Universidade de São Paulo
Escola Politécnica

MAP3121: Métodos Numéricos e Aplicações

Exercício Programa 2

Turmas 05 e 06: Professor Claudio Hirofume Asano

Guilherme Fernandes Alves - 10774361
Ricardo Aguiar de Andrade - 10774674

São Paulo, Junho de 2020

Sumário

Lista de Tabelas

Lista de Figuras

1 Resumo, objetivos e considerações iniciais

O segundo exercício programa (EP) da disciplina *MAP3121 - Métodos Numéricos e Aplicações* se refere à resolução de um **problema inverso** ligado à equação do calor. O problema se trata da determinação de intensidades de fontes de calor aplicadas em posições conhecidas de uma barra, a partir de uma determinada distribuição final de temperatura. Este relatório consistirá de uma análise dos resultados obtidos para diferentes distribuições finais, bem como para diferentes fontes de calor a partir de parâmetros que serão inseridos pelo usuário.

Todos os resultados serão gerados por um programa criado, em linguagem de programação *Python 3*, pela dupla que vos escreve. O programa foi desenvolvido com base na teoria descrita no enunciado deste EP e do EP anterior, assim como nos conhecimentos da dupla adquiridos ao longo da disciplina e além desta. O software é totalmente autêntico, sendo reconhecido todo o auxílio dos monitores da disciplina, bem como as informações retiradas do enunciado do EP.

Para a realização do programa foram utilizadas funções inerentes à linguagem *Python* e bibliotecas permitidas pela disciplina, tais como *numpy* e *matplotlib*. Também houve a utilização de uma variável global, *'n'*, visando uma compactação e facilidade de entendimento no código fonte, pois caso essa variável não fosse global, algumas funções presentes no software teriam argumentos inutilizáveis, visto que dependendo do teste a ser realizado, *'n'* pode ou não ser necessária para os cálculos em determinadas funções. Por fim, o software foi escrito no ambiente de desenvolvimento integrado (IDE) *PyCharm*, em versão comunitária.

Este relatório tem como objetivos interpretar e analisar os resultados obtidos a partir do exercício proposto, com ênfase na resolução de um problema de mínimos quadrados em diferentes situações propostas pelo enunciado. Ademais, busca-se avaliar a evolução do erro quadrático da solução do problema ao passo que se refina a malha utilizada para representar a barra.

2 Fundamentos teóricos

Conforme enunciado do EP, a evolução da distribuição de temperatura em uma barra é dada pela seguinte equação diferencial parcial:

$$u_t(t,x) = u_{xx}(t,x) + f(t,x) \;\; em \;\; [0,T] \times [0,1] \tag{1}$$

Sendo as condições de contorno dadas por:

$$\begin{aligned} u(0,x) &= u_0(x) \;\; em \;\; [0,1] && (2)\\ u(t,0) &= g_1(t) \;\; em \;\; [0,T] && (3)\\ u(t,1) &= g_2(t) \;\; em \;\; [0,T] && (4) \end{aligned}$$

Onde t é a variável temporal e x a variável espacial. Usa-se a seguinte notação compacta para as derivadas parciais:

$$u_{xx}(t,x) = \frac{\partial^2 u(t,x)}{\partial x^2}$$

O comprimento da barra foi normalizado para 1 e integrar-se-á a equação num intervalo de tempo de 0 a T. A variável $u(t,x)$ descreve a temperatura no instante t e na posição x. Nota-se que as condições de contorno (3) e (4) denotam-se condições de fronteira, enquanto a condição (2) trata-se da temperatura no instante inicial para cada posição x. A função f descreve as fontes de calor ao longo do tempo.

2.1 Método implícito

Em um método implícito, a solução em um ponto de malha no novo instante depende não só de uma combinação de valores vizinhos do passo anterior, mas também de outros valores no mesmo instante, tornando-se necessário a solução de um sistema de equações a cada passo no tempo. Um exemplo disso é o método de Crank-Nicolson, que é incondicionalmente estável, mas tem convergência de ordem 2 em Δx e Δt. O método de Crank-Nicolson é dado por:

$$\begin{aligned} u_i^{k+1} &= u_i^k + \frac{\lambda}{2}((u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}) + (u_{i-1}^k - 2u_i^k + u_{i+1}^k)) + \frac{\Delta t}{2}(f(x_i, t_k + f(x_i, t_{k+1})), \\ & i = 1, \cdots, N-1, \;\; k = 0, \cdots, M-1 \end{aligned} \tag{5}$$

Para este método implícito, resolve-se um sistema linear com uma matriz tridiagonal simétrica a cada passo:

$$\begin{bmatrix} 1+\lambda & -\frac{\lambda}{2} & 0 & \cdots & 0 \\ -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} \\ 0 & \cdots & 0 & -\frac{\lambda}{2} & 1+\lambda \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ u_{N-2}^{k+1} \\ u_{N-1}^{k+1} \end{bmatrix} = \tag{6}$$

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

As análises de convergência do método de Crank-Nicolson pode ser encontrada no enunciado deste EP e no livro **Analysis of Numerical Methods, E. Isaacson and H. B. Keller** não sendo repetidas aqui.

2.2 Problema inverso

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

Mais precisamente, seja $u_T(x) = u(T,x)$ a solução da equação do calor dada por (1)-(4), com condições iniciais e de fronteiras nulas ($u_0(x) = g_1(t) = g_2(t) = 0$), com termo forçante da forma:

$$f(t,x) = r(t)\sum_{k=1}^{nf} a_k g_h^k(x), \tag{7}$$

com $g_h^k(x)$ sendo forçantes pontuais em $0 < p_1 < ... < p_{nf} < 1$ ao longo da barra, dadas por

$$g_h^k(x) = \frac{1}{h},\ \ se\ \ (p-\frac{h}{2}) \leq x \leq (p+\frac{h}{2}),\ \ e\ \ g_h(x) = 0\ \ caso\ contrario,\ \ com\ \ h = \Delta x \tag{8}$$

Obs.: nf é o número de fontes pontuais que o problema possui.

