Giovanna Girotto - 11803416
Vittor Braide Costa - 11806920

Relatório EP3 MAP3121 - Modelagem de um Sistema de Resfriamento de Chips

São Paulo, SP

Julho, 2022

Giovanna Girotto - 11803416
Vittor Braide Costa - 11806920

Relatório EP3 MAP3121 - Modelagem de um Sistema de Resfriamento de Chips

Relatório da Tarefa 3 - *Modelagem de um Sistema de Resfriamento de Chips* proposto pela disciplina MAP3121 - Métodos Numéricos e Aplicações

Escola Politécnica da Universidade de São Paulo

São Paulo, SP
Julho, 2022

Lista de ilustrações

Sumário

1 Introdução

A prática de engenharia envolve o desenvolvimento e fabricação de diversos produtos que contribuem na solução de problemas concretos da sociedade. Assim sendo, para viabilizar uma boa produção e tornar tudo aquilo produzido eficiente e suficientemente duradouro, é necessário aplicar diversos conceitos científicos, como aspectos físicos, que influenciam no resultado final.

Nesse sentido, o presente trabalho toma como exemplo a necessidade de uma análise térmica de um chip de computador dado variadas condições de ambiente. Tal análise é fundamental, já que temperaturas muito elevadas podem danificar alguns componentes e torna o produto inoperável, fato indesejável para as boas práticas de engenharia.

Dado tal problema, realiza-se algumas modelagens matemáticas que simulam o processo de interesse. Para a análise térmica, deseja-se representar a distribuição de calor de um corpo e avaliar a evolução da temperatura. Para tanto, utiliza-se a Lei de Fourier, a qual é representada por uma equação diferencial.

Apesar de descrever fisicamente o sistema, a resolução de uma equação diferencial envolve alguns processos matemáticos que, em determinados casos, deixam a busca por uma solução analítica muito complexa. Dessa forma, métodos numéricos são introduzidos, já que, com a ajuda de um computador, a solução fica mais simplificada, mesmo com o surgimento de alguns erros.

Com isso, a proposta do trabalho consiste em modelar um sistema de resfriamento de chips aplicando métodos numéricos para lei de Fourier. Para esse propósito, utiliza-se da construção de um código em *Python* que possibilita a execução do método de Elementos Finitos para encontrar a distribuição de temperaturas no chip.

Posteriormente, para realizar uma validação, executa-se um teste com equações diferenciais de solução exata conhecida e avalia-se o erro obtido. Com o código validado, simula-se diversas condições de ambiente para encontrar a distribuição de temperatura do chip. Ao fim, é possível tomar algumas conclusões que possibilitam a fabricação do chip mantendo bos práticas de engenharia.

2 Equação do Calor

Nesta tarefa, estuda-se o comportamento da difusão térmica que ocorre em um chip de computador ao utilizar um resfriador colado em sua parte superior. Chama-se de L e h o comprimento e altura do chip, respectivamente, e consida-se o problema unidimensional em que a variação de temperatura ocorre somente na horizontal. Assim, a modelagem da distribuição de calor pode ser representada pela equação do calor proposta pela lei de Fourier:

$$\rho C \frac{\partial T(t,x)}{\partial t} = \frac{\partial}{\partial x}\left(k(x)\frac{\partial T(t,x)}{\partial x}\right) + Q(t,x) \tag{2.1}$$

Onde:

- $T(t,x)$ é a temperatura em uma posição x e instante t
- ρ é a densidade do material do chip
- C é o calor específico do material
- k é o parâmetro de condutividade térmica do material
- Q é uma fonte de calor

Além disso, convenciona-se que o calor gerado pelo chip é Q_+, enquanto o calor retirado pela resfriador é Q_-, o que resulta em $Q = Q_+ - Q_-$.

2.1 Considerações e Simplificações adotadas

Para a modelagem do problema, adotou-se uma série de simplificações:

Em primeiro lugar, assume-se que a troca de calor no topo do chip com o resfriador é perfeita e que não há troca com o ambiente na parte inferior. Ainda, considera-se que a temperatura nos extremos será exatamente a temperatura do ambiente externo.

Por fim, considera-se que o processador trabalha em regime constante, i. e., gera sempre a mesma quantidade de calor e o resfriador extrai a mesma quantidade de calor, de modo que a temperatura do chip tende a um estado estacionário.

