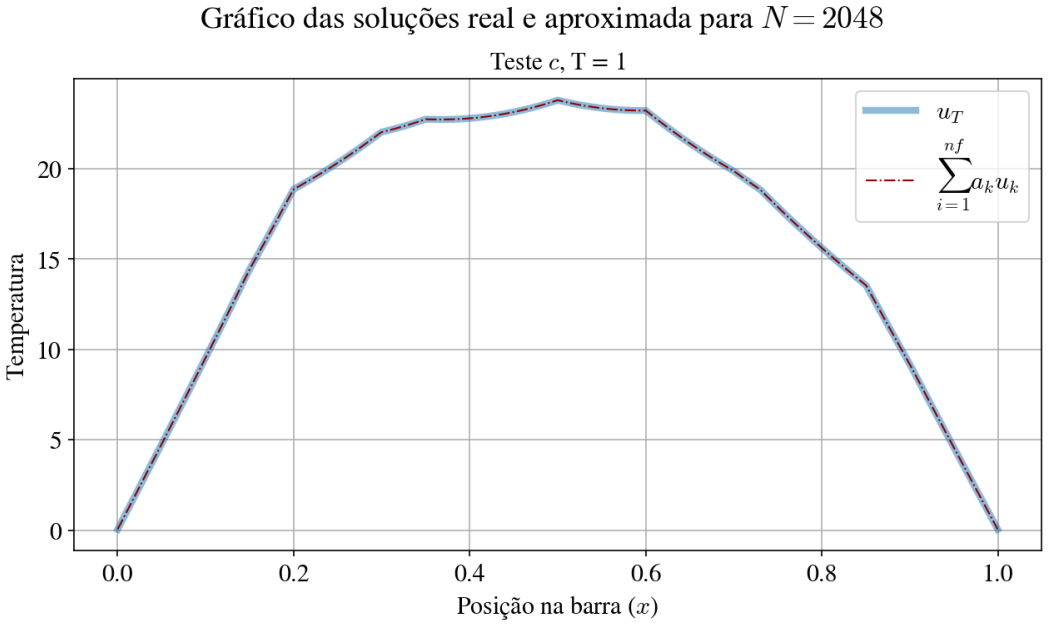

Figura 6: Gráfico de comparação entre as soluções real e aproximada no teste c para $N = 2048$

Na Figura 7 é possível observar a variação do erro quadrático E_2 com N:

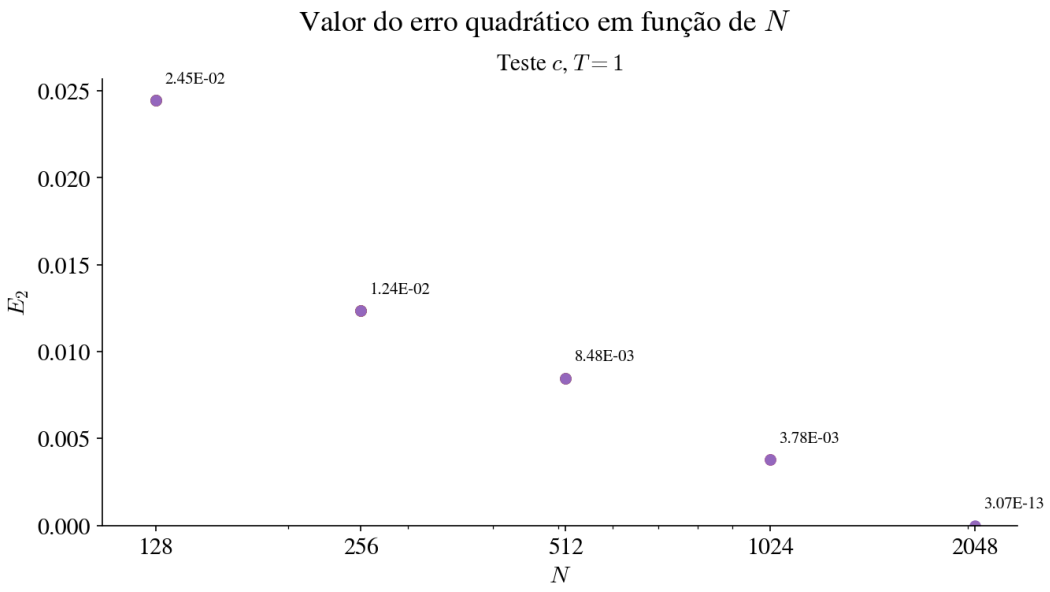

Figura 7: Valor do Erro Quadrático em função N no teste c

Observa-se que o erro quadrático já é relativamente pequeno para $N = 128$, da ordem de 10^{-2}. Por conta disso, a solução obtida já concorda fortemente com a solução dada, e não observam-se diferenças relevantes em seus gráficos conforme varia-se N. Já o erro quadrático, em contrapartida, apresenta uma acelerada redução, com sua ordem caindo de 10^{-2} quando $N = 128$ para 10^{-13} quando $N = 2048$. Este resultado é esperado, uma vez que mais informação sobre as temperaturas é fornecida ao programa, e portanto este é capaz de fornecer uma melhor aproximação.