A função $r(t)$ descreve uma variação temporal das forçantes e os coeficientes a_k as respectivas intensidades. Nosso problema será a determinação das intensidades a_k a partir do conhecimento de $u_T(x)$ em pontos de uma malha

$$x_i = i\Delta x,\ \ i = 0, \cdots, N,\ \ com\ \ \Delta x = 1/N. \tag{9}$$

Para tanto iremos inicialmente determinar funções $u_k(t,x)$, $k = 1, \cdots, nf$, soluções de (1)-(4), com forçantes $f_k(t,x) = r(t)g_h^k(x)$ respectivamente. Devido à linearidade das equações (com condições iniciais e de contorno nulas), teremos necessariamente que:

$$u_T = \sum_{k=1}^{nf} a_k u_k(T,x). \tag{10}$$

Verificação de (10)

Demonstração. Partindo de (1) e considerando $t = T$, temos:

$$u_T(T,x) = u_{xx}(T,x) + f(T,x)$$

Substituindo (7) em (1), temos:

$$u_T(T,x) = u_{xx}(T,x) + \sum_{k=1}^{nf} a_k r(T) g_h^k(x)$$

Como $u_k(t,x)$, $k = 1, \cdots, nf$ é solução de (1)-(4), com forçantes $f_k(t,x) = r(t)g_h^k(x)$, temos:

$$u_k(T,x) = u_{xx}^k(T,x) + f_k(T,x)$$

Portanto:

$$u_k(T,x) = u_{xx}^k(T,x) + r(T)g_h^k(x)$$

Devido à linearidade das equações vem que $u_{xx}(T,x) \equiv 0$

Então:

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

$$u_k(T,x) = r(T) g_h^k(x)$$

Donde concluímos o que buscávamos:

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

□

Como conheceremos apenas os valores de $u_T(x)$ medidos nos pontos x_i da malha, iremos determinar os valores de intensidade a_k de forma a minimizar

$$E_2 = \sqrt{\Delta x \sum_{i=1}^{N-1} (u_T(x_i) - \sum_{k=1}^{nf} a_k u_k(T,x_i))^2} \tag{11}$$

que corresponde a um problema de mínimos quadrados. Como as funções $u_k(T,x)$ são desconhecidas, iremos aproximar estes valores resolvendo as equações 1)-(4) através do método de Crank-Nicolson, desenvolvido no EP1 e apresentado anteriormente (tomaremos sempre $M = N$ para definir Δt.

Observação: A definição de E_2 acima não inclui os extremos do intervalo, pois as funções $u_T(x)$ e $u_k(x)$ aí se anulam, devido às condições de fronteira da equação. Este erro corresponde a uma versão discreta do erro quadrático $E = \sqrt{\int_0^1 [u_T(x) - \sum_{k=1}^{nf} a_k u_k(T,x)]^2 dx}$, com a integral aproximada pelo método dos trapézios.

No problema de mínimos quadrados (11) queremos aproximar o vetor de medições $u_T(x_i), i = 1, \cdots, N-1$ por uma combinação linear dos vetores $u_k(T,x_i), i = 1, \cdots, N-1$ obtidos pelas integrações usando o método de Crank-Nicolson (com $M = N$). Para a solução do problema de mínimos quadrados devemos resolver o seguinte sistema normal:

$$\begin{bmatrix} < u_1, u_1 > & < u_2, u_1 > & \cdots & < u_{nf}, u_1 > \\ < u_1, u_2 > & < u_2, u_2 > & \cdots & < u_{nf}, u_2 > \\ \vdots & \vdots & \vdots & \vdots \\ < u_1, u_{nf} > & < u_2, u_{nf} > & \cdots & < u_{nf}, u_{nf} > \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{nf} \end{bmatrix} = \begin{bmatrix} < u_T, u_1 > \\ < u_T, u_1 > \\ \vdots \\ < u_T, u_1 > \end{bmatrix} \tag{12}$$

com produto interno $< u, v >= \sum_{i=1}^{N-1} u(x_i) v(x_i)$.

3 Tarefas

3.1 Tarefa (a)

A primeira tarefa deste exercício programa pede para que, dados os pontos $p_1, \cdots, p_{nf}$ gere os vetores $u_k(T, x_i), i = 1, \cdots, N-1$ a partir da integração da equação do calor (1)-(4), com condição inicial $u_0(x) = 0$ e de fronteira $g_1(t) = g_2(t) = 0$, com forçante $f(t,x) = r(t)g_h^k(x), k = 1, \cdots, nf$. Para tanto devemos utilizar o método de Crank-Nicolson desenvolvido no EP1 (com $M = N$).

Logo abaixo encontra-se um trecho do código fonte deste exercício programa responsável por criar os vetores u_k pedidos:

```
def cria_uk(nf, m, t, posic, gr):
    lista = []
    for i in range(nf):
        vet = crie_vetor(n+1)
        u_ft = u_fonte(vet, m, t, posic, i)
        lista.append(u_ft)

    if gr:
        for i in range(nf):
            pos = []
            for j in range(n+1):
                pos.append((1 / n) * j)
            plt.plot(pos, lista[i], label="f={0}".format(i+1))
            plt.xlabel('position')
            plt.ylabel('temperature')
            plt.title('Vetores Uk - Temperatura x posição - Tempo = 1')
            plt.legend()
        plt.grid(True)
        plt.show()
    return lista
```

Note que a variável u_ft chama a função u_fonte, sendo esta última responsável por executar o método de Crank-Nicolson.

