ALUNOS
Lucas Oliveira de Paula – 10773680
Victor Nascimento Pereira – 10773530

Título

Relatório sobre um problema inverso para obtenção de distribuição de Temperatura (parte 2)

Versão Original

Relatório apresentado ao Instituto de Matemática e Estatística da Universidade de São Paulo como parte complementar ao EP2 de Métodos Numéricos e Aplicações (MAP-3121)

Área de concentração Métodos Numéricos

São Paulo 2020

SUMÁRIO

1. Introdução

Há problemas diretos, onde se determina o efeito da causa, e problemas inversos, onde a partir do efeito observado tenta-se apurar a causa. Este EP resolverá problemas inversos relacionados a equação do calor.

Utilizando de métodos computacionais sendo aplicados através da linguagem Python, e dispondo das bibliotecas permitidas, neste EP, tratou-se a obtenção de coeficientes que determinam a intensidade de fontes pontuais de calor em determinadas regiões da barra a partir dos valores finais de temperatura obtidos previamente (por medição, por exemplo).

2. Hipótese

Como temos um sistema com N (número de divisões da barra) equações e nf (número de fontes pontuais) incógnitas, esse é um sistema sobredeterminado, e para resolvê-lo devemos utilizar o método dos mínimos quadrados (MMQ), que tem como objetivo minimizar a equação do erro quadrático:

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

Para isso, deve ser utilizado sistema a seguir, com o produto interno apresentado na equação (4). Nota-se que para o cálculo do produto interno são ignorados o primeiro e último pontos (x = 0 e x = 1), pois foi adotado o valor de 0 para as duas fronteiras, logo, não precisam entrar na equação.

$$\begin{bmatrix} < u_1, u_1 > & < u_2, u_1 > & \cdots & < u_{nf}, u_1 > \\ < u_1, u_2 > & < u_2, u_2 > & \cdots & < u_{nf}, u_2 > \\ \vdots & \vdots & \vdots & \vdots \\ < u_1, u_{nf} > & < u_2, u_{nf} > & \cdots & < u_{nf}, u_{nf} > \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{nf} \end{bmatrix} = \begin{bmatrix} < u_T, u_1 > \\ < u_T, u_2 > \\ \vdots \\ < u_T, u_{nf} > \end{bmatrix} \tag{2}$$

3. Procedimento e funções do código fonte

As funções utilizadas no programa estão apresentadas e brevemente explicadas a seguir. Apenas seus propósitos, funcionalidades (de um modo geral) e particularidades/observações são ditas, desta forma, é necessário um conhecimento prévio de linguagem python para entendê-las.

Para fim de organização o trabalho foi desmembrado em diversos arquivos “.py”, em que cada um possui determinadas funcionalidades, o que facilita a implementação e modificações nos arquivos. O funcionamento se dá por importações de códigos “.py”, onde, por exemplo, o codigo testeep2.py abre com imports de ep2 e ep1, para realizar suas operações de forma conjunta e organizada.

3.1 . Função main

A função main se encontra no arquivo testeep2.py, ela é a porta de entrada do programa. A partir dela, o usuário pode digitar as entradas e escolher as opções para o funcionamento do programa. O seu código pode ser visto a seguir:

```
    from ep2 import ep2, learquivo
    from ep1 import fazx, fazt, metodo11, euler, crank



    if __name__ == "__main__":
       b = 1
       while b == 1 or b == 2 or b == 3 or b == 4:
     print("\n1-Teste do metodo 11\n2-Euler Implicito\n3-Crank Nicolson\
\n4 - Ep2 (problema inverso) \nDigite outro valor inteiro para sair\n")
           b = int(input("Qual teste deseja executar? - "))
           if b == 1 or b == 2 or b == 3:
               n = int(input("Escolha o valor de N: "))
             lmbda = float(input("Escolha o valor de lambda (Para deltat
       ser igual a deltax, lambda = n; Para inserir M, lambda = 0): "))
               if lmbda == 0:
                  m=int(input("Escolha o valor de M(multiplo de 10): "))
               else:
                   m = n ** 2 / lmbda
               if int(m) == m:
```

```
                m = m-1
            else:
                a = int(input("\nSelecione a função f(t,x) desejada:\
                    \n1: f(t,x) = 10x²(x-1)-60xt+ 20t\
                \n2:f(t,x)=10cos(10t)x²(1-x)²-(1+sin(10t))(12x²-12x+2)\
                    \n3: f(t,x) = 25t²e^(t-x) cos(5tx) - 10te^(t-x)
                                        sen(5tx) - 5xe^(t-x) sen(5tx)\
                \n4: f(t,x) = 10000(1-2t²)N (p = 0.25 e gh cte):  "))

                if a != 1 and a != 2 and a != 3 and a != 4:
                    print ("Teste Invalido")
                    break

                x = fazx(n)
                t = fazt(m)

                if b == 1:
                    metodo11 (x,t,a)
                elif b == 2:
                    euler (x,t,a)
                elif b == 3:
                    crank (x,t,a,0.25,1)

        elif b == 4:
            print(" \na - Teste com N =128, nf = 1 e p1 = 0.35\nb -
                Teste com N =128, nf = 4 e p = [0.15, 0.3, 0.7, 0.8]\
                \nc - Teste com arquivo txt e N a escolha \nd - Teste
                      com arquivo txt, N a escolha e ruido em Ut(x) \
                 \ne - Se deseja digitar nf, p, N e coeficientes ak")
            r = input("Qual teste deseja executar? - ")
            if r == 'a' :
                t = fazt (127)
                x = fazx (128)
                p = [0.35]
                ep2(x,t,1,p,r)

            elif r == 'b':
```

```
                t = fazt (127)
                x = fazx (128)
                p = [0.15, 0.3, 0.7, 0.8]
                ep2(x,t,4,p,r)

            elif r == 'c' or r == 'd':
                n = int(input("Escolha o valor de N: "))
                t = fazt (n-1)
                x = fazx (n)
                p = learquivo (0)
                ep2 (x,t,len(p), p, r)

            elif r == 'e':
                p = []
                nf = int(input("Quantos pontos p serão utilizados
                    (parametro nf)? - ")) # Recebe os pontos de fonte
                for i in range (0,nf):
                    pdigitado = float(input("digite a posição de pk
                                            (k = %d) = " % (i+1)))
                    p.append(pdigitado)
                n = int(input("Escolha o valor de N: "))
                t = fazt (n-1)
                x = fazx (n)
                ep2 (x,t,nf,p,r)
        else:
            print("Teste invalido")
```