6.4 d)

Os valores das intensidades a_i das fontes pontuais para cada valor de N no teste d) estão expostos na Figura 8 a seguir:

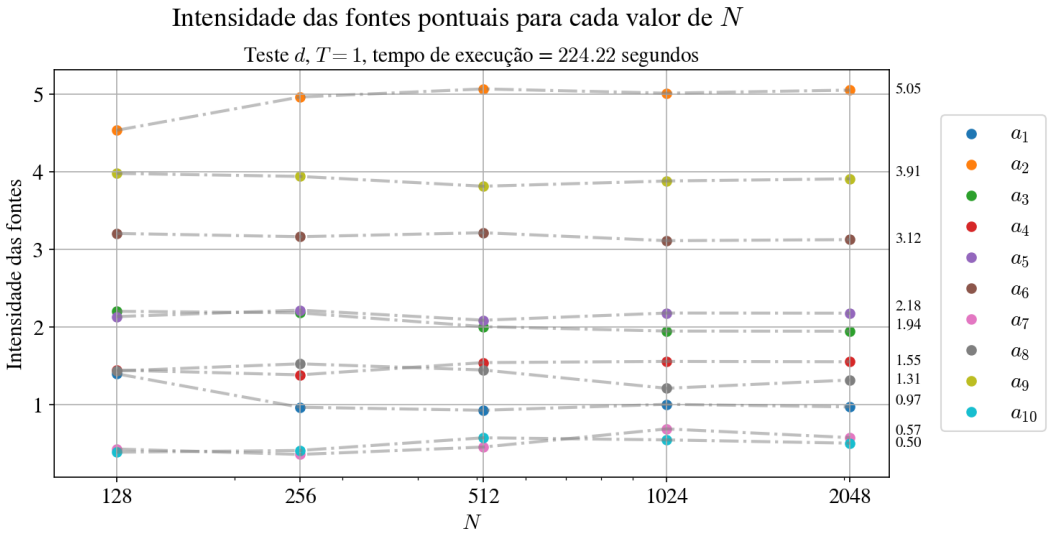

Figura 8: Intensidade das fontes pontuais para cada valor de N para o teste d

Um comportamento similar ao gráfico do teste anterior é observado - conforme aumenta-se N, os valores

das intensidades a_i convergem para certos valores, ainda que apresentem variações mais bruscas devido ao ruído.

As soluções obtidas variando-se o valor de N estão representadas nas Figuras 9 - 13:

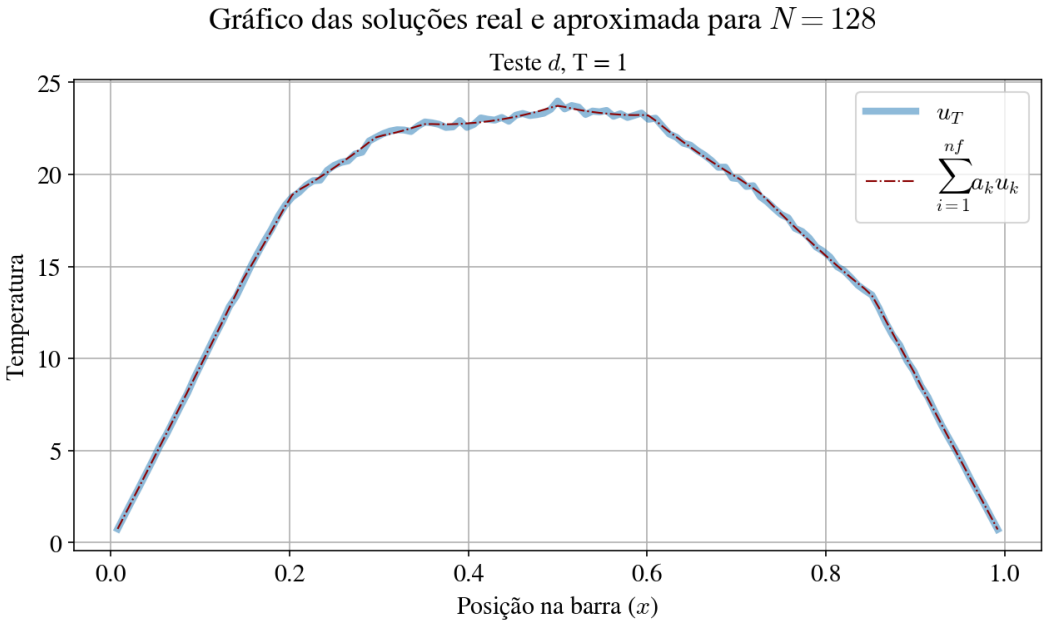

Figura 9: Gráfico de comparação entre as soluções real e aproximada no teste d para $N = 128$