Obs.: A descrição completa desta função e das demais apresentadas neste relatório (docstrings) se encontram no código fonte.

3.2 Tarefa (b)

A segunda tarefa pede para que sejam montadas as matrizes do sistema normal do problema de mínimos quadrados para o cálculo das intensidades.

Para a execução desta tarefa foi implementada a seguinte função:

```
def Matrix(nf, m, t, posic, teste, gr):
    Ma = []
    uk = cria_uk(nf, m, t, posic, gr).copy()
    ut = U_tx(teste, nf, m, t, posic, gr).copy()

    for i in range(nf):
        linha = []
        for j in range(nf):
            u = uk[i].copy()
            v = uk[j].copy()
            a = produto_interno(u,v).copy()
            linha.append(a)
        Ma.append(linha)    # Ma é a matriz do lado esquerdo do sistema

    variavel = []
    for i in range(nf):
        variavel.append(1)   # Matriz das intensidades (a ser preenchida)

    Ba = []
    for k in range(nf):
        c = produto_interno(ut,uk[k])
        Ba.append(c)   # Ba é a matriz do lado direito do sistema
    return [Ma, variavel, Ba]
```

Note que esta função depende de outras funções auxiliares como a *cria_uk* e *U_tx*. Todas essas funções se encontram no código fonte.

Note ainda que a matriz coluna *variavel* é inicialmente preenchida com 1's.

3.3 Tarefa (c)

Por fim, a última tarefa desse EP pede para seja escrita uma rotina para calcular a decomposição LDL^t de uma matriz simétrica e outra para, dada esta decomposição, resolver um sistema linear associado à matriz decomposta. Devemos usar essas rotinas para resolver o problema de mínimos quadrados.

Para a implementação da rotina que faz a decomposição LDL^t é preciso se atentar ao fato de que, diferentemente do EP1, neste caso, a matriz a ser decomposta A não é esparsa. Isso significa que não podemos fazer simplificações no algoritmo, tal qual foi feito no EP1. Nesse sentido, foi preciso recorrer ao livro base da disciplina para implementar o algoritmo de forma mais geral:

```
def decomp_sim(vet):
    li = [0]
    comp = len(vet)
    L = li*comp
    D = li*comp
    v = li*comp

    for i in range(comp):
        L[i] = li*comp
        D[i] = li*comp

    for i in range(comp):
        L[i][i] = 1

    for i in range(comp):           # Step1
        for j in range(i):          # Step2
            v[j] = L[i][j]*D[j][j]

        somatorio = 0               # Step3
        for j in range(i):
            somatorio += L[i][j]*v[j]
        D[i][i] = vet[i][i] - somatorio

        for j in range(i+1,comp):   # Step4
            somat = 0
            for k in range(i):
                somat += L[j][k]*v[k]
            L[j][i] = (vet[j][i] - somat)/D[i][i]

    return [L, D]
```

Note que, apesar do argumento da função ser denominado *vet*, na verdade, trata-se de uma matriz quadrada.

Dada esta decomposição, implementamos uma rotina que resolve o sistema normal através da seguinte função:

```
def solve_sys2(sol, vet_l, vet_d, vet_dir, nf):
    y = crie_vetor(nf)  # Resolução do sistema Ly = b
    y[0] = vet_dir[0]

    for i in range(1, nf):
        soma = 0
        for j in range(0,i):
            soma += vet_l[i][j] * y[j]
        y[i] = vet_dir[i] - soma

    z = crie_vetor(nf)  # Resolução do sistema Dz = y
    for o in range(nf):
        z[o] = y[o] / vet_d[o][o]

    # Vamos transpor a matriz L
    vet_t = []
    for i in range(nf):
        linha = []
        for j in range(nf):
            linha.append(0)
        vet_t.append(linha)

    for i in range(nf):
        for j in range(nf):
            vet_t[i][j] = vet_l[j][i]

    sol[nf - 1] = z[nf - 1]
    for q in range(nf - 2, -1, -1):  # Resolução do sistema L^t x = z
        soma = 0
        for r in range(q+1, nf, 1):
            soma += vet_t[q][r] * sol[r]
        sol[q] = z[q] - soma

    return sol
```

Novamente foi preciso recorrer ao livro texto da disciplina para a implementação desta rotina.

4 Testes

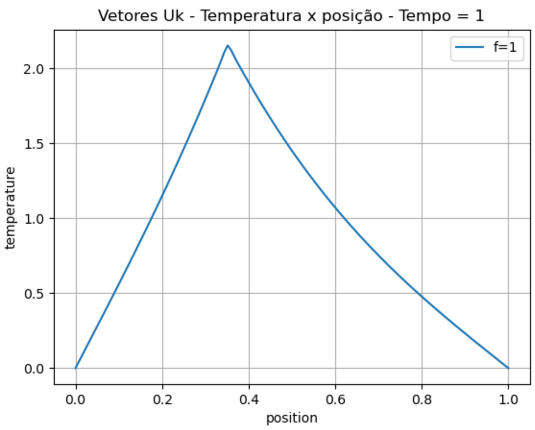
Tendo montado o código capaz de resolver o problemas de mínimos quadrados, podemos partir para os testes propostos pelo exercício programa. Como o próprio nome sugere, a ideia dos testes é verificar se o código elaborado está se comportando como esperado. Para todos os casos, utilizaremos $T = 1$ e $r(t) = 10(1 + cos(5t))$.