Primeiramente o usuário escolherá o teste a ser executado, como os casos do ep1 já foram explicados no relatório anterior, estes serão considerados já compreendidos, ou seja, não adentrarmos nos casos de o usuário dar como entrada 1, 2 ou 3. Caso o usuário entre com o valor 4, será solicitado desde a escolha de qual teste deseja executar (os testes *a, b, c* e *d* mencionados encontram-se no enunciado do exercício programa).

Caso a:

É criado o vetor x (que representa a barra com N subdivisões Δx), utilizando a função fazx, com o valor sugerido para N no enunciado do ep2 (N = 128), e o vetor t

que representa o tempo com M subdivisões Δt), utilizando a função fazt, com valor M = N = 128. Como explicado no exercício programa anterior, a função fazt recebe o parâmetro M-1, por questões explicadas no relatório anterior que não vêm ao caso agora. Após, é criado o vetor de forçante pontual p, que para o teste a, é único com p = 0.35. Após estas tarefas preparadas, chama-se a função ep2, enviando os vetores x, t, o número de fontes pontuais (nf = 1), o vetor p, e o valor da variável r (r == a, no caso) para que a função ep2 trate com a funcionalidade do teste a .

Caso b:

Análogo ao caso a, porém, conta com mais forçantes pontuais, que caracterizam o vetor p. Para este teste, foram definidas as fontes pontuais $p_1 = 0.15$, $p_2 = 0.15$, $p_3 = 0.15$ e $p_4 = 0.8$, todas inseridas no vetor p. Além disso, o valor de nf, passado como parâmetro para a função ep2, vale 4, por haver 4 fontes pontuais no vetor p.

Casos c e d:

É colhido do usuário o valor N, que será o parâmetro de fazt e fazx neste caso. Para os forçantes pontuais p, utiliza-se a função learquivo, com o parâmetro enviado n = 0, que faz com que a função learquivo retorne apenas o vetor p que é lido do arquivo .txt fornecido. E então, assim como nos outros casos, é chamada a função ep2, com os parâmetros obtidos. Se o parâmetro r enviado for d, a função ep2 chamará também a função que transforma o vetor ut em um novo vetor ut com ruído, mas isto será explicado na respectiva função.

Caso e (adicional, não pedido no enunciado):

Foi desenvolvido para que o usuário digite as entradas desejadas para os parâmetros p, N e coeficientes ak. Neste caso, inicia-se criando o vetor p, a partir do número de fontes pontuais obtido pelo parâmetro nf digitado pelo usuário. Então colhe-se do usuário os valores para as posições pk das forçantes pontuais a serem utilizada. Adiante, é colhido do usuário o valor N, que será o parâmetro de fazt e fazx neste, e finalmente é chamada a função ep2 com os parâmetros atuais.

3.2 fazx

Esta função se encontra em ep1.py, e é usada para fazer o vetor x (posição na barra). O seu código pode ser visto a seguir:

```
def fazx(n):
    x = []
    for i in range(n + 1):
        x.append(i / n)
    return x
```

3.3 fazt

Esta função se encontra em ep1.py, e é usada para fazer o vetor t (tempo). Lembrando que para a lógica funcionar, como explicado no relatório do ep1, o valor m recebido é menor em uma unidade que o M digitado (no caso M recebido será igual a N-1). As funções do programa executam essa lógica automaticamente a partir do valor M digitado pelo usuário (o usuário deve digitar o valor de M que deseja), ou a partir do valor de N digitado ou pré-definido (para a funcionalidade “4 - Ep2 (problema inverso)”, pois para esta funcionalidade, N = M). Toda essa lógica existe para que a função fazt criasse um vetor com M sendo um número inteiro, caso o usuário digite o valor de λ nas funcionalidades 1, 2 e 3. O código da função fazt pode ser visto a seguir:

```
def fazt(m):
    t = []
    for i in range(int(m) + 2):
        t.append(i / (int(m)+1))
    return t
```

A partir destas funções, é criado o vetor x, que representa as posições espaciais na barra, que vai de 0 a 1, sendo $\Delta x = 1/N$, com N+1 valores (logo $x_N = 1$, partindo de x_0). E o vetor t (tempo), que vai de 0 a T = 1, onde $\Delta t = 1/M$, com M+2 valores (M=N-1, o que resulta em $t_0 = 0$, $t_k = k / (M+1)$ e $t_{M+1} = 1$). Relembrando, o valor do parâmetro M é criado automaticamente para esta tarefa do exercício programa 2.

