



ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Exercício Programa 2 de Cálculo Numérico (MAP3121)

Turma 2	No. USP
Lucas Haug	10773565
Renzo Armando dos Santos	10772414
Abensur	

São Paulo
2020

Lucas Haug
Renzo Armando dos Santos Abensur

Exercício Programa 2 de Cálculo Numérico (MAP3121)

Relatório apresentado como requisito para avaliação na disciplina MAP3121 - Métodos Numéricos e Aplicações, no curso de Engenharia Elétrica oferecido pela Escola Politécnica da Universidade de São Paulo.

Professor: Nelson Mugayar Kuhl

São Paulo
2020

SUMÁRIO

1. Introdução	4
2. Desenvolvimento	6
2.1 Primeira tarefa	7
2.2 Segunda tarefa	8
2.3 Terceira tarefa	9
3. Testes	11
3.1 Teste a	11
3.1.1 Dados do problema	11
3.1.2 Resultados	12
3.1.2.1 Gráfico	12
3.1.2.2 Coeficiente	12
3.1.2.3 Erro Quadrático	12
3.1.3 Discussão	12
3.2 Teste b	13
3.2.1 Dados do problema	13
3.2.2 Resultados	13
3.2.2.1 Gráfico	13
3.2.2.2 Coeficientes	14
3.2.2.3 Erro Quadrático	14
3.2.3 Discussão	14
3.3 Teste c	15
3.3.1 Dados do problema	15
3.3.2 Resultados	15
3.3.2.1 Gráficos	15
3.3.2.2 Coeficientes	18
3.3.2.3 Erro Quadrático	19
3.3.3 Discussão	19
3.4 Teste d	20
3.4.1 Dados do problema	20
3.4.2 Resultados	20
3.4.2.1 Gráficos	20
3.4.2.2 Coeficientes	24
3.4.2.3 Erro Quadrático	25
3.4.3 Discussão	26
4. Conclusão	26

1. Introdução

Este relatório procura especificar a resolução encontrada para o segundo exercício programa da disciplina MAP3121 - Métodos Numéricos e Aplicações, no primeiro semestre de 2020.

O exercício programa procura, a partir do conhecimento da distribuição final da temperatura no instante T , determinar a intensidade das fontes de calor aplicadas em posições conhecidas da barra.

Para isso, foram consideradas as forçantes da seguinte forma:

$$f(t, x) = r(t) \sum_{k=1}^{nf} a_k g_h^k(x) ,$$

sendo $g_h^k(x)$ as forçantes pontuais ao longo da barra, a_k as intensidades dessas forçantes e $r(t)$ uma função que descreve a variação temporal das forçantes que nesse exercício programa será considerada como $r(t) = 10(1 + \cos(5t))$.

Devida à linearidade das equações (considerando as condições iniciais e de contorno nulas) pode ser feito o seguinte equacionamento:

$$u_T(x) = \sum_{k=1}^{nf} a_k u_k(T, x) .$$

Onde nf é o número de fontes pontuais.

Dessa forma, como $u_T(x)$ é conhecida e como as funções $u_k(T, x)$ podem ser aproximadas resolvendo-se as equações abaixo por meio do método de Crank-Nicolson, é possível determinar os coeficientes a_k com o método dos mínimos quadrados.

$$\begin{aligned}
u_t(t, x) &= u_{xx}(t, x) + f(t, x) \text{ em } [0, T] \times [0, 1], \\
u(0, x) &= u_0(x) \text{ em } [0, 1] \\
u(t, 0) &= g_1(t) \text{ em } [0, T] \\
u(t, 1) &= g_2(t) \text{ em } [0, T].
\end{aligned}$$

Para a resolução do exercício programa foi elaborado em python 3, sendo separado em arquivos diferentes para melhor organizar o código.

2. Desenvolvimento

O exercício programa foi dividido em vários arquivos para facilitar a compreensão do código. Dessa forma foram feitos os seguintes arquivos:

- `plotter.py` → função para criar os gráficos
- `crank_nicolson.py` → funções relacionadas à resolução pro problema direto pelo método de Crank-Nicolson (retirado e ligeiramente alterado do EP1).
- `mmq.py` → funções relacionadas ao método dos mínimos quadrados, incluindo resolução de sistemas lineares necessárias para o método.
- `tests.py` → funções que retornam os dados específicos de cada um dos testes a, b, c e d.
- `main.py` → interage com o usuário, sendo possível ser feita a escolha de qual teste se irá fazer, além de se fazer a chamada de todas as funções necessárias para o cálculo das intensidades a_k e o erro quadrático.

Dessa forma para a resolução do problema do cálculo das intensidades, primeiramente foram calculados os vetores $u_k(T, x_i)$ (primeira tarefa), com eles, conhecido o valor de $u_T(x_i)$, foi montado o sistema normal para o problema de mínimos quadrados (segunda tarefa), então, utilizando-se a decomposição LDL^t da matriz simétrica do sistema normal, foi resolvido o sistema linear e foram calculadas as intensidades a_k e o erro quadrático da solução.

Para o cálculo do erro quadrático foi utilizada a seguinte equação:

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

Para o cálculo do erro discreto então se considerou uma função $f(x_i)$ genérica da seguinte forma:

$$f(x_i) = a_0 \cdot g_0(x_i) + a_1 \cdot g_1(x_i) + \dots + a_k \cdot g_k(x_i), (i = 1, \dots, N - 1),$$

Dessa forma, foi feito o seguinte código onde, para o problema a ser solucionado, `f_array` representa os valores conhecidos da função $u_T(x_i)$, `g_matrix` o vetor de vetores (matriz) $u_k(T, x_i)$ e a_k as intensidades das fontes pontuais.

```
def squared_error_calculation(f_array, g_matrix, coefficients_array):
    """
    Cálculo discreto do erro quadrático.
    """

    num_of_xs = len(f_array)
    f_approx_array = f_approximation(g_matrix, coefficients_array)

    error_sum = 0

    for i in range(0, num_of_xs):
        error_sum += ((f_array[i] - f_approx_array[i])**2)

    error_sum /= num_of_xs

    error = np.sqrt(error_sum)
    return error
```

2.1 Primeira tarefa

Para a primeira tarefa, gerar os vetores $u_k(T, x_i)$, $i = 1, \dots, N - 1$, foi retirado o método de Crank-Nicolson desenvolvido do EP1 e foi desenvolvida a seguinte função:

```
def generate_uk(heat_sources_positions_array, N):
    """
    Gera os vetores uk(T, xi), i = 0, ..., N
    """

    nf = len(heat_sources_positions_array)

    uk_matrix = np.zeros((nf, N + 1))

    for k in range(0, nf):
        uk_matrix[k], scale_array = solve_heat_equation(heat_sources_positions_array[k], N)

    return uk_matrix, scale_array
```

Essa função resolve a equação de calor dependendo do número de fontes de calor pontuais, a partir do método de Crank-Nicolson (função `solve_heat_equation`), gera um vetor de vetores $u_k(T, x_i)$ (uma matriz), retornando-o.