4.1 Teste (a)

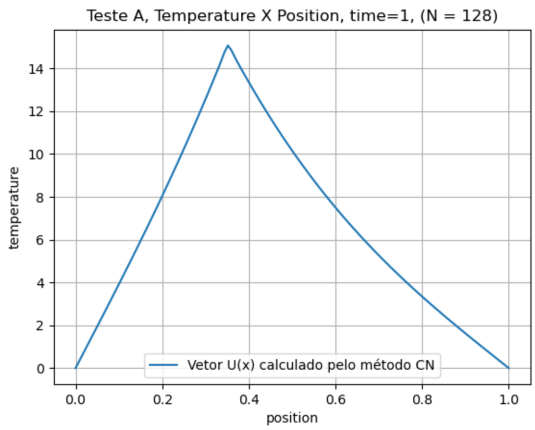
Para elaboração do primeiro teste proposto, precisaremos dos dados apresentados na tabela 1:

Parâmetros de entrada	
N	128
nf	1
p_1	0.35
$u_T(x_i) = 7u_1(T, x_i)$	

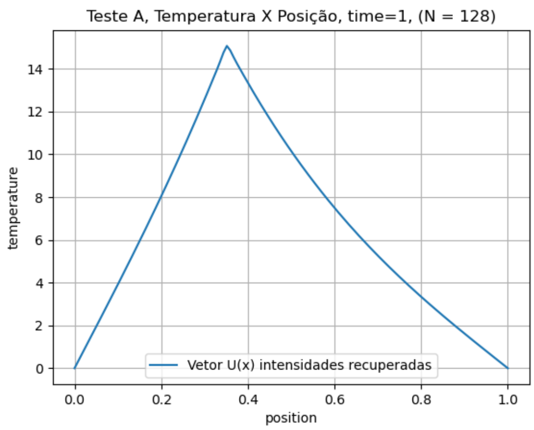
Tabela 1: Parâmetros de entrada para o teste (a)



(a) Vetores u_k



(b) Calculado



(c) Recuperado

Figura 1: *teste a:* $N = 128$, $nf = 1$

Como resultado o programa retorna:

Parâmetros de saída	
a_1	7
Erro quadrático	3.04e-15

Tabela 2: Parâmetros de saída para o teste (a)

Vale notar que o erro quadrático **não é exatamente zero** devido à erros de arredondamento ocasionados pelos cálculos do computador.

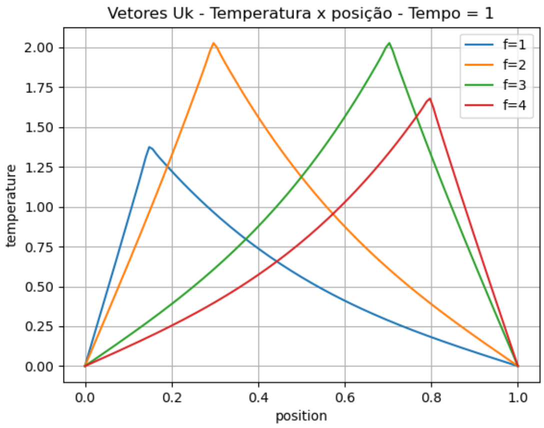
Obs.: Para mais informações sobre como executar o programa veja o arquivo *LEIAME*.

4.2 Teste (b)

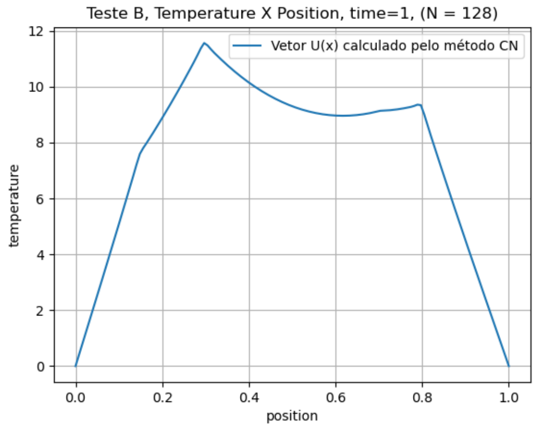
Para o segundo teste proposto temos os seguintes parâmetros de entrada:

Parâmetros de entrada	
N	128
nf	4
p_1	0.15
p_2	0.30
p_3	0.70
p_4	0.80
$u_T(x_i) = 2.3u_1 + 3.7u_2 + 0.3u_3 + 4.2u_4$	

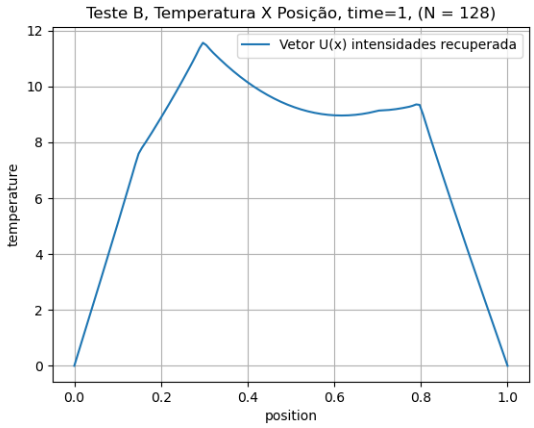
Tabela 3: Parâmetros de entrada para o teste (b)



(a) Vetores u_k



(b) Calculado



(c) Recuperado

Figura 2: *teste b:* $N = 128$, $nf = 4$

Como resultado o programa retorna:

Parâmetros de saída	
a_1	2.3
a_2	3.7
a_3	0.3
a_4	4.2
Erro quadrático	8.47e-15

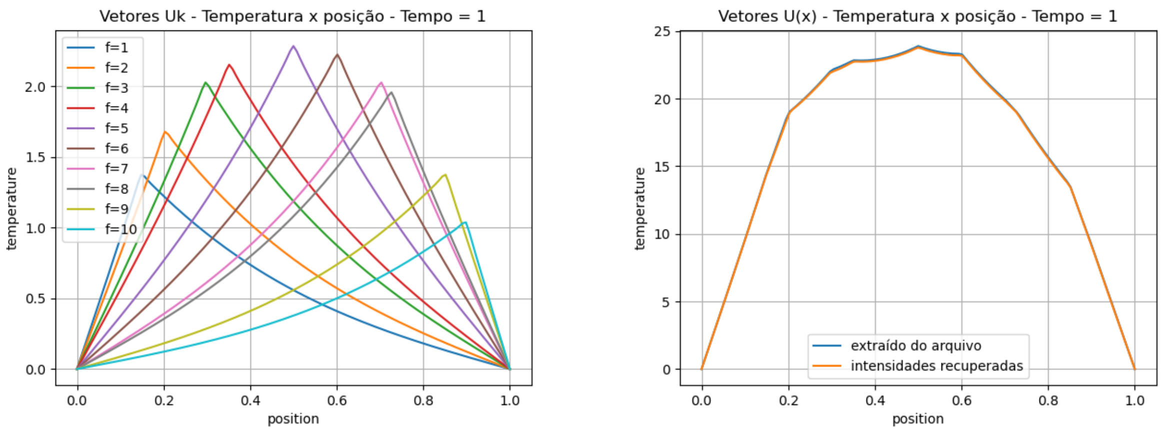
Tabela 4: Parâmetros de saída para o teste (b)

Os resultados na tabela 4 já eram esperados, corroborando a eficácia do algoritmo. Novamente, note que o erro quadrático não é exatamente zero devido aos erros de arredondamento.

4.3 Teste (c)

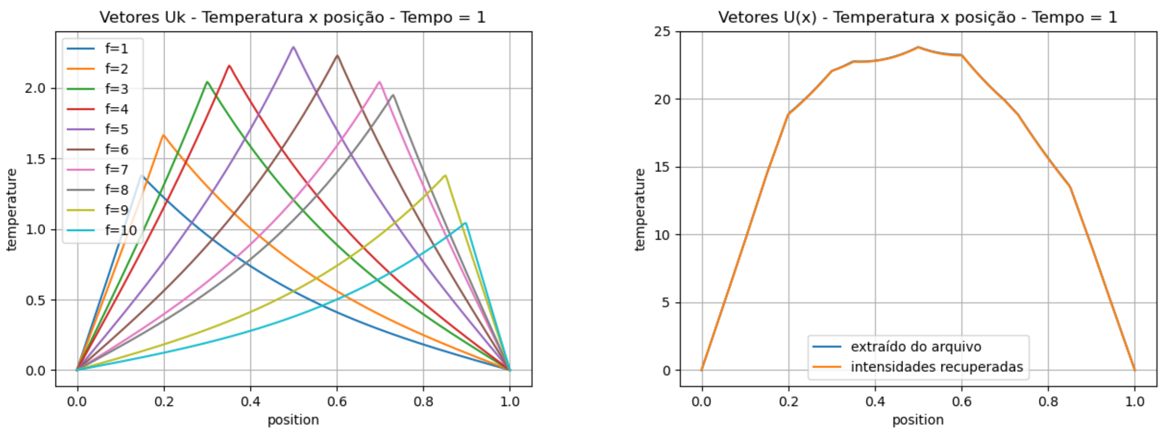
Para elaboração dos testes c e d foi utilizado o arquivo *teste* disponibilizado no site da disciplina. Para a exibição destes testes julgamos melhor apresentarmos os gráficos num mesmo plano cartesiano, poupando, assim, a exibição de muitas imagens. Tanto no teste c quanto no d iremos utilizar 5 valores de N: 128, 256, 512, 1024 e 2048. O número de fontes nf será sempre 10, nos dois testes, sendo os p_{nf} diferentes para cada valor de N.

Iremos apenas aprensentar os gráficos nos testes c e d, bem como seus erros quadráticos e intensidades, discutindo ao final os resultados obtidos.



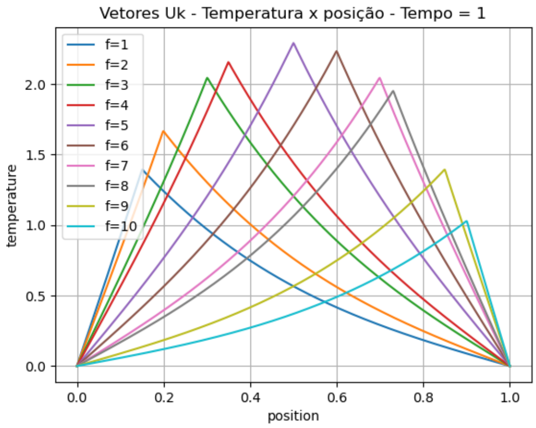
(a) Vetores u_k (b) Calculado e Recuperado