Nesse caso:

$$\frac{\partial T(t,x)}{\partial t} = 0 \Rightarrow \boxed{-\frac{\partial}{\partial x}\left(k(x)\frac{\partial T(x)}{\partial x}\right) = Q(x)} \tag{2.2}$$

3 Método de Elementos Finitos

Neste capítulo será descrito o método de elementos finitos para resolução de uma equação diferencial da forma:

$$-\frac{\partial}{\partial x}\left(k(x)\frac{\partial T(x)}{\partial x}\right) + q(x)T(t,x) = Q(x), \text{para } 0 \leq x \leq 1 \tag{3.1}$$

As condições de contorno são $T(0) = T(1) = 0$

Para resolver a equação em $T(x,t)$, pode-se utilizar o método de **Rayleigh-Ritz**, que consiste em uma técnica variacional que busca minimizar uma certa integral.

Uma função $T(x,t)$ é a única solução para a equação 3.1 se e somente se $T(x,t)$ é a única função que minimiza a integral:

$$I[u] = \int_0^1 k(x)[u'(x)]^2 + q(x)[u(x)]^2 - 2Q(x)u(x)dx \tag{3.2}$$

Define-se um subespaço de dimensão finita U_n de U_0, através da projeção ortogonal de $u(x)$ em U_n. Para a escolha de U_n e sua base, usa-se o espaço de Splines Lineares $S^0_{2,n}[0,1]$ com nós uniformemente espaçados em $[0,1]$. Tomando $h = 1/(n+1)$ e $x_i = ih, i = 0, 1, ..., n+1$ tem-se que cada Spline é uma função contínua em $[0,1]$ que se anula nos extremos e coincide com uma reta entre cada dois nós.

Uma base para esse espaço de Splines é dada por funções:

$$\Phi_i(x) = \begin{cases} 0, \text{fora de } [x_{i-1}, x_{i+1}] \\ \frac{x - x_{i-1}}{h} \text{em } [x_{i-1}, x_i] \\ \frac{x_{i+1} - x_i}{h} \text{em } [x_i, x_{i+1}] \end{cases} \tag{3.3}$$

E a derivada dessa função é dada por:

$$\Phi'_i(x) = \begin{cases} 0, \text{fora de } [x_{i-1}, x_{i+1}] \\ \frac{1}{h} \text{em } [x_{i-1}, x_i] \\ \frac{-1}{h} \text{em } [x_i, x_{i+1}] \end{cases} \tag{3.4}$$

Visto isso, prova-se que uma aproximação da solução de $T(t,x)$ que minimiza a integral apresentada é dada por $T(x) = \sum_{i=1}^{n} c_i \Phi_i$. Assim, as constantes $c_1, c_2, ..., c_3$ devem ser calculadas.

Para isso, é necessário utilizar o sistema linear $\alpha c = \beta$, em que α é uma matriz tridiagonal com valores:

$$\alpha_{i,i} = D_{4,i} + D_{4,i+1} + D_{2,1} + D_{3,1} \text{ para cada } i = 1, 2, ..., n; \tag{3.5}$$

$$\alpha_{i,i+1} = -D_{4,i+1} + D_{1,i} \text{ para cada } i = 1, 2, ..., n-1; \tag{3.6}$$

$$\alpha_{i,i-1} = -D_{4,i} + D_{1,i-1} \text{ para cada } i = 2, 3, ..., n; \tag{3.7}$$

e

$$\beta_i = D_{5,i} + D_{6,i} \text{ para cada } i = 1, 2, ..., n; \tag{3.8}$$

Os coeficientes D, por sua vez, são calculados da seguinte forma:

$$\begin{cases} D_{1,i} = (\frac{1}{h_i})^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)(x - x_i) q(x) dx, \quad \text{para cada } i = 1, 2, ..., n-1; \\ D_{2,i} = (\frac{1}{h_{i-1}})^2 \int_{x_{i-1}}^{x_i} (x - x_{i-1})^2 q(x) dx, \quad \text{para cada } i = 1, 2, ..., n; \\ D_{3,i} = (\frac{1}{h_i})^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)^2 q(x) dx, \quad \text{para cada } i = 1, 2, ..., n; \\ D_{4,i} = (\frac{1}{h_{i-1}})^2 \int_{x_{i-1}}^{x_i} k(x) dx, \quad \text{para cada } i = 1, 2, ..., n+1; \\ D_{5,i} = \frac{1}{h_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1}) Q(x) dx, \quad \text{para cada } i = 1, 2, ..., n; \\ D_{6,i} = \frac{1}{h_i} \int_{x_i}^{x_{i+1}} (x_{i+1} - x) Q(x) dx, \quad \text{para cada } i = 1, 2, ..., n; \end{cases} \tag{3.9}$$

Com os coeficientes α e β é possível montar o sistema de matrizes e calcular todos os coeficientes c, o que resulta no cálculo de $T(x) = u(x) = \sum_{i=1}^{n} c_i \Phi_i$

3.1 Condições de contorno não-homogêneas

Para o caso em que $T(0) = a$ e $T(b) = b$, pode-se reduzir o caso para:

$$\bar{Q(x)} = Q(x) + (b-a)k'(x) - q(x)(a + (b-a)x), v(0) = v(1) = 0 \tag{3.10}$$

Mostra-se que $u(x) = v(x) + a + (b-a)x$ é solução da equação com condições de fronteira $u(0) = a$ e $u(1) = b$.