3.4 ep2

Esta função se encontra em ep2.py, e organiza os cálculos a serem feitos. Recebe os parâmetros provenientes da função main para o tipo de teste desejado. A

função começa originando os vetores Uk no mesmo número de fontes pontuais existentes no vetor p (parâmetro nf). Para isso, a função deve chamar uma outra função, já utilizada no exercício programa anterior, porém modificada, a função crank. Esta nova função crank retornará os vetores u_k (u_k é o vetor de temperaturas $u(t_k, x_i)$ para a barra no tempo t = T = 1, e o índice k é o número da fonte pontual utilizada no cálculo do método de Crank-Nicolson) caso o usuário tenha escolhido a funcionalidade "4- Ep2 (problema inverso)". Após isto, são pedidos os valores dos coeficientes ak, caso o usuário tenha escolhido o teste *a,b* ou *e*. Com isso, o vetor u_T é criado a partir dos coeficientes inseridos e os vetores u_k.

$$u_T(x) = \sum_{k=1}^{nf} a_k u_k(T, x) \tag{3}$$

Se for escolhidos o teste c ou d, o vetor u_T é proveniente da função learquivo, com parâmetro N passado (N escolhido na função main caso o teste seja c ou d). Além disso, para o teste d, o vetor u_T, lido do arquivo .txt, é modificado pela função ruido.

Com os vetores u_T e os vetores u_k (k = 1 , 2… nf) em mãos, podemos desenvolver a matriz (2). Sendo A, matriz de coeficientes das incógnitas a_k ,composta pelos produtos internos entre os vetores u_k. E sendo B, a matriz de termos independentes, no segundo membro, composta pelos produtos internos entre vetores u_k e a solução u_T. Todos estes produtos internos são obtidos pela função produtointerno, com primeiro e último valor ignorados por serem 0 (condições de fronteira) em todos os vetores. Após, é chamada a função que resolverá o sistema da equação (2) e apresentará a solução para os valores dos coeficientes a_k provenientes do MMQ realizado, chamada de resolvenovox. Por fim, a última função chamada é e2, que calcula o erro quadrático (1) e apresenta ao usuário.

O código pode ser visto a seguir:

```
def ep2(x, t, nf, p, r):
    n = len(x) - 1
    uk = []
    coef = []
```

```
        ut = []
        matrizA = []
        vetorB = []

        for i in range(0, nf):              # Cria matriz com vetores Uk
            vetoru = []
            vetoru = crank(x, t, 4, p[i], 2)
            uk.append(vetoru)

        if r in ('a', 'b', 'e'):
            for i in range(0, nf):         # Cria vetor coef para vetor Ut
                cf=float(input("Digite o coeficiente de u%d - " % (i+1)))
                coef.append(cf)

            for i in range(0, n+1):             # Cria vetor Ut
                valorut = 0
                somaut = 0
                for j in range(0, nf):
                    valorut = coef[j] * uk[j][i]
                    somaut = somaut + valorut
                ut.append(somaut)
        elif r in ('c', 'd'):
            ut = learquivo(n)
            if r == 'd':
                ut = ruido(ut)

        # Faz matriz A (coeficientes) e vetor B (independentes)
        for i in range(0, nf):
            linhaA = []
            B = produtointerno(ut, uk[i])
            vetorB.append(B)
            for j in range(0, nf):
                A = produtointerno(uk[i], uk[j])
                linhaA.append(A)
            matrizA.append(linhaA)

        vetorx = resolvenovox(matrizA, vetorB)
```

```
        print('Valores de ak (vetor):')
        print(vetorx)
        e2(vetorx, ut, uk)
```

3.5 learquivo

Esta função se encontra em ep2.py, é a responsável pela leitura de arquivos e gravação de seus dados em vetores. O seu código pode ser visto a seguir:

```
def learquivo(n):
    v = []
    f = open('teste.txt', 'r')
    raw = f.read()
    f = raw.split('\n')
    aux = ''
    if n == 0:
        raw = f[0].split('       ')
        for i in raw:
            aux = ''
            for j in i:
                if j != ' ':
                    aux = aux+j
            v.append(np.float128(aux))
        return v
    for i in range(1, len(f)-1, int((len(f)-3)/n)):
        aux = ''
        for j in f[i]:
            if j != ' ':
                aux = aux+j
        v.append(np.float128(aux))
    return v
```

Nesta função, por início cria-se o vetor v, então é criado f, que recebe o arquivo sugerido, raw “lê” o arquivo, e passa a ter seu conteúdo, f então recebe o conteúdo

dividido por quebras de linha, e então é criado um auxiliar.

Caso o parâmetro passado para a função seja 0, esta estará criando um vetor p de forçantes pontuais, onde raw receberá a primeira linha do arquivo sugerido, subdividida entre os espaços. No laço, o índice toma os valores de raw, que serão percorridos em um novo laço, onde, quando o valor atual não for um espaço, ele é inserido no auxiliar, deixando o auxiliar com todos valores “validos”, para ser acoplado ao vetor v, que é retornado pela função.

Caso o parâmetro passado para a função for um N válido, esta tem um início de operação similar, lidando com o arquivo, porém, ignora a parte que faz o vetor p, e adentra um laço, que começa em 1 (ignorando a primeira linha, que seria índice 0, com os valores sugeridos para o vetor p), vai até a quantidade de linhas do arquivo - 1, pulando de quantidade de possíveis valores de N / n recebido como parâmetro. Neste laço, o auxiliar é esvaziado, então, em um laço são analisados os valores em cada linha de f, e os válidos são passado ao auxiliar, que é acoplado no vetor v, neste caso de valores das funções $u_T(x_i)$.