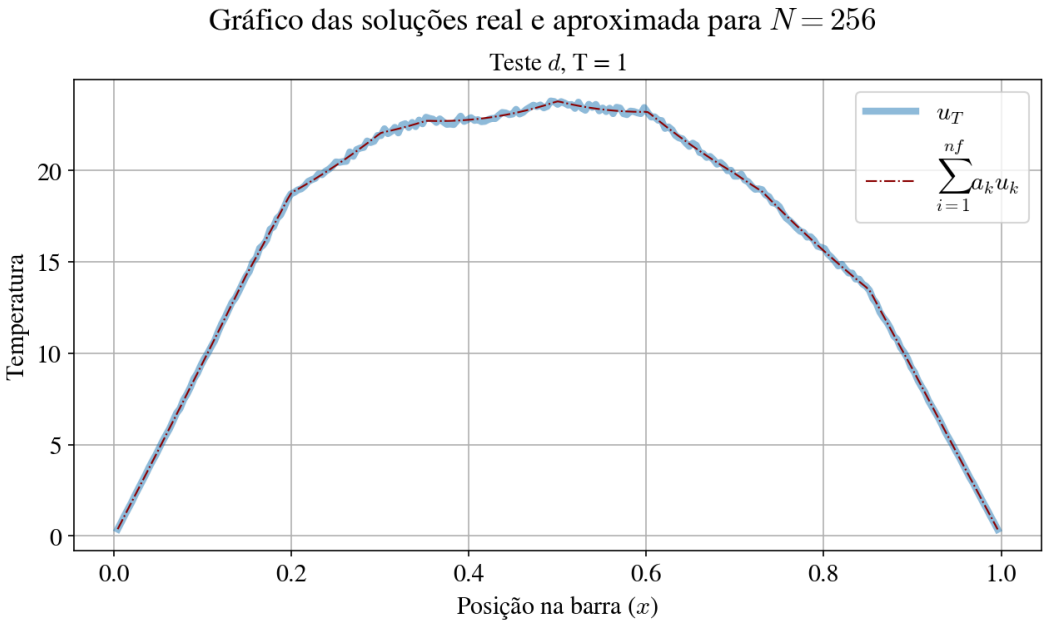

Figura 10: Gráfico de comparação entre as soluções real e aproximada no teste d para $N = 256$

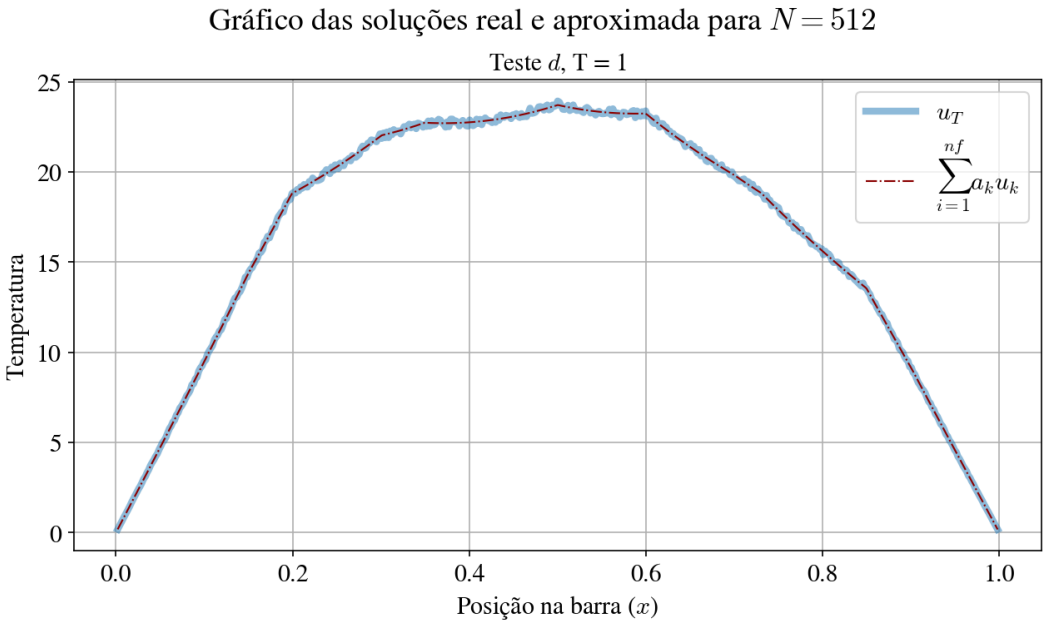

Figura 11: Gráfico de comparação entre as soluções real e aproximada no teste d para $N = 512$