Figura 3: *teste c:* $N = 128$, $nf = 10$

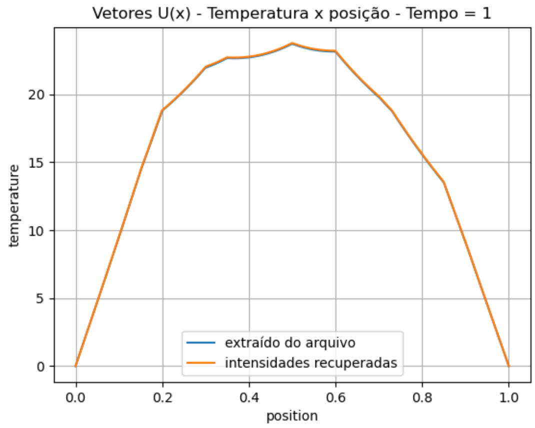


(a) Vetores u_k (b) Calculado e Recuperado

Figura 4: *teste c:* $N = 256$, $nf = 10$

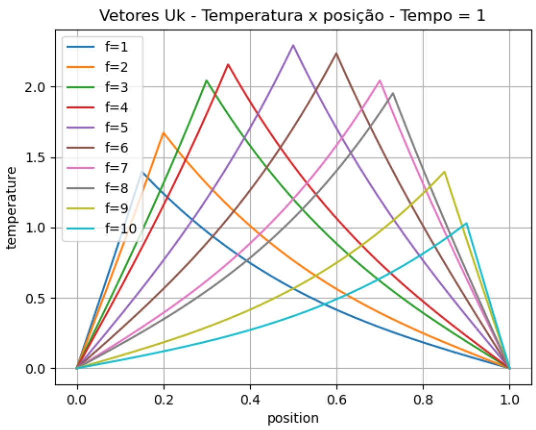


(a) Vetores u_k

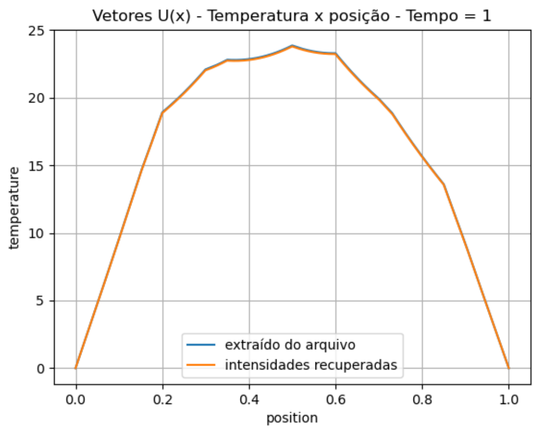


(b) Calculado e Recuperado

Figura 5: *teste c:* $N = 512$, $nf = 10$

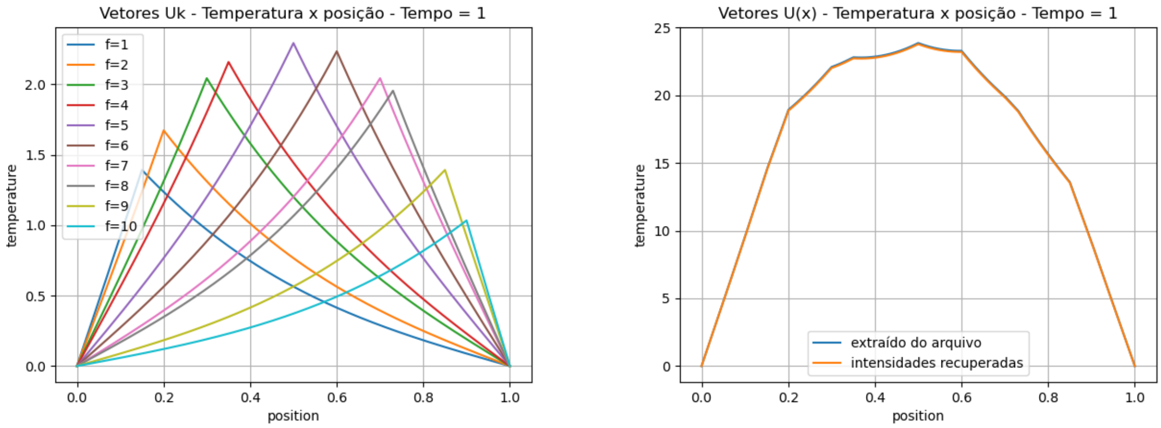


(a) Vetores u_k



(b) Calculado e Recuperado

Figura 6: *teste c:* $N = 1024$, $nf = 10$



(a) Vetores u_k

(b) Calculado e Recuperado

Figura 7: *teste c:* $N = 2048$, $nf = 10$

N	Erro quadrático	Vetor de intensidades
128	2.44e-02	[1.21; 4.84; 1.89; 1.58; 2.21; 3.12; 0.38; 1.49; 3.98; 0.40]
256	1.24e-02	[0.90; 5.08; 2.10; 1.41; 2.23; 3.10; 0.51; 1.39; 3.95; 0.41]
512	8.48e-03	[0.93; 5.05; 2.04; 1.47; 2.20; 3.09; 0.64; 1.27; 3.88; 0.53]
1024	3.78e-03	[1.01; 4.99; 1.99; 1.51; 2.19; 3.10; 0.65; 1.25; 3.88; 0.53]
2048	2.68e-12	[1.00; 5.00; 2.00; 1.50; 2.20; 3.10; 0.60; 1.30; 3.90; 0.50]

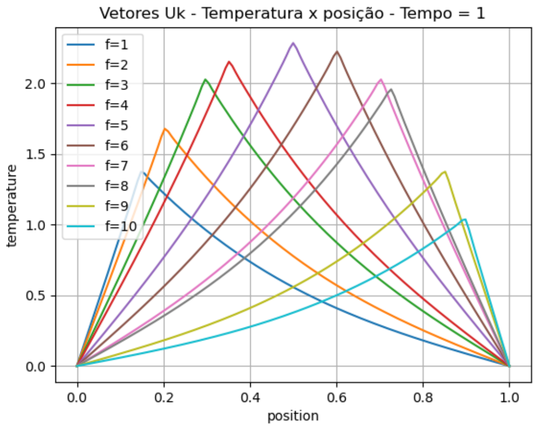
Tabela 5: Erros quadráticos e intensidades para o teste (c)

Nos gráficos (b) das figuras acima é possível notar que as intensidades recuperadas estão bem próximas das retiradas do arquivo teste.