3.6 Função crank

Esta função se encontra em ep1.py, é responsável pelo cálculo das matrizes pelo método Crank-Nicolson. Ela será utilizada para a resolução do novo problema proposto. Para relembrar, o método de Crank-Nicolson consiste na resolução do sistema (4), mostrado a seguir, para cada iteração. O sistema provém da equação (5).

$$\begin{bmatrix} 1+\lambda & \frac{-\lambda}{2} & 0 & \dots & 0 \\ \frac{-\lambda}{2} & 1+\lambda & \frac{-\lambda}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \frac{-\lambda}{2} & 1+\lambda & \frac{-\lambda}{2} \\ 0 & \dots & 0 & \frac{-\lambda}{2} & 1+\lambda \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} = \begin{bmatrix} u_1^k+(u_0^k-2u_1^k+u_2^k)+\frac{\Delta t}{2}(f_1^k+f_1^{k+1})+\frac{\lambda}{2}g_1^{k+1} \\ u_2^k+(u_1^k-2u_2^k+u_3^k)+\frac{\Delta t}{2}(f_2^k+f_2^{k+1}) \\ \vdots \\ u_{N-2}^k+(u_{N-3}^k-2u_{N-2}^k+u_{N-1}^k)+\frac{\Delta t}{2}(f_{N-2}^k+f_{N-2}^{k+1}) \\ u_{N-1}^k+(u_{N-2}^k-2u_{N-1}^k+u_N^k)+\frac{\Delta t}{2}(f_{N-1}^k+f_{N-1}^{k+1})+\frac{\lambda}{2}g_2^{k+1} \end{bmatrix} \quad (4)$$

$$u_i^{k+1} = u_i^k + \frac{\lambda}{2}((u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}) + (u_{i-1}^k - 2u_i^k + u_{i+1}^k)) + \frac{\Delta t}{2}(f(x_i, t_{k+1}) + f(x_i, t_k)) \quad (5)$$

Para isso, devemos desenvolver as matrizes de coeficientes (matriz A, não é a mesma apresentada anteriormente, é apenas um nome genérico) e o vetor de termos independentes (vetor b, assim como A, não é o mesmo da função ep2). Para resolução deste sistema linear ainda utiliza-se duas outras funções, a ldl e a resolvex. A diferença consiste na adição dos parâmetros recebidos: p, que no exercício programa anterior era um ponto fixo em 0,25, agora é variável e recebido pela função. O outro novo parâmetro é v, que pode ser 1, para a funcionalidade do exercício programa 1, ou v pode ser 2, que utiliza outra funcionalidade: faz a função crank retornar o vetor u para o instante final (t = 1), ou seja, o ultimo vetor da matriz u desenvolvida pelo método de Crank-Nicolson. Além disso, o parâmetro v faz com que o método utilize a função r(t) correspondida (para v=1: r(t) = 10000(1 − 2²), para v=2: r(t) = 10(1 +cos(5t))). Assim como no exercício programa anterior, o parâmetro a deve ser igual a 4, para a funcionalidade de função pontual, o que diferencia entre os problemas propostos é o parametro v já explicado. A nova função crank pode ser vista a seguir:

```
def crank(x, t, a, p, v):
    n = len(x) - 1
    m = len(t) - 1
    lmbda = n*n/m
    u = []
    linha = []
    principal = []
    secundaria = []
    b = []
    vetorx = []
    uo = 0
    meiodeltax = x[1] / 2

    for j in range(0, n-1):                    # Matriz A
        principal.append(1 + lmbda)
```

```
            if j != n-2:
                secundaria.append(-lmbda/2)

        for k in range(0, m+1):
            b = []
            vetorx = []
            if k == 0:
                if a == 1 or a == 4:
                    linha.append([0] * (n+1))
                    u = linha
                    progressbar1(len(u)-1, len(t), 'Calculando')
                    plt.plot(linha, 'k', color='orange')
                elif a == 2:
                    for j in range(n+1):
                        uo = x[j] * x[j] * (1 - x[j]) * (1 - x[j])
                        linha.append(uo)
                    u.append(linha)
                    progressbar1(len(u)-1, len(t), 'Calculando')
                    plt.plot(linha, 'k', color='orange')
                elif a == 3:
                    for j in range(n+1):
                        uo = math.exp(-x[j])
                        linha.append(uo)
                    u.append(linha)
                    progressbar1(len(u)-1, len(t), 'Calculando')
                    plt.plot(linha, 'k', color='orange')

            else:
                linha = []
                if a == 1 or a == 2 or a == 4:
                    g1 = 0
                    linha.append(g1)
                elif a == 3:
                    g1 = math.exp(t[k])
                    linha.append(g1)
                if a == 1 or a == 2 or a == 4:
                    g2 = 0
```

```
                elif a == 3:
                    g2 = math.exp(t[k] - 1) * math.cos(5 * t[k])

                for i in range(1, n):                          # Matriz
B
                    if a == 1:
                        f1 = (10 * x[i] ** 2 * (x[i] - 1) -
                              60 * x[i] * t[k-1] + 20 * t[k-1])
                        f2 = (10 * x[i] ** 2 * (x[i] - 1) -
                              60 * x[i] * t[k] + 20 * t[k])
                        f = f1 + f2
                    elif a == 2:
                        c1 = (10 * math.cos(10 * t[k-1]) * x[i]
                              * x[i] * (1 - x[i]) * (1 - x[i]))
                        d1 = (1 + math.sin(10 * t[k-1])) * \
                            (12 * x[i] * x[i] - 12 * x[i] + 2)
                        f1 = c1 - d1
                        c2 = (10 * math.cos(10 * t[k]) * x[i]
                              * x[i] * (1 - x[i]) * (1 - x[i]))
                        d2 = (1 + math.sin(10 * t[k])) * \
                            (12 * x[i] * x[i] - 12 * x[i] + 2)
                        f2 = c2 - d2
                        f = f1 + f2
                    elif a == 3:
                        b1 = 25*t[k]*t[k] * (math.exp(t[k]-x[i])) * \
                            (math.cos(5*t[k]*x[i]))
                        c1 = 10*t[k] * (math.sin(5*t[k]*x[i]))
                        d1 = 5*x[i] * (math.exp(t[k]*x[i])) * \
                            (math.sin(5*t[k]*x[i]))
                        f1 = b1 - c1 - d1
                        b2 = 25*t[k-1]*t[k-1] * \
                          (math.exp(t[k-1]-x[i])) * (math.cos(5*t[k-1]*x[i]))
                        c2 = 10*t[k-1] * (math.sin(5*t[k-1]*x[i]))
                        d2 = 5*x[i] * (math.exp(t[k-1]*x[i])) * \
                            (math.sin(5*t[k-1]*x[i]))
                        f2 = b2 - c2 - d2
                        f = f1 + f2
```

```
                elif a == 4:
                    if v == 1:
                        if x[i]<= (p+meiodeltax) and x[i]>=
                                                    (p-meiodeltax):
                            f1= 10000*(1- 2* t[k-1]* t[k-1])* (1/x[1])
                            f2= 10000* (1- 2 * t[k] * t[k]) * (1/x[1])
                             f = f1 + f2
                        elif x[i]< (p- meiodeltax) or x[i]>
                                                   (p+ meiodeltax):
                             f = 0
                    elif v == 2:
                        if x[i]<= (p+ meiodeltax) and x[i]>=
                                                    (p-meiodeltax):
                             f1 = (10 + 10*math.cos(5*t[k-1])) / x[1]
                             f2 = (10 + 10*math.cos(5*t[k])) / x[1]
                             f = f1 + f2
                        elif x[i] < (p- meiodeltax) or x[i]>
                                                    (p+meiodeltax):
                             f = 0
                if i == 1:
                  valorb=((lmbda/2) *(u[k-1][i-1]-2*u[k-1][i]+
                u[k-1][i+1]))+ (u[k-1][i])+ (f/(2*m))+ ((lmbda/2)*g1)
                elif i == (n-1):
                  valorb=((lmbda/2)* (u[k-1][i-1]-2*u[k-1][i]
               +u[k-1][i+1]))+ (u[k-1][i])+ (f/(2*m))+ ((lmbda/2)*g2)
                else:
                  valorb=((lmbda/2)* (u[k-1][i-1]-2*u[k-1][i]+
                              u[k-1][i+1])) + (u[k-1][i]) + (f/(2*m))
                b.append(valorb)

            l, d = ldl(principal, secundaria, n)
            vetorx = resolvex(l, d, n, b)
            for j in range(0, len(vetorx)):
                linha.append(vetorx[j])
            linha.append(g2)
            u.append(linha)
            progressbar1(len(u)-1, len(t), "Calculando")
```

```
        if v == 1:
            calculatrunc(t, x, u, a, 3)
        elif v == 2:
            return u[m]
```