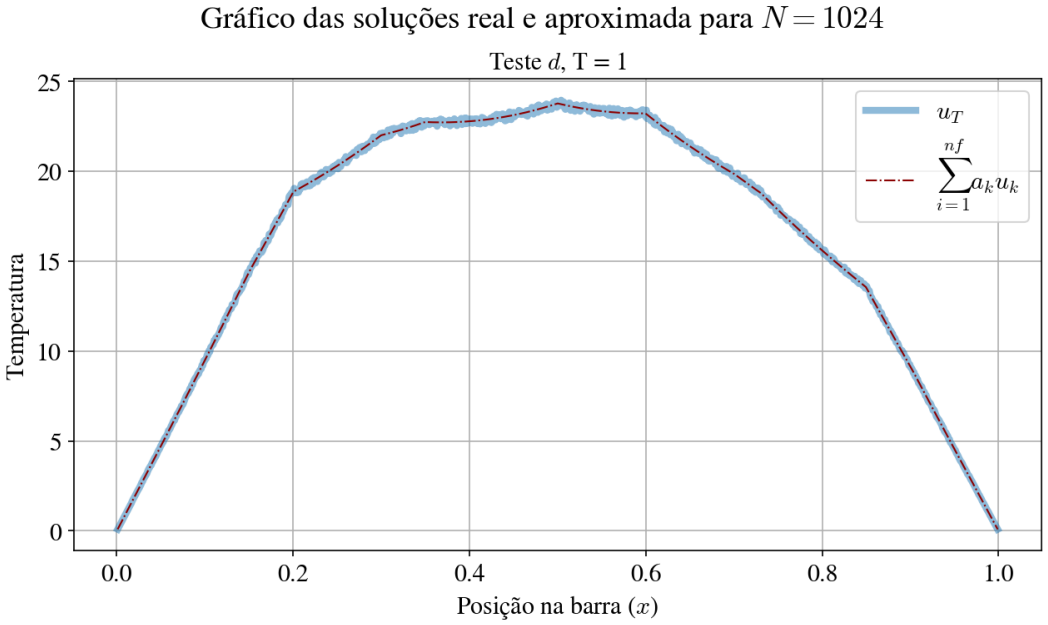

Figura 12: Gráfico de comparação entre as soluções real e aproximada no teste d para $N = 1024$

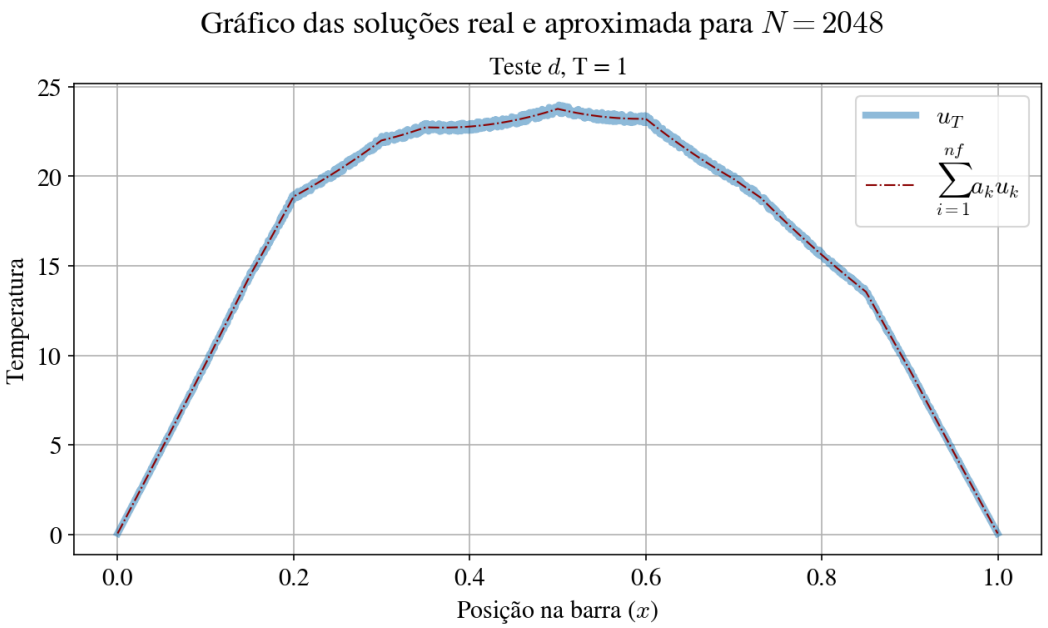

Figura 13: Gráfico de comparação entre as soluções real e aproximada no teste d para $N = 2048$

Já o gráfico da variação do erro quadrático E_2 com N é o seguinte:

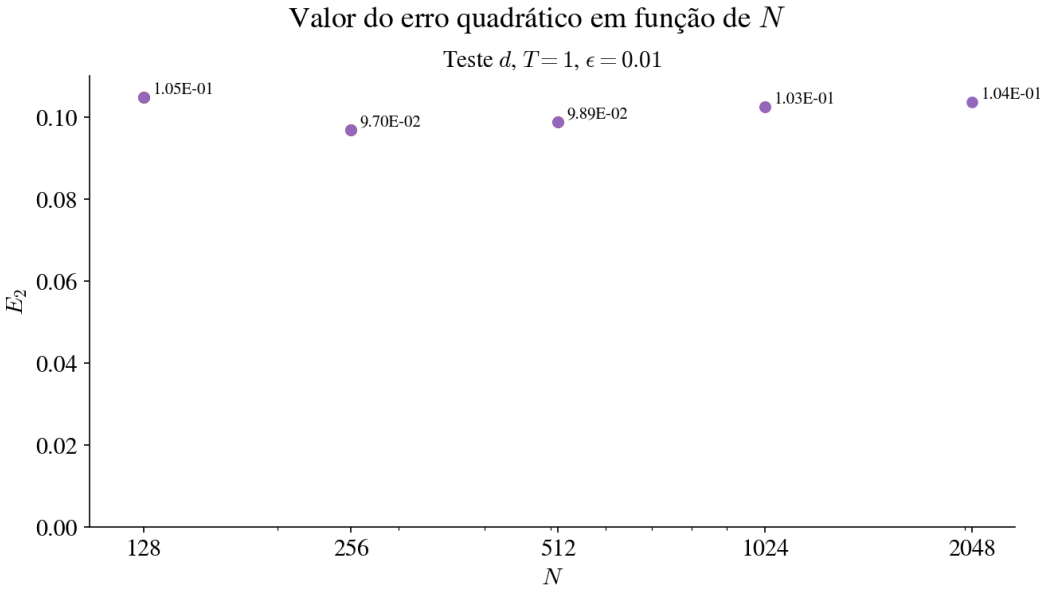

Figura 14: Valor do erro quadrático em função de N no teste d com $\epsilon = 0.01$

Aqui observa-se um fenômeno interessante devido à presença do ruído. O erro quadrático não reduz conforme aumenta-se o valor de N. De fato, ele oscila em torno de um valor igual a 0.1. Isso ocorre pois com a introdução do ruído, os valores da solução u_T tornam-se pseudo-aleatórios, e o erro quadrático é composto por diferenças ponto a ponto. Tomando a temperatura média ao longo da barra em $T = 1$ como da ordem de 10^1, o desvio causado pelo ruído é, em módulo, da ordem $10^1 \cdot 0.01 = 0.1$. Este valor nos fornece uma estimativa para o erro ponto a ponto (note que é superior aos valores do erro quadrático quando não há ruído, sendo o termo dominante no erro do item d)). Por conta deste efeito, que independe de N, o erro quadrático não apresenta redução conforme incrementa-se o número de pontos.

Por fim, têm-se os resultados obtidos ao variar-se o valor de épsilon, o parâmetro do ruído. Nos testes acima foi usado $\epsilon = 0.1$.

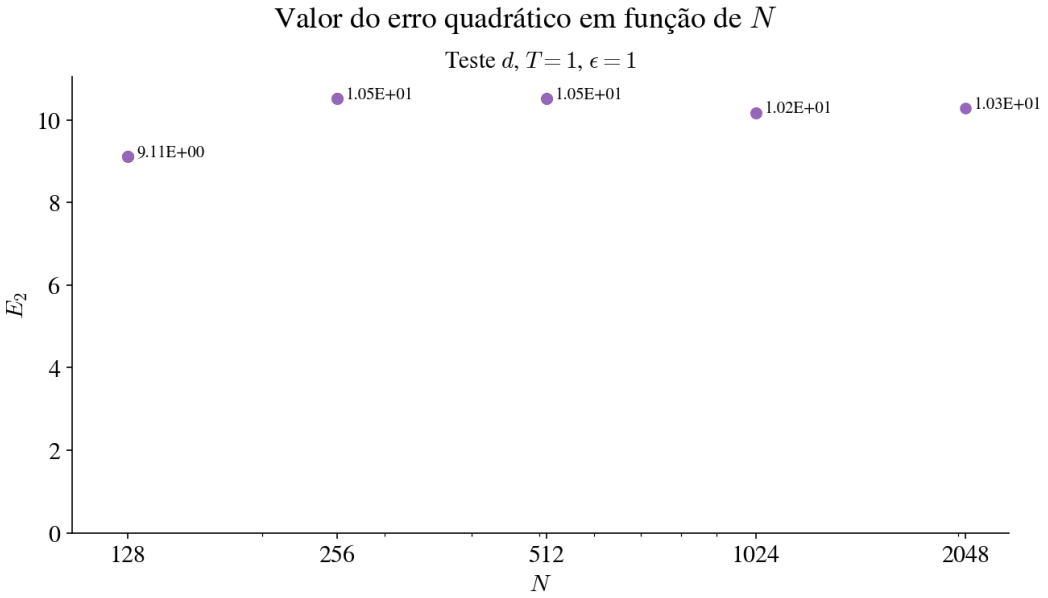

Figura 15: Valor do erro quadrático em função de N no teste d com $\epsilon = 1.0$