Para isso, verifica-se que:

$$v(0) = 0 \Rightarrow u(0) = a \tag{3.11}$$

$$v(1) = 0 \Rightarrow u(1) = b \tag{3.12}$$

Assim, substituindo $u(x)$ em $L(u(x)) = (-k(x)u'(x))' + q(x)u(x) = Q(x)$:

$$L(u(x)) = (-k(v' + (b-a)))' + q(v(x) + a + (b-a)x) \tag{3.13}$$

$$L(u(x)) = -(kv'(x)' + q(v(x)) - k'(x)(b-a) + q[a + (b-a)x] \tag{3.14}$$

Mas, se $L(v(x)) = -(kv'(x) + q(v(x)))$ e $L(u(x)) = Q(x)$:

$$L(v(x)) = Q(x) + k'(x)(b-a) - q(x)[a + (b-a)x] \tag{3.15}$$

Essa é, portanto, a equação a ser resolvida, mostrando que $u(x)$ é solução da equação 3.1 com condições de fronteira $u(0) = a$ e $u(1) = b$.

4 Tarefa

A Tarefa consiste em implementar o método de elementos finitos para $q(x) = 0$ e condições de contorno nas extremidades $u(0) = a, u(L) = b$. para isso, foi criada uma função 'fourier_fem' que recebe como *inputs* as condições de contorno 'a' e 'b', 'L', 'n', 'q','Q_{ger}','Q_{dis}', em que define-se $Q(x) = Q_{ger} - Q_{dis}$.

4.1 Elaboração do código

O primeiro passo do código é definir as variáveis a serem calculadas, criando vetores vazios:

```
def fourier_fem(a,b,L,n,q,Q_ger,Q_dis,k):
    h=L/(n+1)
    f=lambda x: Q_ger(x)-Q_dis(x)
    #x0=0, x_n+1=L
    X=np.arange(0,L+h,h)
    D_1,D_2,D_3,D_4,D_5,D_6=np.zeros(n-1),np.zeros(n),np.zeros(n),np.zeros(n+1),np.zeros(n),np.zeros(n)
    alfa1 , alfa2 , alfa0=np.zeros(n),np.zeros(n),np.zeros(n)
```

Feito isso, parte-se para o cálculo dos coeficientes D, que são necessários para os cálculos de α e β. Para isso, faz-se necessário o cálculo de integrais, as quais foram calculadas pelo método de Gauss com a utilização do código criado para o **EP2**, com a função 'integracao_gaussiana'.

```
    for i in range(1,n):
        D_1[i-1]=((1/h))**2*integracao_gaussiana(X[i],X[i+1],lambda x: (X[i+1]-x)*(x-X[i])*q(x))
        D_2[i-1]=((1/h))**2*integracao_gaussiana(X[i-1],X[i],lambda x: ((x-X[i-1])**2)*q(x))
        D_3[i-1]=((1/h)**2)*integracao_gaussiana(X[i],X[i+1],lambda x: ((X[i+1]-x)**2)*q(x))
        D_4[i-1]=((1/h)**2)*integracao_gaussiana(X[i-1],X[i],lambda x: k(x))
        D_5[i-1]=(1/h)*integracao_gaussiana(X[i-1],X[i],lambda x: (x-X[i-1])*f(x))
        D_6[i-1]=(1/h)*integracao_gaussiana(X[i],X[i+1],lambda x: (X[i+1]-x)*f(x))
```

Feito isso, parte-se para a determinação de α e β:

```
def fourier_fem(a,b,L,n,q,Q_ger,Q_dis,k):
    for i in range(1,n):
```

```
        alfa1[i-1]=D_4[i-1]+D_4[i]+D_2[i-1]+D_3[i-1]
        alfa2[i-1]=-D_4[i]+D_1[i-1]
        #loop para i=2,3,...,n
        alfa0[i]=-D_4[i]+D_1[i-1]
    alfa0[0]=0
    alfa1[n-1]=D_4[n-1]+D_4[n]+D_2[n-1]+D_3[n-1]
    alfa2[n-1]=0
    beta=D_5+D_6
```

Com esses valores, torna-se possível resolver o sistema $\alpha c = \beta$, e, lembrando que α é uma matriz tridiagonal, utiliza-se o código de resolução de sistemas criado para o **EP1**, com a função 'solucao_sistema':

```
c=solucao_sistema([alfa0,alfa1,alfa2],beta)
```