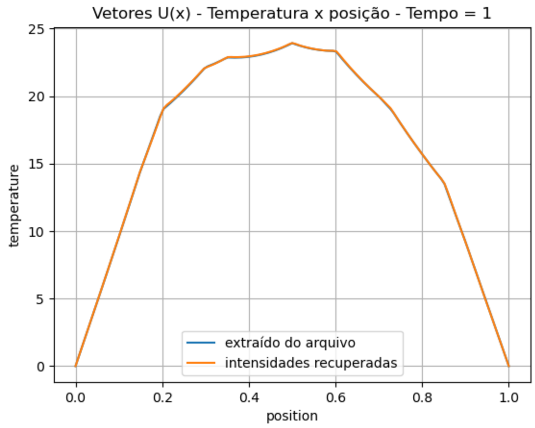
É interessante notar que, conforme N aumenta o erro quadrático diminui, o que era esperado, visto que a malha está sendo cada vez mais refinada. Este fenômeno acontecia no EP1 também, porém com uma redução do erro com fator de $1/4$ para o método de Crank-Nicolson. No caso do erro quadrático, podemos observar um fator de redução de $1/2$ até $N = 1024$. Quando N dobra e se torna 2048 o erro se reduz em ordens de grandeza.

4.4 Teste (d)

No quarto e último teste introduziremos ruídos, que representam erros de medição na temperatura final. Este teste é análogo ao teste c, com os mesmos parâmetros de entrada.