3.7 Função ruido

Esta função se encontra em ep2.py, e é responsável pela geração pseudo aleatória de distorções para os calores das funções u_T. Isso consiste na soma de cada valor do vetor u_T por um fator que é igual a 1% do valor de u_T em questão multiplicado pela variável chamada ruido (esta variável é um valor pseudo aleatória entre -1 e 1). O código pode ser visto a seguir:

```
def ruido(ut):
    for i in range(0, len(ut)):
        ruido = (random.random()-0.5)*2
        ut[i] = ut[i] + ut[i]*ruido*0.01
    return ut
```

3.8 produtointerno

Esta função é encontrada em ep2.py, e é responsável por realizar o cálculo do produto interno entre vetores passados como parâmetros. Vale lembrar que o produto interno é calculado ignorando o primeiro e último valor dos vetores (pois estes são iguais a 0). O código pode ser visto a seguir:

```
def produtointerno(vetoru, vetorv):
  n = len(vetoru)-1
  soma = 0
  for i in range(1, n):
     produto = vetoru[i]*vetorv[i]
     soma = soma + produto
  return soma
```

3.9 resolvenovox

Poderíamos ter reutilizado a função resolvex desenvolvida no exercício programa anterior, porém ela está sendo utilizada na função crank, e entendemos que

seria de mais proveito criar uma nova função. Esta nova função, que também se encontra em ep2.py, é responsável por calcular o novo vetor x (este vetor x é a solução para as incógnitas a_k, não tendo relação com o vetor x que dita as posições da barra. O nome vetor x para este vetor é apenas genérico, como a matriz A e o vetor b apresentados anteriormente). A diferença para esta nova função resolvenovox é que o parâmetro L (minúsculo no código) está em forma de matriz, diferente da função anterior que possui o parâmetro L em forma de vetor. A matriz L e o vetor d são provenientes da função novaldl que é chamada no início. O código da função resolvenovox pode ser visto a seguir:

```
def resolvenovox(A, b):

    n = len(A)
    l, d = novaldl(A)
    y = []
    z = []
    x = []

    y.append(b[0])
    for i in range(1, n):
        somay = 0
        for j in range(0, i):
            somay = somay + l[i][j]*y[j]
        y.append(b[i] - somay)

    for i in range(0, n):
        z.append(y[i]/d[i])

    for i in range(0, n):
        x.append(0)
    x[n-1] = z[n-1]
    for i in range(1, n):
        somax = 0
        for j in range(n-i, n):
            somax = somax + l[j][n-i-1]*x[j]
        x[n-i-1] = z[n-i-1] - somax
```

```
        return x
```