O último passo, é, portanto, calcular a soma $\sum_{i=1}^{n} c_i \Phi_i$:

```
    for i in range(1,n+1):
        #calculo phi_i
        for j in range(1,n+1):
            if X[i]>X[j-1] and X[i]<=X[j]:
                phi=(1/h)*(X[i]-X[j-1])
            elif X[i]>X[j] and X[i]<=X[j+1]:
                phi=(1/h)*(X[j+1]-X[i])
            else:
                phi=0
            sum_phi[i-1]+=c[j-1]*phi
        #Adicionar casos não homogêneos
        sum_phi[i-1]+=a+((b-a)/L)*X[i]
    sum_phi=np.insert(sum_phi,0,a)
    sum_phi=np.insert(sum_phi,n+1,b)
    return sum_phi
```

4.2 Validação

No primeiro passo do projeto, faz-se a validação do código criado com dois exemplos distintos.

O primeiro exercício recebe como *inputs*:

- $L = 1$
- $k(X) = 1$
- $q(x) = 0$
- $f(x) = Q(x) = 12x(1 - x) - 2$

E pede como resposta a solução numérica da equação diferencial do tipo 3.1 variando-se os passos do intervalo de integração para $n = 7, 15, 31, 63$.

Sabendo que a solução exata para o problema é $u(x) = x^2(1 - x)^2$, utilizou-se a função 'fourier_fem' para o cálculo da solução numérica, e definiu-se como erro da função a diferença entre elas:

```
def validacao(N,a,b,L,q,Q_ger,Q_dis,k,u,str_sol_exata):
    h_qua=np.zeros(len(N))
    error=np.zeros(len(N))
    X_exata=np.arange(0,L+0.01,0.01)
    sol_plot=np.zeros(len(X_exata))
    for l in range(len(X_exata)):
        sol_plot[l]=u(X_exata[l])
    for i in range(len(N)):
        h=L/(N[i]+1)
        sol_exata=np.zeros(N[i]+2)
        X=np.arange(0,L+h,h)
        h_qua[i]=(L/(N[i]+1))**2
        sol_numerica=fourier_fem(a,b,L,N[i],q,Q_ger,Q_dis,k)
        for j in range(0,N[i]+1):
            sol_exata[j]=u(X[j])
        error[i]=max(abs(sol_numerica-sol_exata))
        plt.plot(X_exata,sol_plot,'-',label="Solução exata ($u(x)$)")
        plt.plot(X,sol_numerica,'p',label="Solução numérica ($\overline{u}(x)$)")
        plt.legend()
        plt.title('Paridade da simulação e solucão exata '+str_sol_exata+' para n=%i '%N[i])
        plt.xlabel('x')
        plt.ylabel('Solução')
        plt.grid()
        plt.show()
    plt.plot(h_qua, error,'-p')
    plt.title('Erro da simulação para solução exata '+str_sol_exata,y=1.08)
    plt.xlabel('$h^2$')
    plt.ylabel('Erro')
    plt.grid()
    plt.show()
```

Por fim, plotou-se os gráficos da solução numérica e exata para $n = 7, 15, 31, 63$:

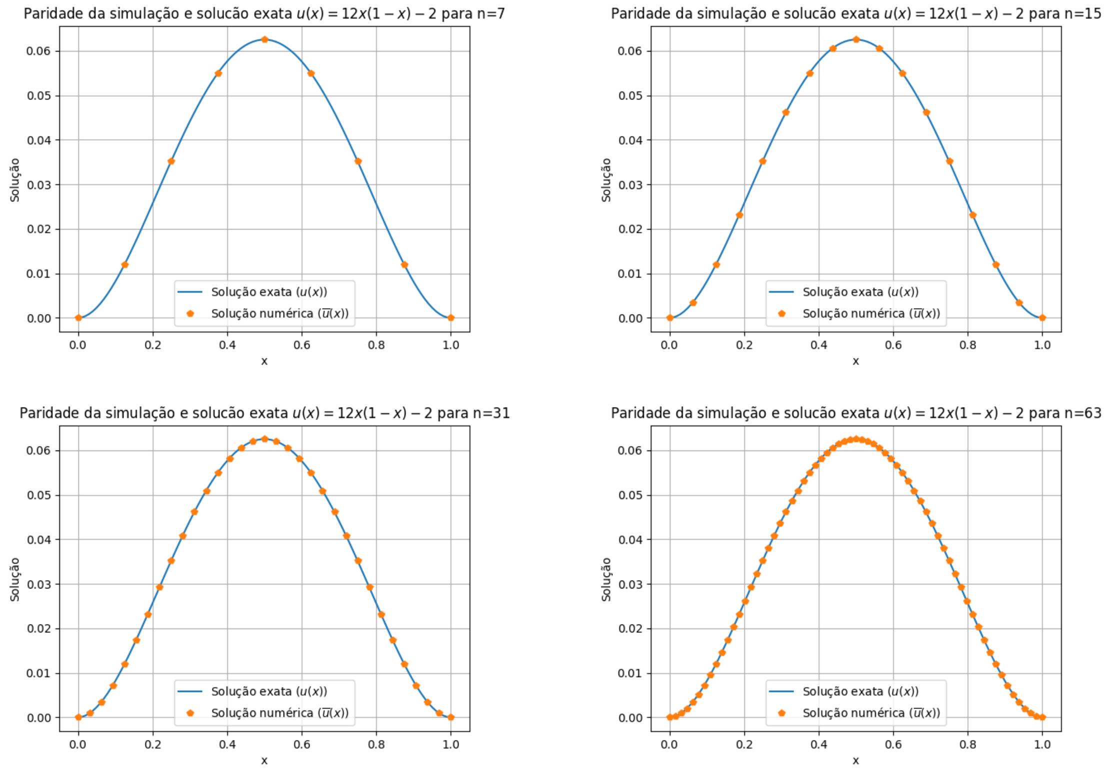

Figura 1 – Validação para $f(x) = Q(x) = 12x(1-x) - 2$

Depois, calculou-se o erro máximo entre a solução numérica e exata e plotou-se um gráfico das diferenças em relação ao passo elevado ao quadrado (h^2):

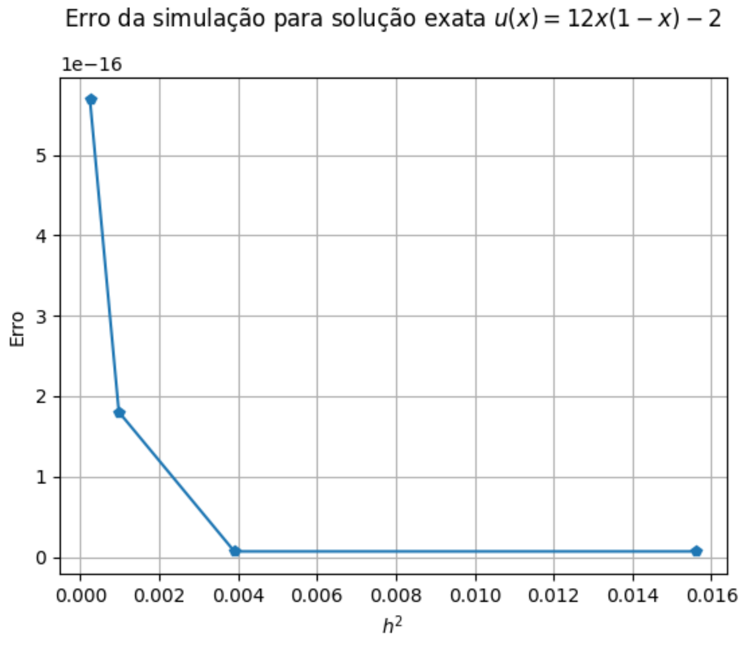

Figura 2 – Erro em função do passo

O segundo exercício recebe como *inputs*:

- $L = 1$
- $k(X) = e^x$
- $q(x) = 0$
- $f(x) = Q(x) = e^x + 1$

Sabendo que a solução exata é $u(x) = (x-1)(e^{-x}-1)$ faz-se o mesmo processo que o anterior para $n = 7, 15, 31, 63$, de modo a se obter os gráficos:

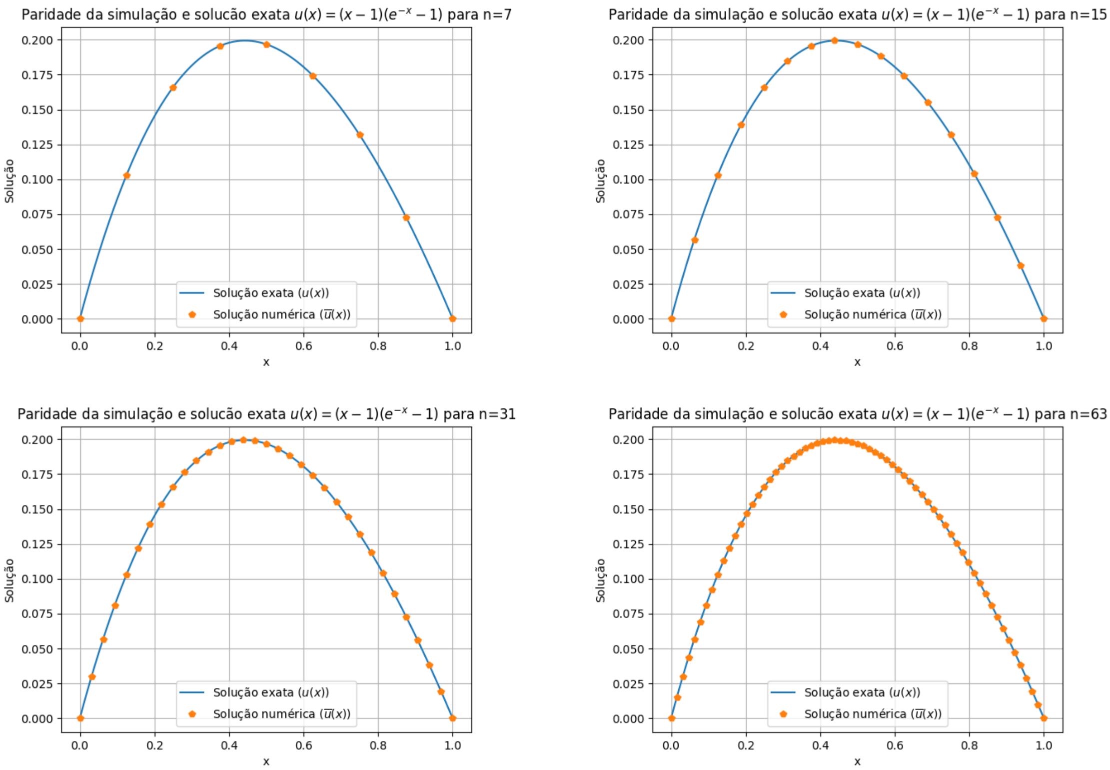

Figura 3 – Validação para $f(x) = Q(x) = (x-1)(e^{-x}-1)$

Da mesma forma, o erro em função do passo ao quadrado (h^2):

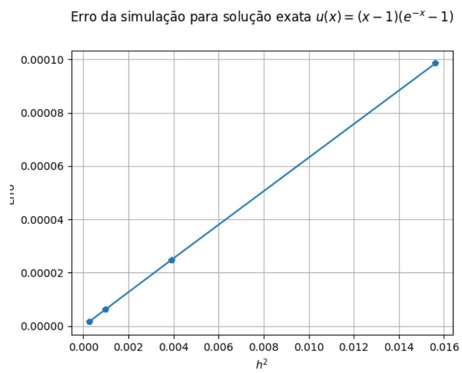

Figura 4 – Erro em função do passo

Assim, nota-se que no segundo exercício a convergência do método é de segunda ordem, uma vez que o erro varia linearmente com o quadrado do passo. No primeiro exercício, no entanto, isso não ocorre, o que pode estar atrelado ao fato do erro ser muito pequeno (10^{-16}), na mesma ordem da precisão de cálculo de um computador.

4.3 Equilíbrio com forçantes de calor

Neste exercício, pede-se para calcular a distribuição de temperatura no chip considerando que ele é feito apenas de silício, ou seja, $k(x) = k = 3,6W/(m \cdot K)$. Para isso, alguns diferentes casos serão plotados variando-se a função do calor $Q(x)$.

4.3.1 Caso 1: Sem resfriamento e Q=cte

O primeiro caso de estudo considera que não há resfriamento no chip ($Q_- = 0$) e que o calor gerado no chip é constante e é dado por $Q = P/V$, sendo P a potência e V o volume.

Assim, para $P = 30W$ e $V = 8 \cdot 10^{-7} m^3$, e condições de contorno nas extremidades $T(0) = T(L) = 25^oC$, obteve-se a distribuição de temperatura no chip:

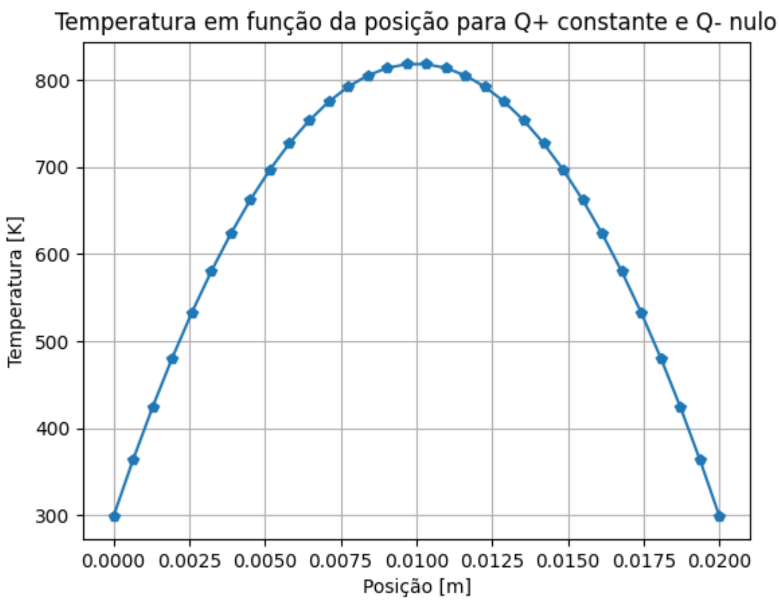

Figura 5 – T(x) para Q(x) constante e sem resfriamento

Nesse caso, retoma-se a equação:

$$-\frac{\partial}{\partial x}\left(k(x)\frac{\partial T(x)}{\partial x}\right) = Q(x) \tag{4.1}$$

Sendo $k(x)$ e $Q(x)$ constantes:

$$-k\frac{\partial^2 T(x)}{\partial x^2} = Q \Rightarrow \frac{\partial^2 T(x)}{\partial x^2} = -Q/k \tag{4.2}$$

Esse resultado indica que $T(x)$ deve ser uma função quadrática, uma vez que sua segunda derivada é uma constante. Isso de fato é verificado no gráfico obtido, que apresenta um perfil parabólico de distribuição de temperaturas no interior do chip.

A temperatura é, portanto, mais alta no centro do chip, chegando ao patamar de $T_{máx} \approx 540^o$, enquando as extremidades permanecem à temperatura ambiente.

4.3.2 Caso 2: Geração e dissipação de calor constantes

Agora, introduz-se o resfriamento do chip. Para isso, acrescenta-se um valor dissipado Q_- igual à metade do módulo de Q_+. O gráfico plotado fica, portanto:

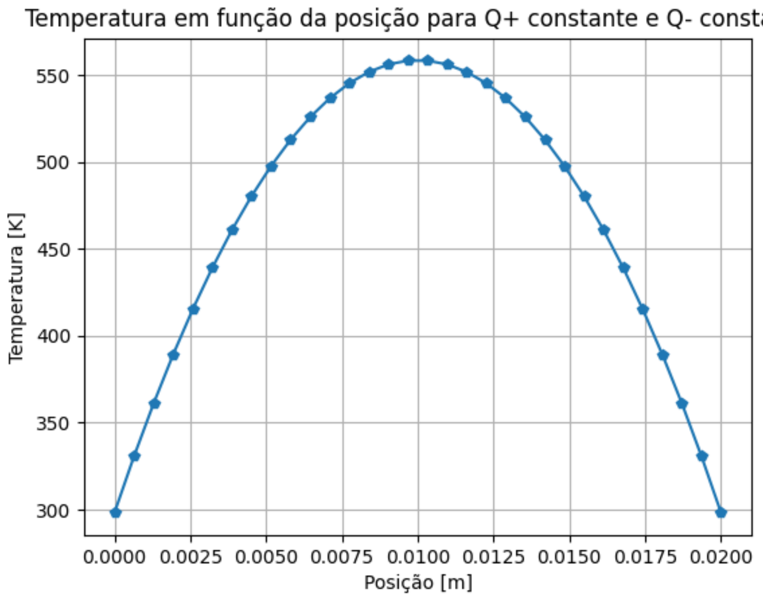

Figura 6 – T(x) para Q(x) constante

Percebe-se que a forma do gráfico é quadrática e exatamente a mesma do Caso 1, uma vez que a segunda derivada da temperatura se mantém constante com a posição. A única diferença é, portanto, na magnitude das temperaturas calculadas, que passou de $T_{máx} \approx 820\ K$ no Caso 1 para $T_{máx} \approx 560\ K$ no caso com resfriamento, ou seja, houve diminuição na temperatura interna do chip conforme o esperado.

4.3.3 Caso 3: Calor gerado dado por uma gaussiana e sem resfriamento

Desconsidera-se novamente o resfriamento do chip, mas agora é introduzida uma função para o calor gerado pelo chip. Tem-se:

$$\begin{cases} Q+ = Q_+^0 e^{-(x-L/2)^2/\sigma^2} \\ Q- = 0 \end{cases} \tag{4.3}$$

A função dada é uma Gaussiana, que representa que o chip esquenta mais no centro que nas bordas. Ainda, Q_+^0 representa o calor gerado no centro do chip, ao qual atribuiu-se o valor $Q_+^0 = 30/(8 \cdot 10^{-7})$.