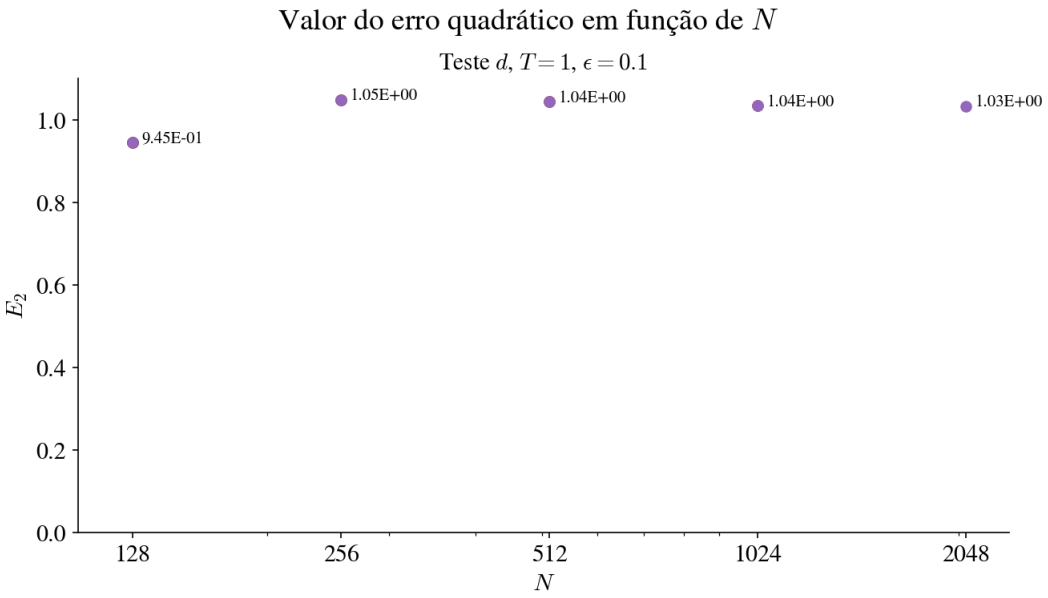

Figura 16: Valor do erro quadrático em função de N no teste d com $\epsilon = 0.1$

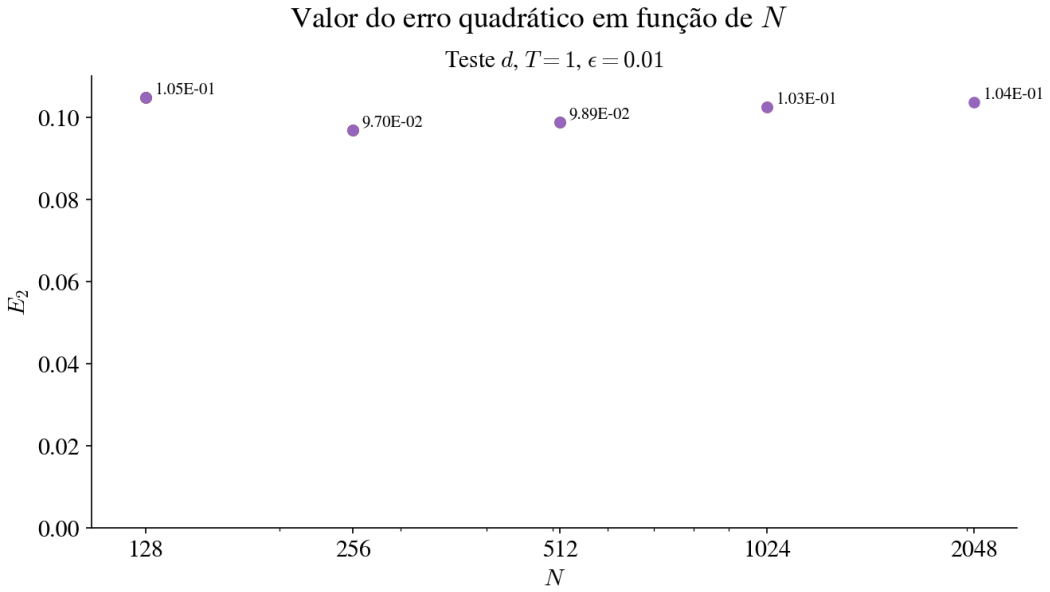

Figura 17: Valor do erro quadrático em função de N no teste d com $\epsilon = 0.01$