2.2 Segunda tarefa

Para a segunda tarefa foi desenvolvida uma função genérica que dada a função

$$f(x_i) = a_0 \cdot g_0(x_i) + a_1 \cdot g_1(x_i) + \dots + a_k \cdot g_k(x_i), (i = 1, \dots, N - 1),$$

monta a matriz e o sistema normal do problema de mínimos quadrados para o cálculo dos coeficientes a_k .

Para isso foi elaborada a seguinte função:

```
def generate_linear_system(f_array, g_matrix):
    """
    Dado f(x) = a0 * g0(x) + a1 * g1(x) + ... + ak * gk(x)

    Retorna a matriz A e b do sistema normal A * x = b, gerados
    a partir de f(x) e os vetores g.
    """

    num_of_coefficients = len(g_matrix)
    a_matrix = np.zeros((num_of_coefficients, num_of_coefficients), dtype=float)
    b_array = np.zeros(num_of_coefficients, dtype=float)

    for k in range(0, num_of_coefficients):
        b_array[k] = np.inner(f_array, g_matrix[k])

    for i in range(0, num_of_coefficients):
        for j in range(i, num_of_coefficients):
            inner_product = np.inner(g_matrix[i], g_matrix[j])
            a_matrix[i][j] = inner_product

        if i != j:
            a_matrix[j][i] = inner_product
```



```
return a_matrix, b_array
```

2.3 Terceira tarefa

Para a terceira tarefa, foi implementada a seguinte função que faz a decomposição LDL^t de uma matriz A simétrica não esparsa:

```
def matrix_decomposition(a_matrix):
    """
    Decompõe uma matrix A simétrica em três matrizes L, D e
    Lt, retornando apenas dois vetores que representam as
    matrizes L e D.

    Aviso: A matriz A é mudada dentro da função.
    """

    matrix_dimension = len(a_matrix)

    l_matrix = np.zeros((matrix_dimension, matrix_dimension), dtype=float)
    d_matrix = np.zeros((matrix_dimension, matrix_dimension), dtype=float)

    for f in range(0, matrix_dimension):
        # Generate L matrix
        for l in range(f, matrix_dimension):
            l_matrix[l][f] = a_matrix[l][f] / a_matrix[f][f]

        # Generate the new matrix A
        for c in range(0, matrix_dimension):
            for l in range(f + 1, matrix_dimension):
                a_matrix[l][c] = a_matrix[l][c] - l_matrix[l][f] * a_matrix[f][c]

        # Generate D matrix
        for l in range(0, matrix_dimension):
            d_matrix[l][l] = a_matrix[l][l]

    return l_matrix, d_matrix
```

Além disso foi desenvolvida a seguinte função que utiliza a decomposição LDL^t para resolver um sistema linear:

```
def solve_linear_system(a_matrix, b_array):  
    """  
    Soluciona um sistema  $Ax = b$ , onde A é uma matrix simétrica.  
  
    Para a resolução do sistema, é feita a decomposição de A para  $L^*D^*L^t$ .  
  
    É feita a divisão do problema em três sistemas menores:  
     $L * y = b$   
     $D * z = y$   
     $L^t * x = z$   
    """  
  
    matrix_dimension = len(a_matrix)  
  
    l_matrix, d_matrix = matrix_decomposition(a_matrix)  
  
    # First system solution ->  $L * y = b$   
    y_array = np.zeros(matrix_dimension, dtype=float)  
  
    y_array[0] = b_array[0]  
  
    for l in range(1, matrix_dimension):  
        y_array[l] = b_array[l]  
  
        for m in range(0, l):  
            y_array[l] = y_array[l] - y_array[m] * l_matrix[l][m]  
  
    # Second system solution ->  $D * z = y$   
    z_array = np.zeros(matrix_dimension, dtype=float)  
  
    for l in range(0, matrix_dimension):  
        z_array[l] = y_array[l] / d_matrix[l][l]  
  
    # Third system solution ->  $L^t * x = z$   
    x_array = np.zeros(matrix_dimension, dtype=float)  
  
    x_array[-1] = z_array[-1]
```

```

for l in range(matrix_dimension - 2, -1, -1):
    x_array[l] = z_array[l]

    for m in range(matrix_dimension - 1, l, -1):
        x_array[l] = x_array[l] - x_array[m] * np.transpose(l_matrix)[l][m]

return x_array

```

Essa função recebe a matriz A simétrica e um vetor b para achar a solução da equação $Ax = b$ e através da decomposição da matriz A simétrica por meio da função “matrix_decomposition” implementada acima, resolve o problema, quebrando-o em três sistemas menores para facilitar a resolução:

$$L \times y = b$$

$$D \times z = y$$

$$L^t \times x = z$$

Para então retornar o valor do vetor x que representa a solução do problema.

3. Testes

Os dados de cada teste foram organizados no arquivo tests.py, onde há uma função que retorna o valor do vetor $u_T(x_i)$, o vetor de vetores $u_k(T, x_i)$, os valores x_i e o valor de N utilizados para o teste escolhido.

Em todos os testes utilizaremos $T = 1$ e $r(t) = 10(1 + \cos(5t))$.

3.1 Teste a

3.1.1 Dados do problema

Para o teste a foi considerado $N = 128$, $nf = 1$ e $p_1 = 0.35$. Além de se calcular $u_T(x_i)$ da seguinte forma:

$$u_T(x_i) = 7 \cdot u_1(T, x_i)$$

3.1.2 Resultados

3.1.2.1 Gráfico

Para o teste em questão foi obtido o seguinte gráfico de $u_T(x_i)$, sendo mostrado o valor medido e o valor calculado aproximado.