3.10 Função e2

Esta função se encontra em ep2.py, sua responsabilidade é calcular o erro quadrático para cada teste realizado. O erro quadrático é apresentado na equação (1) e está descrito em forma de linguagem python na função e2. O código pode ser visto a seguir:

```
def e2(ak, ut, uk):

  somaut = 0
  for i in range(0, len(ut)):
    somauk = 0
    for k in range(0, len(ak)):
      somauk = somauk + ak[k]*uk[k][i]
    somaut = somaut + (ut[i] - somauk)*(ut[i] - somauk)

  print('Erro quadrático =', f'{(math.sqrt(somaut / (len(ut)-1))):.2e}')
```

3.11 Função progressbar1

Esta função se encontra em ep1.py, veio da necessidade de alguma interação com o programa durante etapas de cálculos duradouras. O seu código pode ser visto a seguir:

```
def progressbar1(tamu, tamt, f="Progresso"):
  nt = tamt-1
  porcentagem = (tamu/nt*100)
  t = (nt)/10
  v = [0, t, 2*t, 3*t, 4*t, 5*t, 6*t, 7*t, 8*t, 9*t, nt]
  if tamu in v:
    s = f + ": [" + ('#'*int(porcentagem)) + (" "*int(100-porcentagem))
                  + '] ' + str('%d' % (porcentagem)) + '% concluido'
    print(s)
```

Trata-se de uma função para mostrar a porcentagem concluída de laços, ela recebe como parâmetros o tamanho de 2 vetores, tamu é o tamanho do vetor que está sendo calculado e tamt o tamanho do vetor usado de base dos cálculos, nt recebe tamt - 1, porcentagem recebe a comparação do tamanho dos vetores, t recebe a proporção de 1/10 (um décimo) do tamanho nt, v é um vetor com 10 (dez) posições equidistantes predefinidas de nt. Então, se o vetor sendo calculado tem o tamanho de uma das posições predefinidas em v, é mostrada na tela uma barra de progresso, com sua porcentagem e indicação visual adequada.

3.12 Função novaldl

Esta equação se encontra em ep2.py, e retorna o novo sistema tridiagonal simétrico que é utilizado para resolução do sistema (2). A diferença para a função ldl que foi desenvolvida no exercício programa 1 é que esta função retorna o parâmetro L em forma de matriz, como dito em 3.9. O seu código pode ser visto a seguir:

```
def novaldl(A):
    n = len(A)
    d = []
    linhal = []
    l = []
    v = []

    for i in range(0, n):            # cria matriz l
        linhal = []
        for j in range(0, n):
            if j == i:
                linhal.append(1)
            else:
                linhal.append(0)
        l.append(linhal)

    for i in range(0, n):                    # cria vetor d
        somad = 0
        v = []
        for j in range(0, i):
            v.append(l[i][j]*d[j])
```

```
            somad = somad + v[j]*l[i][j]
        d.append(A[i][i]-somad)
        for j in range(i+1, n):        # modifica matriz l
            somal = 0
            for k in range(0, i):
                somal = somal + l[j][k]*v[k]
            l[j][i] = (A[j][i] - somal) / d[i]

    return (l, d)
```

4. Resultados dos testes e dados

4.1 Teste a

Com N = 128, nf = 1, $p_1 = 0.35$, $u_T(x\ i\ ) = 7\ u_1(T, x_i)$ e $a_1 = 7$:

Valores de ak (vetor):

[6.999999999999998]

Erro quadrático = 2.05e-15

Como o parâmetro a_1 é digitado pelo usuário, podemos adotar qualquer um diferente de 7, mas para o teste do enunciado, foi sugerido este valor

4.2 Teste b

Com N = 128, nf = 4, $p_1 = 0.15$, $p_2 = 0.3$, $p_3 = 0.7$ e $p_4 = 0.8$ e $u_T(x_{i\ }) = 2.3\ u_1(T, x_i) + 3.7\ u_2(T, x_i) + 0.3\ u_3(T, x_i) + 4.2\ u_4(T, x_i)$:

Valores de ak (vetor):

[2.300000000000013, 3.6999999999999815, 0.30000000000003046, 4.1999999999999735]

Erro quadrático = 8.55e-15

O teste b é análogo ao teste a, porém com maior número de fontes e, consequentemente, mesmo número de parâmetros a_k . Lembrando que o teste e (adicionado) é análogo aos testes a e b, porém com maior controle sobre os parâmetros, por isso não será utilizado como comparações de resultados.