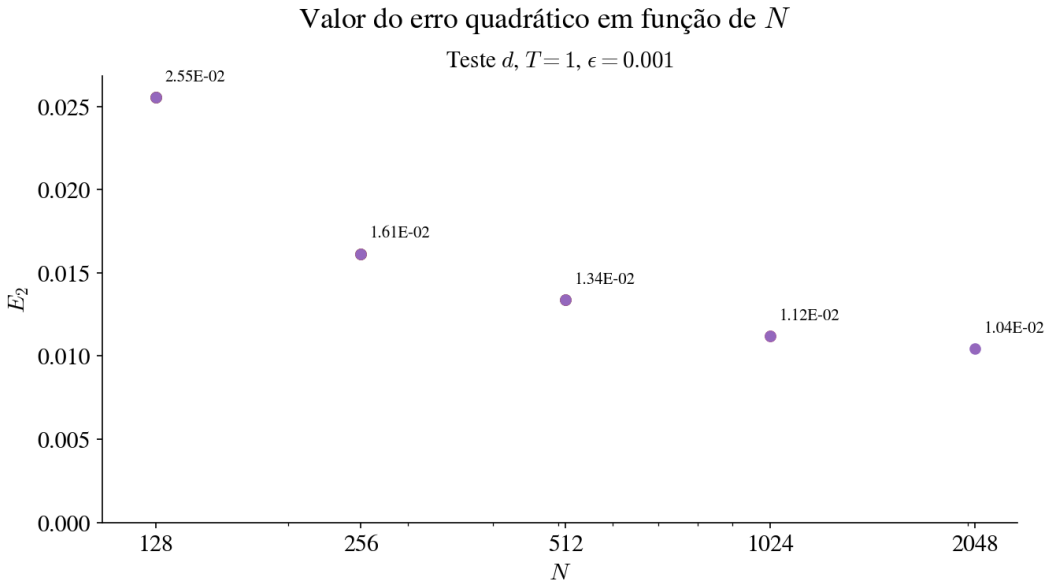

Figura 18: Valor do erro quadrático em função de N no teste d com $\epsilon = 0.001$

Nos casos em que $\epsilon > 0.001$, observa-se que o erro quadrático não é reduzido por N, gravitando em torno de certos valores. Isto pode ser explicado pela análise feita acima, na qual conclui-se que o termo dominante do erro quadrático era devido ao ruído, e portanto não era uma função de N. Já no caso $\epsilon = 0.001$, observa-se um declínio no valor do erro quadrático. Isto ocorre pois neste caso, nos menores valores de N, a magnitude do desvio causado pelo ruído é comparável ao erro obtido quando não há ruído. Sendo que este diminui com N, e nos maiores valores deste o erro quadrático se estabiliza em torno de certo valor, representando a parcela devida ao ruído.

Por fim, é possível notar que, em todos os casos, o erro quadrático (ou seu valor para valores maiores de N, no caso $\epsilon = 0.001$) é dado aproximadamente pelo produto entre ϵ e nf, o número de fontes.

7 Conclusões e considerações finais

Neste exercício programa, muito pôde ser concluído quanto à resolução de problemas inversos. Foi dada uma utilização prática do método dos mínimos quadrados e, a partir de sua implementação, foi constatado um aumento na precisão conforme a discretização das informações fornecia aumentava. Tal conclusão é corroborada pela diminuição do erro quadrático. Além disso, foi possível constatar o efeito do ruído no erro, sendo diretamente responsável por manter seu valor constante conforme N aumentava. Isto ressalta a importância de suprimir quaisquer ruídos ou perturbações nas medições, pois estes acarretam na perda de precisão da solução. A experiência adquirida neste exercício foi valiosa e certamente será de muita utilidade no futuro.

Referências

[1] Richard L. Burden, Douglas J. Faires e Annette M. Burden. *Análise Numérica*. Cengage Learning, 2014.

[2] Guido van Rossum, Barry Warsaw e Nick Coghlan. *Style Guide for Python Code*. PEP 8. 2001. URL: `https://www.python.org/dev/peps/pep-0008/`.