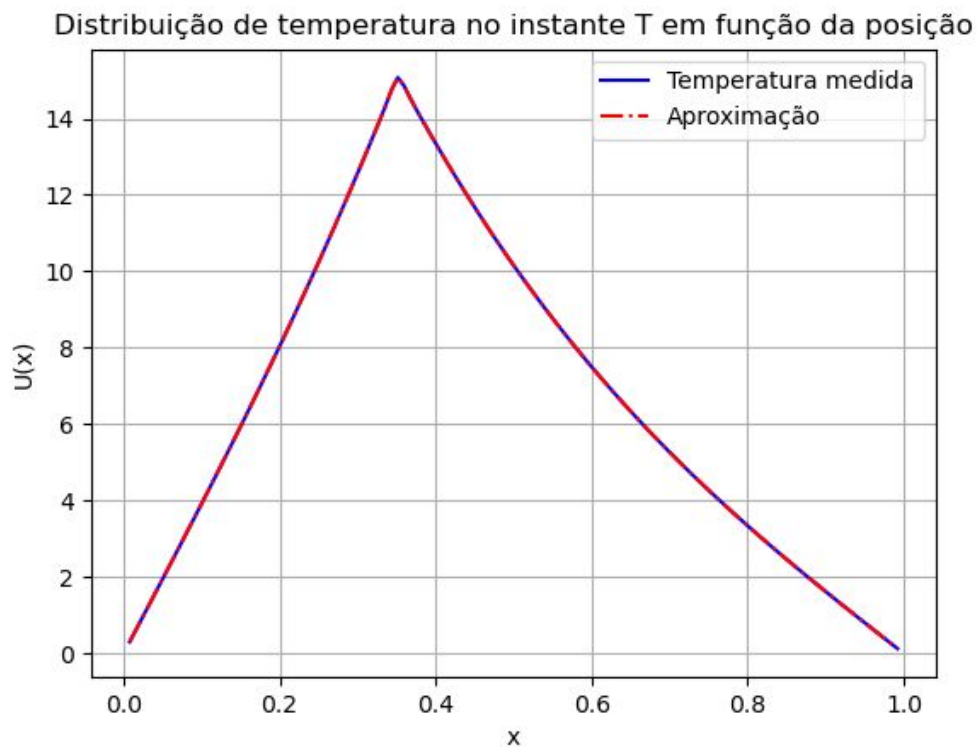


Gráfico 1 - Teste a com $N = 128$

3.1.2.2 Coeficiente

Para o teste a, o coeficiente 7 foi recuperado, obtendo-se precisamente 7.0.

3.1.2.3 Erro Quadrático

O erro obtido para o teste foi de 0.

3.1.3 Discussão

Para o teste a, como esperado o coeficiente obtido foi 7 e como era um sistema simples, não houveram muitos erros de aproximação, dessa forma se obteve o valor exato 7 com erro quadrático 0.

3.2 Teste b

3.2.1 Dados do problema

Para o teste b foi considerado $N = 128$, então foram utilizadas 4 fontes pontuais (nf = 4) sendo as posições de cada uma as seguintes: $p_1 = 0.15$, $p_2 = 0.3$, $p_3 = 0.7$ e $p_4 = 0.8$.

Além disso, o cálculo de $u_T(x_i)$ foi feito da seguinte forma:

$$u_T(x_i) = 2.3 \cdot u_1(T, x_i) + 3.7 \cdot u_2(T, x_i) + 0.3 \cdot u_3(T, x_i) + 4.2 \cdot u_4(T, x_i)$$

3.2.2 Resultados

3.2.2.1 Gráfico

Para o teste em questão foi obtido o seguinte gráfico de $u_T(x_i)$, sendo mostrado o valor medido e o valor calculado aproximado.

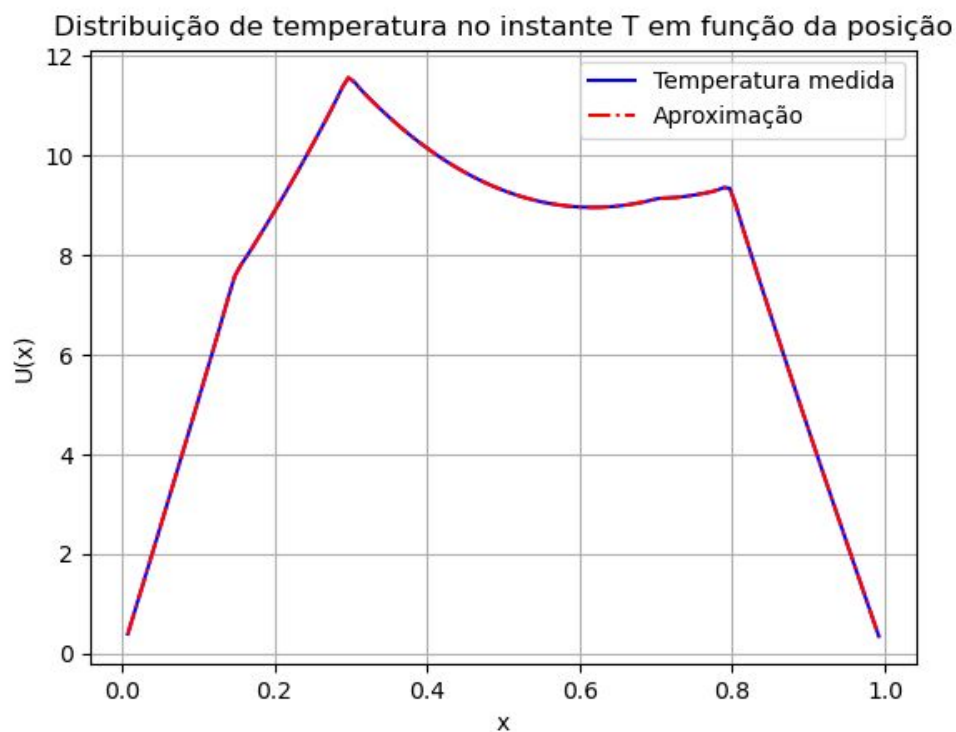


Gráfico 2 - Teste b com $N = 128$

3.2.2.2 Coeficientes

Foram obtidos os seguintes coeficientes:

Intensidades	N = 128
a_0	2.30000000000000496
a_1	3.69999999999999615
a_2	0.300000000000004867
a_3	4.19999999999999496

Tabela 1 - Intensidades do teste b com N = 128

3.2.2.3 Erro Quadrático

Para o teste em questão foi obtido o seguinte erro quadrático:

	N = 128
Erro	1.5499605530411918e-14

Tabela 2 - Erro quadrático do teste b com N = 128

3.2.3 Discussão

Para o problema, os coeficientes dados eram de $a_0 = 2,3$, $a_1 = 3,7$, $a_2 = 0,3$ e $a_3 = 4,2$, fazendo-se os cálculos com o programa os coeficientes obtidos resultaram em valores um pouco diferente dos valores esperados, isso pode ser justificado visto que, durante o execução do programa, ocorrem diversos arredondamentos. Contudo, podemos verificar que o erro quadrático associado (Erro = 1,5499605530411918e-14), é relativamente pequeno, então pode-se considerar que foram recuperados os coeficientes dados para o problema. Além disso, pode-se ver no gráfico de $u_T(x_i)$ como cada uma das fontes pontuais influencia na distribuição da temperatura.