4.3 Teste c

Com $\Delta x = 1/N$ e N = 128:

Valores de ak (vetor):

[1.2091231792046620453, 4.839258715746566398, 1.8872408557564149013, 1.5833999318644641913, 2.214504046287935124, 3.1212947787768608099, 0.3773402863716625831, 1.4923482881226575172, 3.9751388015997336028, 0.40414515364777024739]

Erro quadrático = 2.45e-02

Com $\Delta x = 1/N$ e N = 256:

Valores de ak (vetor):

[0.9045010343181617528, 5.077572635561159932, 2.1008535954774862544, 1.4141556850900282635, 2.229245013052500053, 3.104613856992625852, 0.5094525973879268448, 1.3865087904608050543, 3.9498786461511186672, 0.41489312833156495827]

Erro quadrático = 1.24e-02

Com $\Delta x = 1/N$ e N = 512:

Valores de ak (vetor):

[0.9286883784929526684, 5.053707844480655846, 2.043701048904807402, 1.4676706728624837987, 2.1967633320010850105, 3.0911311688978230822, 0.6375875163901560229, 1.2716872153134484444, 3.878094867330631777, 0.53055677864216620415]

Erro quadrático = 8.48e-03

Com $\Delta x = 1/N$ e N = 1024:

Valores de ak (vetor):

[1.0072813220752491678, 4.992443012455015438, 1.9858767276132414832,

1.5132584652261398192, 2.1926928376831849397, 3.0951528759327473542, 0.6523266477836633559, 1.2537898890592238839, 3.8796670569411461982, 0.5297366253017004965]

Erro quadrático = 3.78e-03

Com $\Delta x = 1/N$ e N = 2048:

Valores de ak (vetor):

[0.9999999999931278097, 5.000000000017139335, 2.000000000009534323, 1.5000000000532518751, 2.2000000000521942496, 3.1000000000324374008, 0.6000000000347936093, 1.3000000000090322205, 3.9000000000275843524, 0.5000000000045434119]

Erro quadrático = 2.65e-12

A partir dos resultados, pode-se notar a convergência à medida que o valor de N cresce. E consequentemente, o erro quadrático diminui.

4.4 Teste d

Com $\Delta x = 1/N$ e N = 128:

Valores de ak (vetor):

[1.1117887329746769902, 4.964215153295211496, 1.9568063653138766163, 1.3959831152123856682, 2.3088115110303021873, 3.1261909051905786036, 0.6321923173436257296, 1.182998983940126151, 4.1468749379837264554, 0.26312506012649376812]

Erro quadrático = 9.32e-02

Com $\Delta x = 1/N$ e N = 256:

Valores de ak (vetor):

[0.9845983807886308989, 5.0512195537121481817, 1.9656801023854950563, 1.4936570443064469925, 2.2560500327957040126, 3.0208724687680328911, 0.8751316600493046008, 1.0589817207642474323, 3.9751006537933548347, 0.4240967317848102315]

Erro quadrático = 1.10e-01

Com $\triangle x = 1/N$ e N = 512:

Valores de ak (vetor):

[1.077693673725991173, 4.8257122791313995305, 2.1920612346153448419, 1.4136355147110694734, 2.2212310421509654174, 2.9892215484985582186, 0.87710172322478540 58, 1.0930947717951377653, 3.891163130782313075, 0.52344439228282688934]

Erro quadrático = 1.03e-01

Com $\triangle x = 1/N$ e N = 1024:

Valores de ak (vetor):

[0.9371714549824310056, 5.091685101509784244, 1.9069766530004193528, 1.5822936647775575625, 2.1394467698743158896, 3.119863557317657746, 0.7486864056700307966, 1.1346979599158907295, 3.9055269098955810148, 0.53439909704175039384]

Erro quadrático = 1.00e-01

Com $\triangle x = 1/N$ e N = 2048:

Valores de ak (vetor):

[0.9813082831563020459, 5.0142416100798941903, 2.0154801329418211859, 1.4694994939675549914, 2.2217457026025135749, 3.0934781491632144267, 0.64241020465365336264, 1.241615677291772829, 3.9222593145771244788, 0.48970695333238772735]

Erro quadrático = 1.03e-01

Como esperado, a adição dos ruídos fez com que a obtenção dos resultados reais fosse prejudicada. Isso é notado pelo aumento do erro quadrático para cada valor de N e pela diferença entre os valores encontrados para N = 2048 no teste c (teste em que os coeficientes apresentaram-se na forma mais próxima aos reais).

5. Comparações e conclusões

A partir dos dados apresentados anteriormente e pelo gráfico da figura 01 exemplificado a seguir, podemos notar o desprezível erro quadrático obtido e a proximidade dos valores obtidos pelo vetor a_k com os valores digitados. Isso comprova a eficiência do programa em linguagem python desenvolvido e, além disso, a eficiência do método dos mínimos quadrados (MMQ) utilizado para a obtenção dos parâmetros.