A Resultados de todos os testes rodados

Todos os testes requisitados no enunciado foram executados e os `outputs` gerados encontram-se abaixo:

A.1 Teste A

```
>> run_test(test_name = 'a', N = 128)
a1 = 6.999999999999999
```

A.2 Teste B

```
>> run_test(test_name = 'b', N = 128)
a1 = 2.3000000000000416
a2 = 3.6999999999999633
a3 = 0.2999999999999945
a4 = 4.200000000000023
```

A.3 Teste C

```
>> run_test(test_name = 'c', N = 128)
N = 128
     a1 = 1.2091231792047594
     a2 = 4.839258715746055
     a3 = 1.8872408557576428
     a4 = 1.58339993186339
     a5 = 2.2145040462879138
     a6 = 3.1212947787781866
     a7 = 0.3773402863633759
     a8 = 1.4923482881307724
     a9 = 3.975138801597466
     a10 = 0.40414515364933906
     E_2 = 0.024453403799693

>> run_test(test_name = 'c', N = 256)
N = 256
     a1 = 0.9045010343167235
     a2 = 5.077572635562824
     a3 = 2.10085359547735
     a4 = 1.4141556850890353
     a5 = 2.2292450130537187
     a6 = 3.104613856990656
     a7 = 0.509452597392106
     a8 = 1.38650879045667
     a9 = 3.949878646151287
     a10 = 0.4148931283310377
     E_2 = 0.012363464048869186

>> run_test(test_name = 'c', N = 512)
N = 512
     a1 = 0.9286883784938489
     a2 = 5.053707844478872
     a3 = 2.0437010489053513
     a4 = 1.4676706728635018
     a5 = 2.196763331997367
     a6 = 3.0911311688990724
     a7 = 0.6375875163788951
     a8 = 1.2716872153206644
     a9 = 3.8780948673276554
     a10 = 0.5305567786423463
     E_2 = 0.008476628330826349

>> run_test(test_name = 'c', N = 1024)
N = 1024
     a1 = 1.007281322075574
     a2 = 4.992443012452249
     a3 = 1.9858767276157145
     a4 = 1.5132584652238705
     a5 = 2.1926928376832038
     a6 = 3.095152875934147
     a7 = 0.6523266477781835
     a8 = 1.2537898890640804
     a9 = 3.879667056940479
     a10 = 0.5297366253017982
     E_2 = 0.003779310463290562

>> run_test(test_name = 'c', N = 2048)
N = 2048
     a1 = 0.9999999999985469
     a2 = 5.000000000001094
     a3 = 2.000000000000128
     a4 = 1.5000000000006573
     a5 = 2.2000000000022144
     a6 = 3.099999999999283
     a7 = 0.6000000000025372
     a8 = 1.2999999999977767
     a9 = 3.899999999999695
     a10 = 0.5000000000009726
     E_2 = 3.068812698281959e-13
```

A.4 Teste D

```
>> run_test(test_name = 'd', N = 128)
N = 128
     a1 = 1.3935210164266003
     a2 = 4.534607374958988
     a3 = 2.199386416010782
     a4 = 1.4413297205936573
     a5 = 2.1311406656627536
     a6 = 3.2035589019063684
     a7 = 0.4230089783029447
     a8 = 1.4303499666910202
     a9 = 3.9783256656177466
     a10 = 0.38260053702100305
     E_2 = 0.10488526630020026

>> run_test(test_name = 'd', N = 256)
N = 256
     a1 = 0.9617946611432941
     a2 = 4.9634205936866
     a3 = 2.1791835158729107
     a4 = 1.3798627954631648
     a5 = 2.214566351873602
     a6 = 3.1621108548636787
     a7 = 0.3533662343287336
     a8 = 1.5223291017938472
     a9 = 3.939121880639698
     a10 = 0.4045477635306339
     E_2 = 0.0969709427160337

>> run_test(test_name = 'd', N = 512)
N = 512
     a1 = 0.9231835811521876
     a2 = 5.067800153806822
     a3 = 2.0016104610412064
     a4 = 1.5379257532957666
     a5 = 2.084695642144725
     a6 = 3.212904391430346
     a7 = 0.4487152151170033
     a8 = 1.4424665120793545
     a9 = 3.8126126853512536
     a10 = 0.5684134001080254
     E_2 = 0.09889865013887882

>> run_test(test_name = 'd', N = 1024)
N = 1024
     a1 = 0.9997422471041943
     a2 = 5.0126411756717175
     a3 = 1.9443016903381114
     a4 = 1.5532828806319117
     a5 = 2.1770957552120187
     a6 = 3.110120336767899
     a7 = 0.6827830591728459
     a8 = 1.2064813275094828
     a9 = 3.8801678955904806
     a10 = 0.5410617646852851
     E_2 = 0.10253123835678817

>> run_test(test_name = 'd', N = 2048)
N = 2048
     a1 = 0.9668715752466497
     a2 = 5.052590861813805
     a3 = 1.9424024141604868
     a4 = 1.5487983830444136
     a5 = 2.1752393020136847
     a6 = 3.124027102810243
     a7 = 0.5702384495733543
     a8 = 1.3131363598707413
     a9 = 3.9081059900917583
     a10 = 0.49857895490787735
     E_2 = 0.1036905162885735
```