3.3 Teste c

3.3.1 Dados do problema

Para o teste c foram utilizados os dados disponibilizados no arquivo teste.txt fornecido. Além disso, foram utilizados diferentes valores de N , sendo eles 128, 256, 512, 1024 e 2048. Porém para a construção da malha foi utilizado, independente do valor de N , $\Delta x = 1/2048$.

3.3.2 Resultados

3.3.2.1 Gráficos

Para o teste em questão foram obtidos os seguintes gráficos de $u_T(x_i)$, para diferentes valores de N , sendo mostrado o valor medido e o valor calculado aproximado.

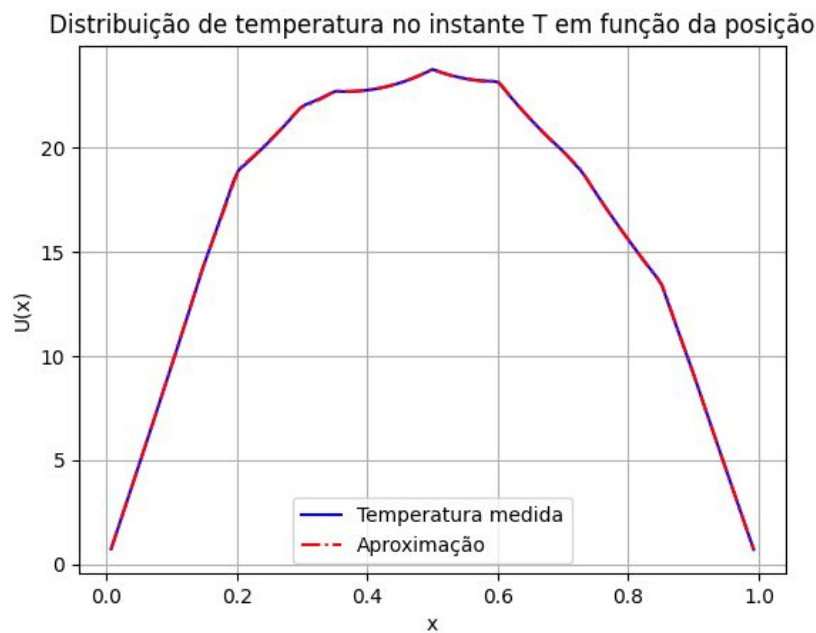


Gráfico 3 - Teste c com $N = 128$

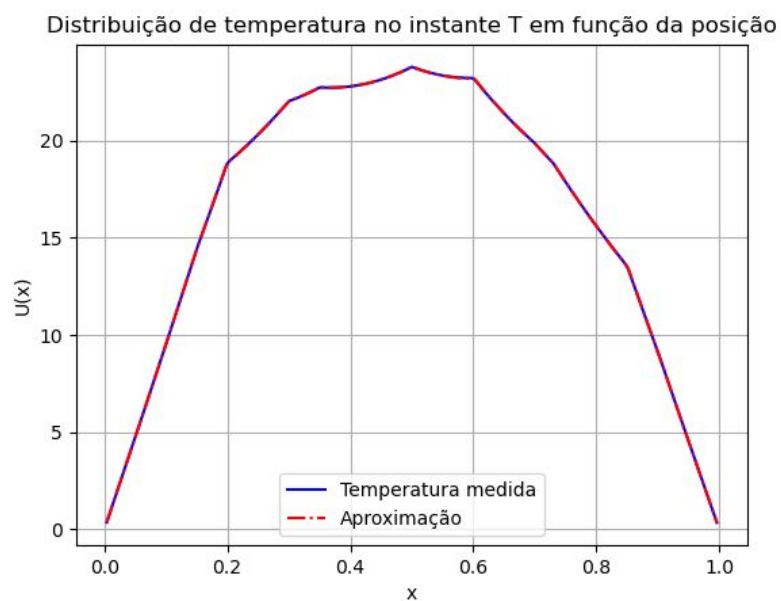


Gráfico 4 - Teste c com $N = 256$

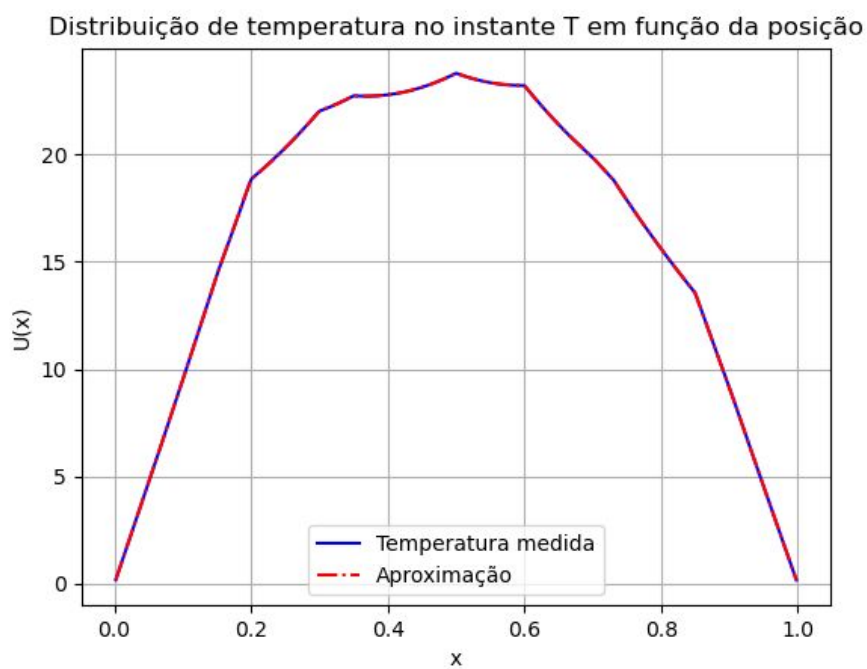
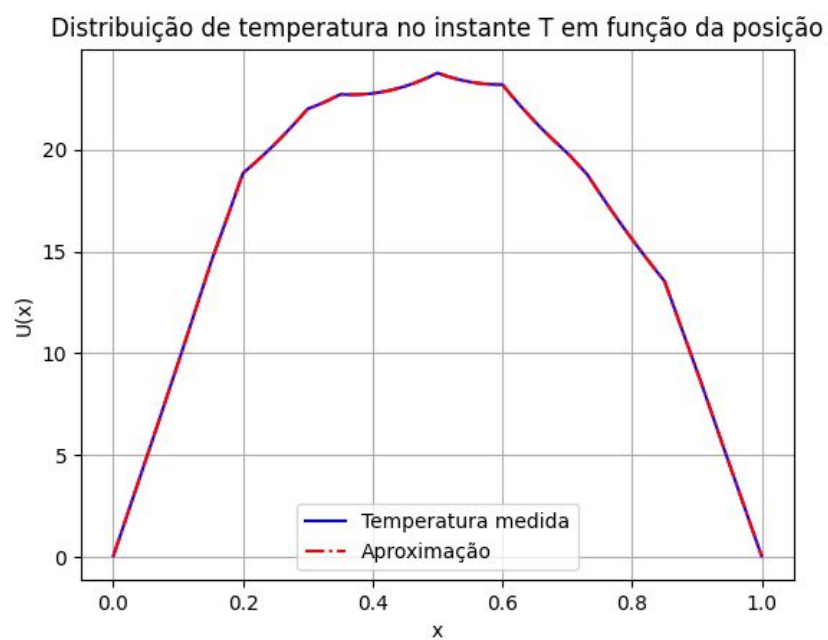
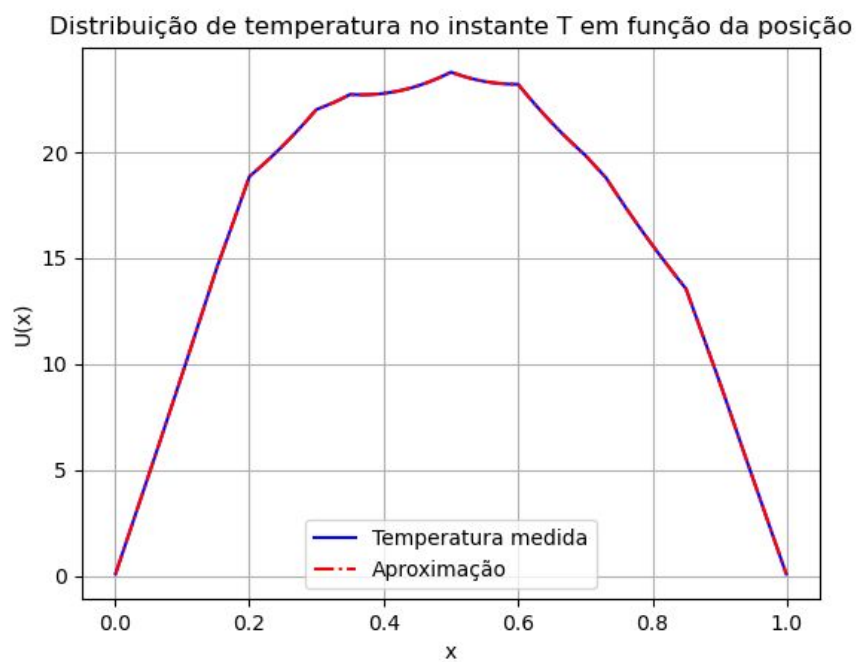


Gráfico 5 - Teste c com $N = 512$



3.3.2.2 Coeficientes

Para o teste c foram obtidos os seguintes valores das intensidades para diferentes valores de N:

Intensidades	N = 128	N = 256	N= 512	N = 1024	N = 2048
a_0	1.20912317 92041466	0.9045010343 18792	0.928687995401 3925	1.0072811320 381456	0.999999999 4974251
a_1	4.83925871 5747438	5.0775726355 59471	5.053708843340 618	4.9924435219 35543	5.000000001 346006