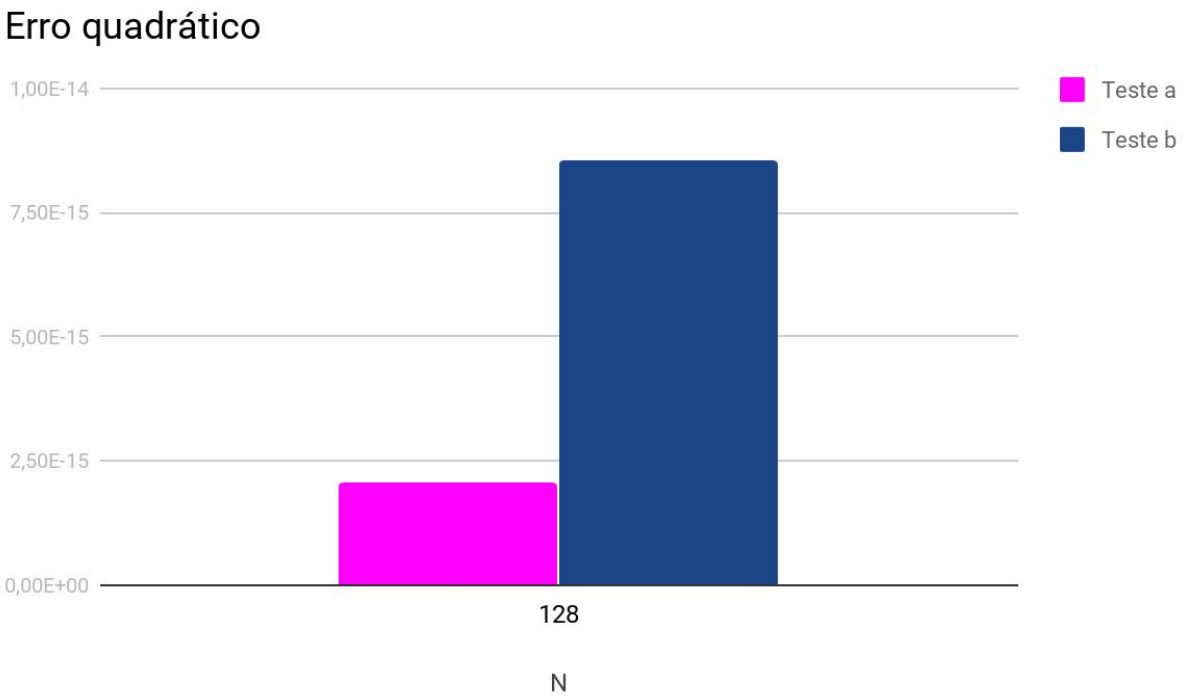

figura 01 - Gráfico do erro quadrático para os testes a e b

Para os testes c e d, podemos notar, a partir do gráfico da figura 02, a diminuição do erro quadrático principalmente para o teste c, onde não há ruídos. Diferente dos testes a e b, os valores de $u_T(x_i)$ não foram obtidos a partir de uma combinação linear dos valores $u_k(x_i)$ com coeficientes digitados pelo usuário, logo, não sabemos com exatidão os valores destes coeficientes a_k , porém, com N = 2048 para o teste c, obtivemos valores para esses coeficientes de uma forma aceitavelmente próxima, principalmente pelo desprezível erro quadrático, como nos testes a e b. Para

o teste d, a adição dos ruídos simula uma situação real onde há ruídos na medição da temperatura, o que claramente prejudica a obtenção dos coeficientes a_k com muita exatidão. Já era esperado o aumento do erro quadrático, porém ainda podemos considerá-lo aceitável. Temos o maior erro relativo entre os coeficientes obtidos no teste c e d (com N=2048) de 6.60% (para a_k com k = 7), o que nos mostra a eficiência do método mesmo com a adição do ruído. O gráfico para o erro quadrático dos testes c e d mencionado anteriormente pode ser visto a seguir na figura 02.

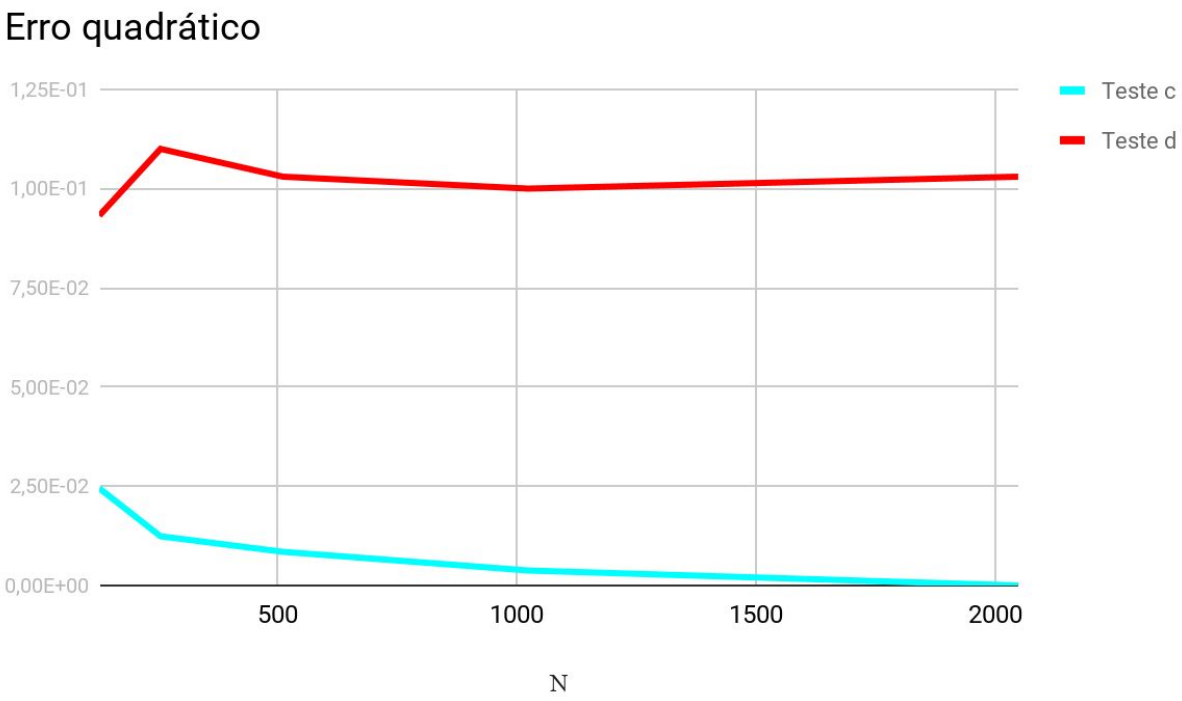

figura 02 - Gráfico do erro quadrático para os testes c e d em função de N