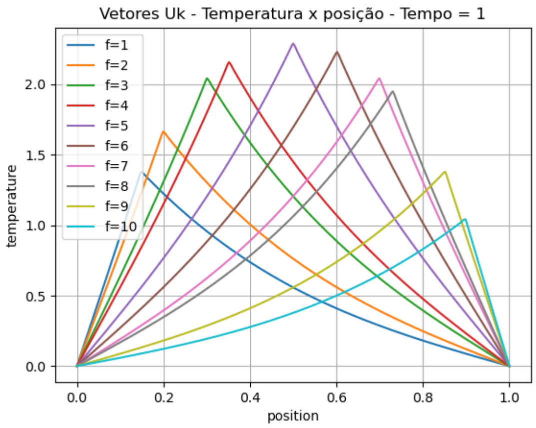


(a) Vetores u_k

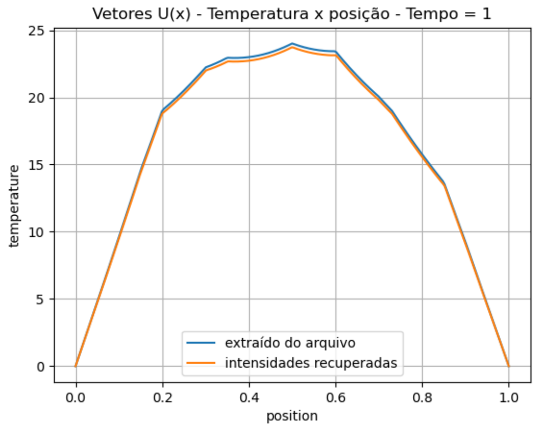


(b) Calculado e Recuperado

Figura 8: *teste d:* $N = 128$, $nf = 10$

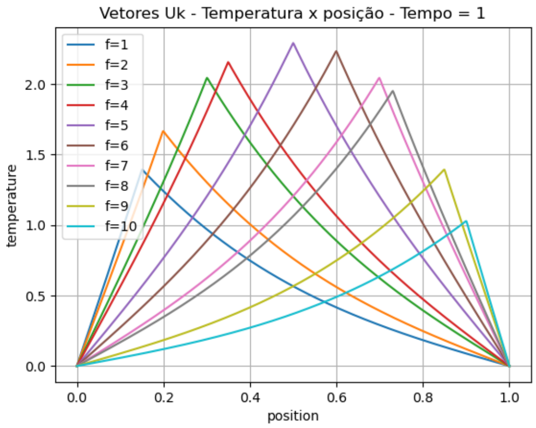


(a) Vetores u_k

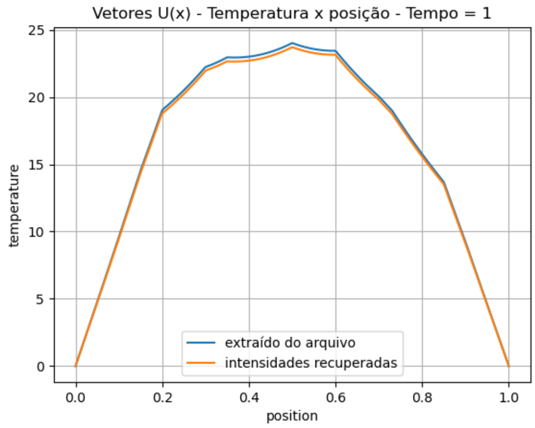


(b) Calculado e Recuperado

Figura 9: *teste d:* $N = 256$, $nf = 10$

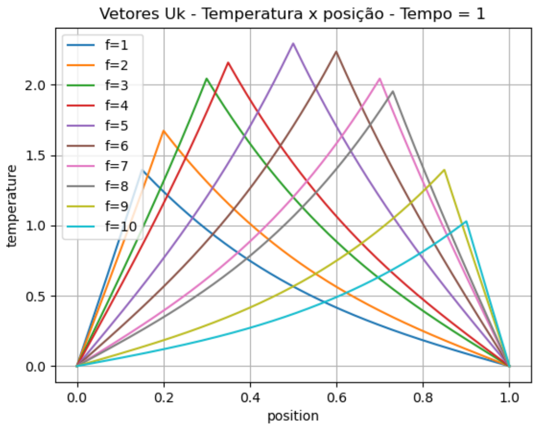


(a) Vetores u_k

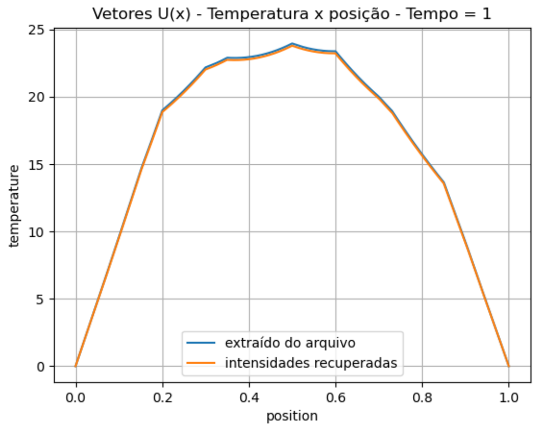


(b) Calculado e Recuperado

Figura 10: *teste d:* $N = 512$, $nf = 10$

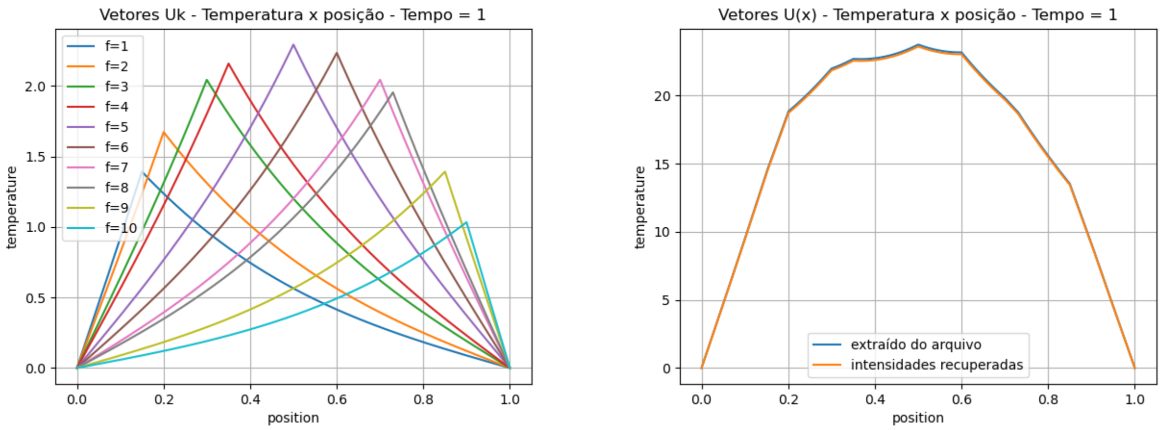


(a) Vetores u_k



(b) Calculado e Recuperado

Figura 11: *teste d:* $N = 1024$, $nf = 10$



(a) Vetores u_k (b) Calculado e Recuperado

Figura 12: *teste d:* $N = 2048$, $nf = 10$

N	Erro quadrático	Vetor de intensidades
128	4.71e-02	[1.22; 4.87; 1.90; 1.59; 2.23; 3.14; 0.38; 1.50; 4.00; 0.41]
256	3.94e-02	[0.91; 5.12; 2.12; 1.43; 2.25; 3.13; 0.51; 1.40; 3.98; 0.42]
512	8.81e-03	[0.94; 5.09; 2.06; 1.48; 2.21; 3.11; 0.64; 1.28; 3.90; 0.53]
1024	2.46e-02	[1.01; 5.03; 2.00; 1.52; 2.21; 3.12; 0.66; 1.26; 3.91; 0.53]
2048	3.15e-02	[1.00; 5.02; 2.01; 1.51; 2.21; 3.11; 0.60; 1.30; 3.91; 0.50]

Tabela 6: Erros quadráticos e intensidades para o teste (d)

Neste teste, para a representação dos ruídos, foi utilizada a função *random* do Python, que produz a cada chamada um valor (pseudo) aleatório entre 0 e 1. Para nossa finalidade, essa função foi adaptada para gerar números pseudoaleatórios entre -1 e 1.

Nesse sentido, a cada vez que se executar o programa com o teste *d* serão gerados **novos erros quadráticos e novas intensidades**. É possível notar este fato ao analisar o erro quadrático conforme N aumenta na tabela 6, onde este erro se comporta de forma "aleatória", não seguindo o mesmo padrão do teste *c*. De qualquer maneira, analogamente ao teste *c*, é possível notar, pelos gráficos (b), que as intensidades estão bem próximas do esperado.

5 Conclusões e comentários finais

As duas partes do Exercício Progama proposto pela disciplina *MAP3121 - Métodos Numéricos e Aplicações* constituíram ótimas oportunidades de colocar em prática conceitos e métodos que vimos durante as aulas teóricas.

No EP1, lançamos mão de três métodos numéricos para resolver um problema direto ligado à equação do calor: o método de Euler explícito, o método de Euler implícito e o método de Crank-Nicolson. A partir deles, observamos as diferenças entre métodos explícitos e implícitos, principalmente no que diz respeito a seus erros e tempos de execução. Pudemos também analisar as condições que regem a convergência destes métodos, vendo, na prática, que uma leve alteração nos parâmetros do problema pode resultar em erros absurdos. Além disso, observamos nos diversos gráficos plotados a evolução das aproximações com o passar do tempo e com o refinamento da malha.

No EP2, por sua vez, estudamos um problema inverso, também ligado à equação do calor. A partir dele, estudamos e criamos rotinas capazes de calcular as intensidades das fontes de calor aplicadas em posições conhecidas de uma barra, conhecendo a distribuição final das temperaturas a priori. A principal ferramenta utilizada para resolver tal problema foi o método dos mínimos quadrados, o qual estudamos a fundo durante o semestre. Com ele, pudemos montar e resolver um sistema normal e, assim, encontrar os valores das intensidades de fonte que minimizam o erro quadrático.

Finalmente, podemos afirmar que o presente Exercício Programa foi muito interessante para os alunos praticarem diversos conceitos e métodos aprendidos durante o semestre e que certamente permearão a nossa futura carreira profissional.

6 Bibliografia

- Burden & Faires - Análise Numérica - tradução da 10ª edição;
- Noções de Cálculo Numérico;