a_2	1.88724085 57555072	2.1008535954 811185	2.043694188439 7672	1.9858733479 945263	1.999999991 242201
a_3	1.58339993 18648742	1.4141556850 867802	1.467679034237 7668	1.5132625788 409024	1.500000012 1757147
a_4	2.21450404 62885293	2.2292450130 542063	2.196760623033 9065	2.1926914902 72952	2.199999992 1992055
a_5	3.12129477 87756202	3.1046138569 9006	3.091132037680 2847	3.0951533063 72225	3.100000004 782977
a_6	0.37734028 637430606	0.5094525973 94451	0.637587121782 3657	0.6523264491 115244	0.599999997 7690007
a_7	1.49234828 81207946	1.3865087904 549664	1.271687421732 9952	1.2537899951 697167	1.300000001 217759
a_8	3.97513880 15992326	3.9498786461 5312	3.878094855744 8465	3.8796670512 795735	3.899999999 950573
a_9	0.40414515 36483273	0.4148931283 302513	0.530556784760 6538	0.5297366283 025545	0.500000000 0401839

Tabela 3 - Intensidades do teste c

3.3.2.3 Erro Quadrático

Para o teste c foram obtidos os seguintes valores de erros quadráticos para diferentes valores de N:

N	Erro Quadrático
128	0.02454948827120726
256	0.0123876824149804
512	0.0084850419037757
1024	0.0037812111602405283
2048	3.6214249335228157e-06

Tabela 4 - Erros quadráticos do teste c

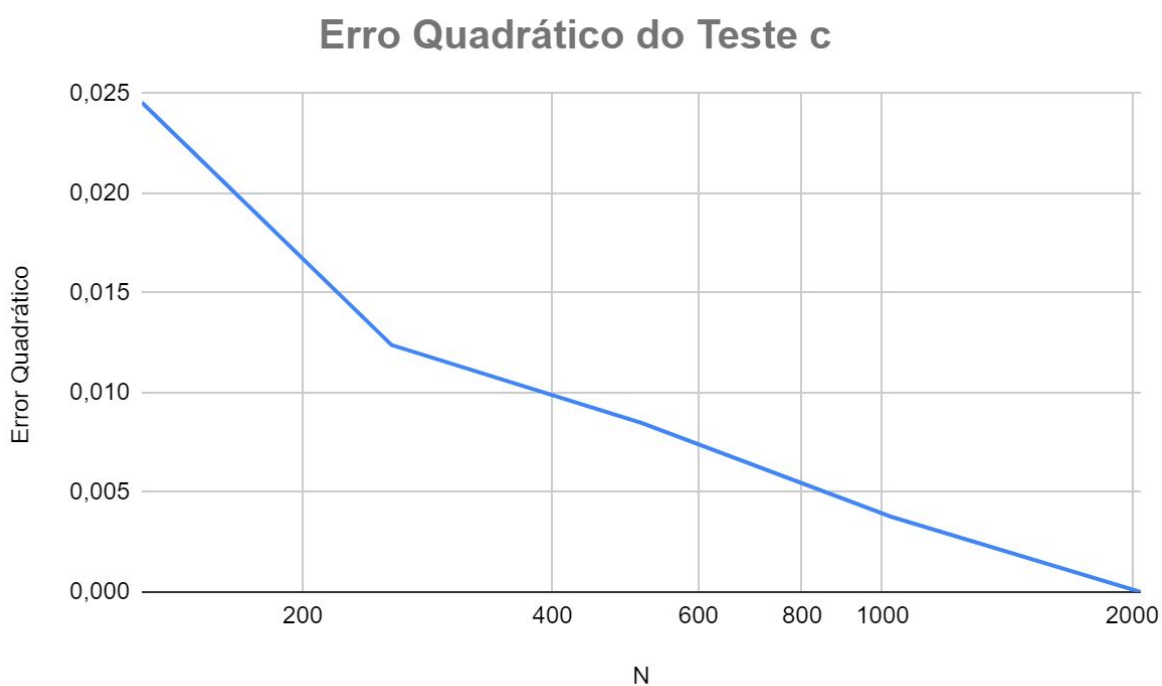


Gráfico 8 - Erro Quadrático do teste c em escala logarítmica

3.3.3 Discussão

Fazendo-se os cálculos, dos coeficientes a_0 a a_9 obtidos pelos valores dados para o teste c, percebe-se que com o aumento do número N o valores de a_0 a

a_9 tendem a variar, porém tendendo para um número, além de diminuir o erro quadrático resultante.

Além disso, percebe-se que até $N = 1024$, ao se dobrar o valor do N , o valor do erro diminui duas vezes, porém, ao se utilizar $N = 2048$, o erro diminui drasticamente, chegando a valores de ordem de grandeza muito pequenos, podendo se dizer que os coeficientes das intensidades das fontes pontuais foram praticamente recuperados. Dessa maneira com o $N = 2048$ podemos verificar o valores de a_0 a a_9 com o menor erro ($3.6214249335228157e-06$).

Olhando-se os gráficos também, pode-se ver picos de intensidade nas posições de cada uma das fontes, podendo-se ver como cada uma delas contribui para a distribuição final de temperatura.

3.4 Teste d

3.4.1 Dados do problema

Para o teste d se utilizou os mesmo dados do teste c, porém se teve a introdução de ruído nos dados. Para isso cada valor de $u_T(x_i)$ foi multiplicado por $(1 + r * \varepsilon)$ com $\varepsilon = 0.01$ e r um número randômico entre -1 e 1 .

3.4.2 Resultados

3.4.2.1 Gráficos

Para o teste em questão foram obtidos os seguintes gráficos de $u_T(x_i)$, para diferentes valores de N , sendo mostrado o valor medido e o valor calculado aproximado.

Distribuição de temperatura no instante T em função da posição

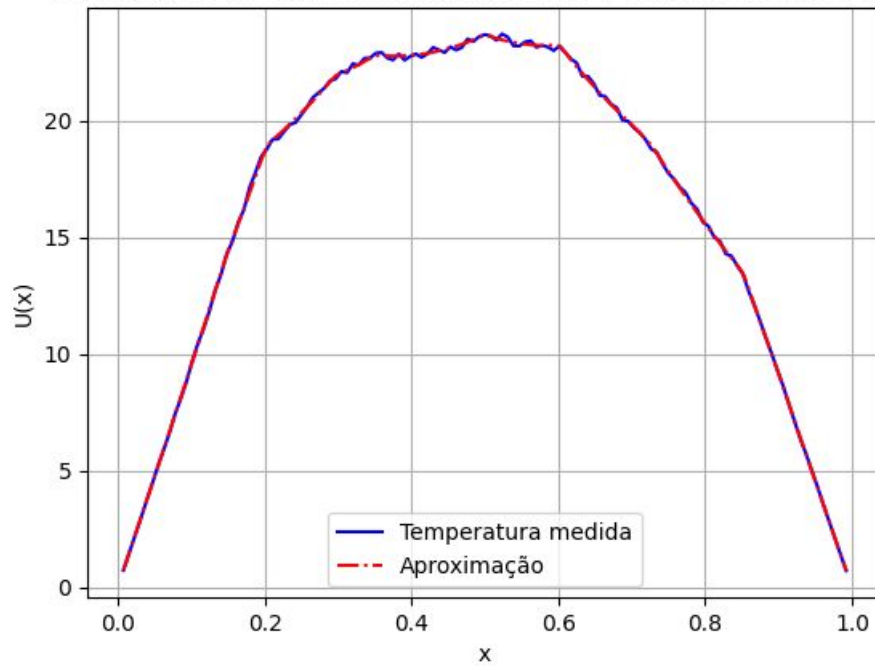


Gráfico 9 - Teste d com $N = 128$

Distribuição de temperatura no instante T em função da posição

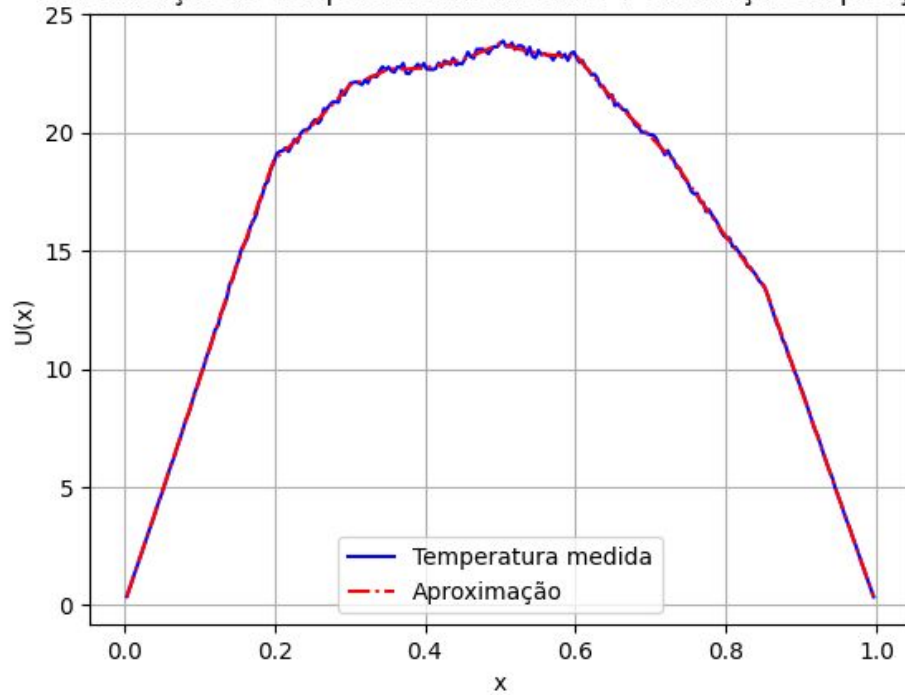


Gráfico 10 - Teste d com $N = 256$

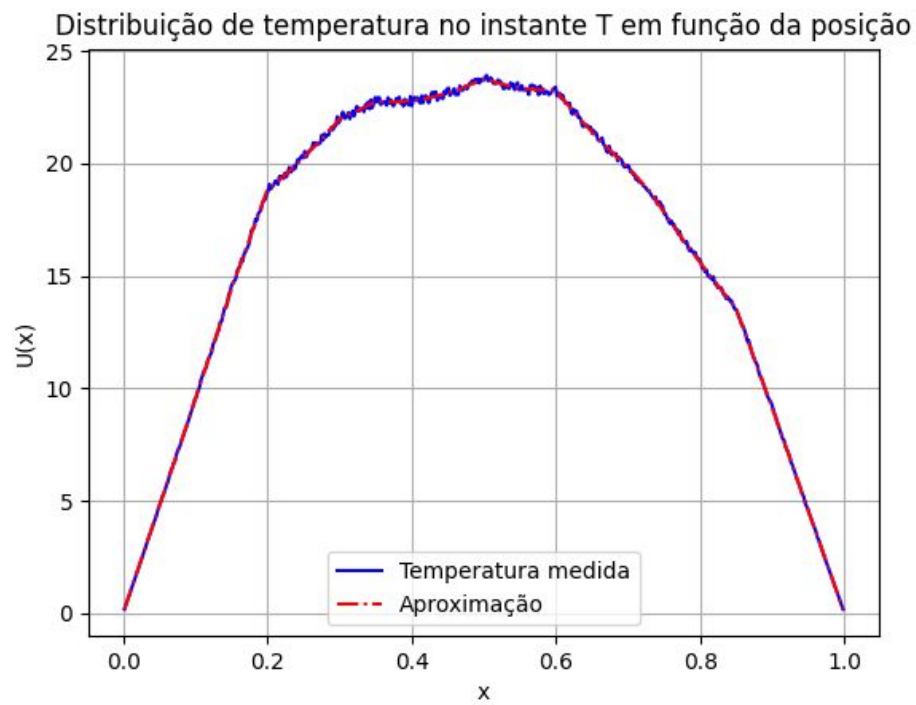


Gráfico 11 - Teste d com $N = 512$

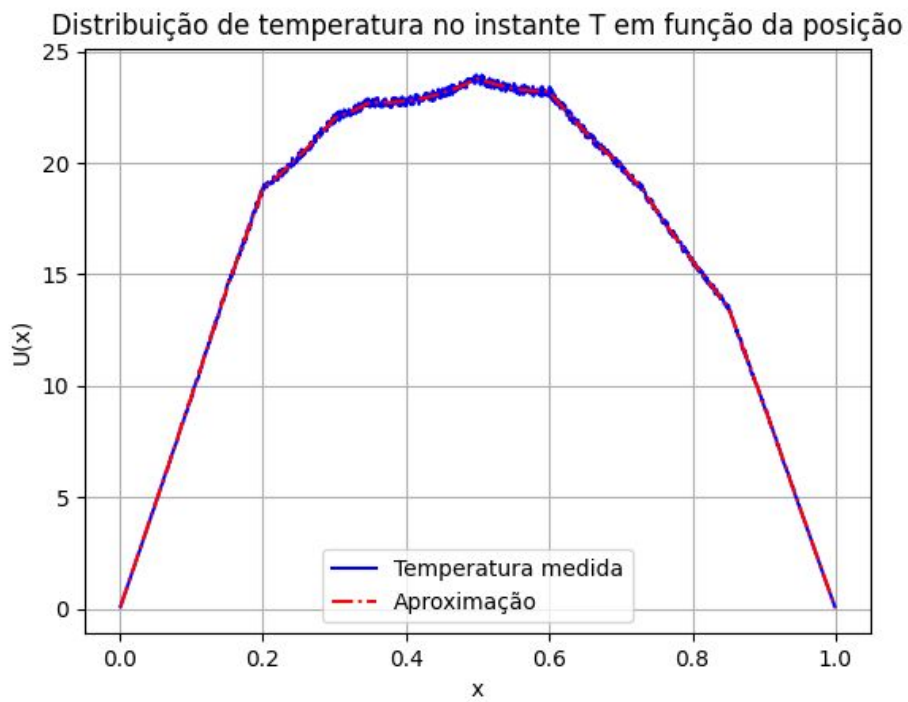


Gráfico 12 - Teste d com $N = 1024$

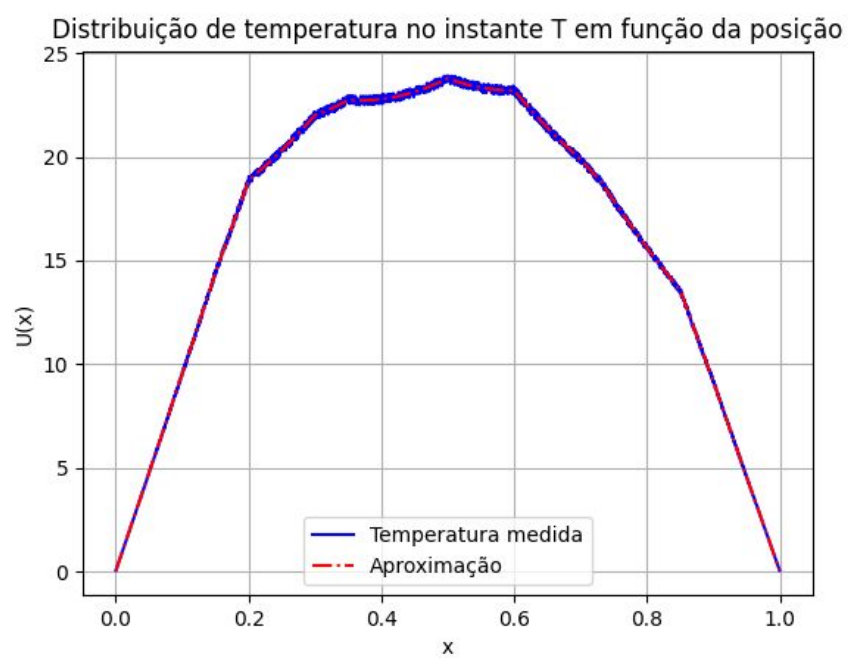


Gráfico 13 - Teste d com $N = 2048$

3.4.2.2 Coeficientes

Para o teste c foram obtidos os seguintes valores das intensidades para diferentes valores de N:

Intensidades	N = 128	N = 256	N = 512	N = 1024	N = 2048
a_0	1.204692765 0712487	0.837289338 0522903	0.987815091 9591802	1.0729881673 893633	0.96596529942 90262
a_1	4.769597419 108453	5.194408471 146296	5.012869427 378172	4.9408225387 772085	5.05838922823 357
a_2	1.882915262 8819643	2.069683839 298919	1.903014221 5216879	1.9655148921 169356	1.93752083953 06034
a_3	1.733843720 2982764	1.378896936 5607349	1.608617522 686563	1.5245601336 553776	1.53895063694 24133
a_4	2.006913671 3108247	2.195417367 4055353	2.139065658 2477465	2.2109704246 86664	2.21090122835 01555
a_5	3.277295694 5725347	3.229055599 70177	3.198746824 938859	3.0637753778 394687	3.05947586966 67244
a_6	0.442567152 8787132	0.369621728 4377943	0.609526628 8152521	0.7611281280 683329	0.68103518571 0432
a_7	1.340038528 6258274	1.453260447 050182	1.196061625 298448	1.1572746890 322403	1.24695644654 16578
a_8	4.023978825 8021045	3.993554776 12961	3.928029087 144553	3.9150923586 50008	3.91836154422 3834
a_9	0.421245170 09434365	0.361229678 7245346	0.508871153 0737177	0.4935130224 153365	0.48273808900 082815

Tabela 5 - Intensidades do teste d

3.4.2.3 Erro Quadrático

Para o teste c foram obtidos os seguintes valores de erros quadráticos para diferentes valores de N:

N	Erro Quadrático
128	0,09747232473795191
256	0,09942841368817505
512	0,09944421369275477
1024	0,10555015880944923
2048	0,10247095936636426

Tabela 6 - Erros quadráticos do teste d

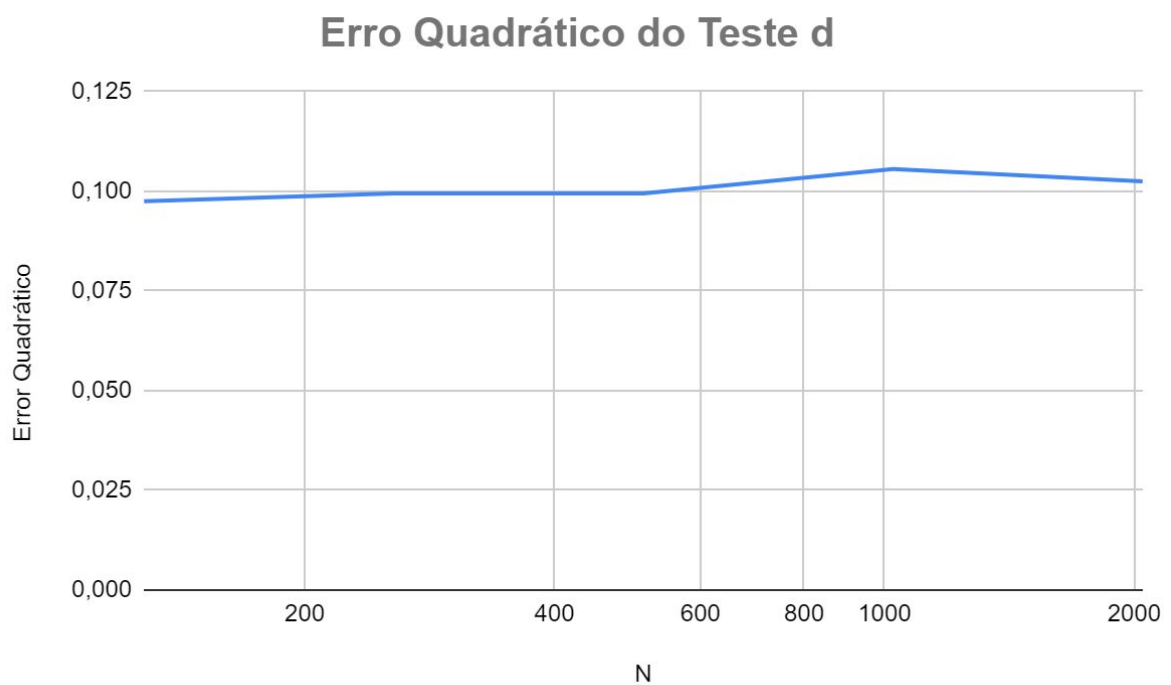


Gráfico 14 - Erro Quadrático do teste d em escala logarítmica

3.4.3 Discussão

Fazendo-se os cálculos para o teste d, percebe-se que os coeficientes a_0 a a_9 obtidos são muito próximos dos obtidos com o teste c, porém com um erro relativamente muito maior, sendo o maior erro do teste c quatro vezes maior que o menor erro do teste d.

Além disso, podemos verificar que ao contrário do que ocorre com o teste c, com o aumento do N o erro quadrático tende a aumentar, isso se deve principalmente devido ao acréscimo do ruído ao teste d, uma vez que com o aumento do N também há um aumento do ruído gerado.

Olhando-se os gráficos, é possível ver a forma como o ruído influencia na temperatura medida, fazendo-a oscilar, porém pode-se ver picos de intensidade nas posições de cada uma das fontes, assim como no teste c.

4. Conclusão

Após a implementação do exercício programa e observação do seu comportamento nos testes a, b, c, e d. Podemos concluir que é possível encontrar os valores dos coeficientes a_0 a a_9 através dos métodos dos mínimos quadráticos.

Além disso podemos observar pelos testes dos itens c que com o aumento o N temos uma diminuição do erro quadrático dos coeficientes, contudo pelo teste d podemos verificar que ao adicionarmos um ruído ao sistema, representando um erro na medição da temperatura, o erro quadrático sofre o comportamento contrário do esperado pelos teste c, uma vez que, com o aumento do N o erro também aumenta ligeiramente, por tanto devemos levar este fator em consideração, quando analisarmos o comportamento de da distribuição da temperatura no espaço.