Por fim, σ controla a variação da geração de calor em torno da parte central. Optou-se por estudar sua influência na curva de temperatura plotando $T(x)$ para $\sigma^2 = 1$, $\sigma^2 = 10^{-4}$ e $\sigma^2 = 10^{-6}$.

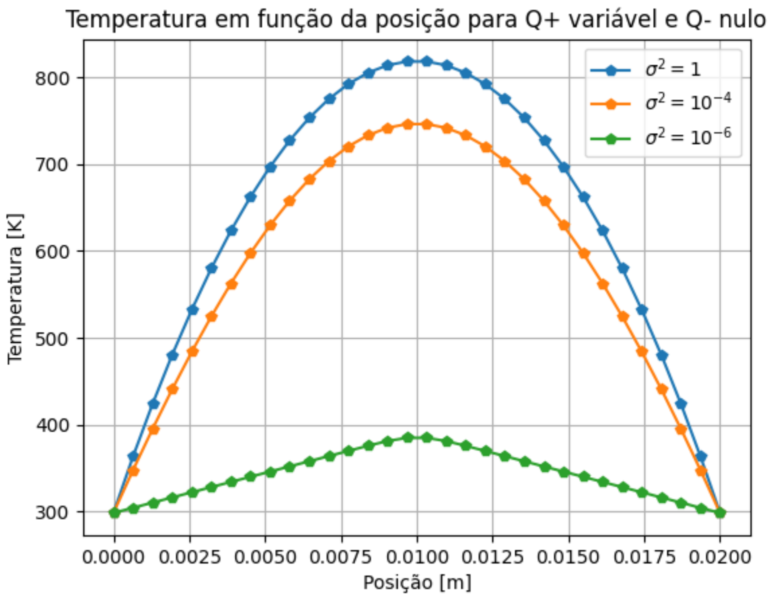

Figura 7 – T(x) para Q(x) gaussiano e sem resfriamento

Nesse caso, nota-se que quanto maior o valor de σ, maior a temperatura no centro do chip. Ainda, o aumento de σ aumenta também a concentração de temperatura mais ao centro que nas extremidades, conforme verifica-se nos gráficos.

4.3.4 Caso 4: Geração e dissipação de calor dados por gaussianas

Agora, introduz-se um refrigerador cuja dissipação de calor é dada por uma função também gaussiana, de forma que se tenha:

$$\begin{cases} Q+ = Q_+^0 e^{-(x-L/2)^2/\sigma^2} \\ Q- = Q_-^0 (e^{-(x^2)/\theta^2} + e^{-(x-L)^2/\theta^2}) \end{cases} \tag{4.4}$$

Primeiro, adotou-se $\sigma^2 = 1$ e $\theta^2 = 0.0005$ e o gráfico da temperatura obtido foi:

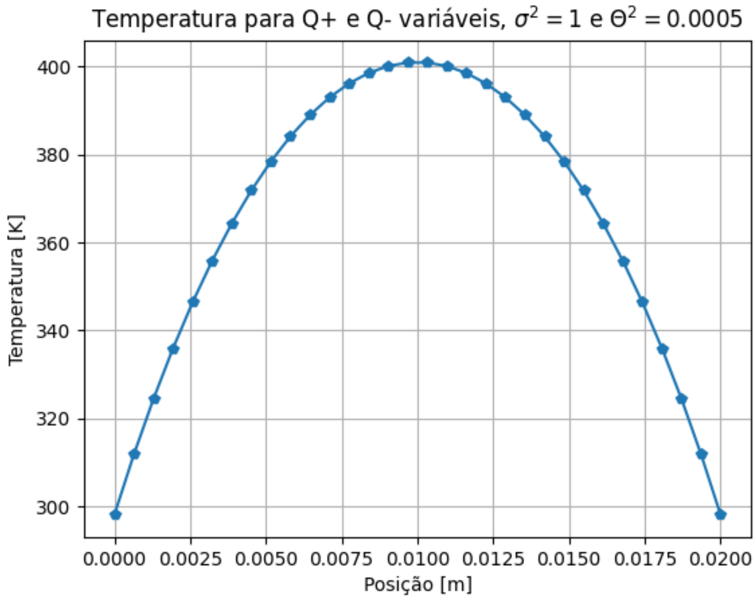

Figura 8 – T(x) para Q(x) gaussiano e com resfriamento

Percebe-se que para pequenos valores de θ, o calor dissipado não é muito influente na curva de $T(x)$, que se assemelha aos casos de quando o resfriamento é dado por uma constante.

Mas, ao plotar o gráfico para $\sigma^2 = 0.0005$ e $\theta^2 = 1$, nota-se que há uma maior dissipação na parte central do chip, tornando a temperatura nessas posições menores, o que altera o formato da curva:

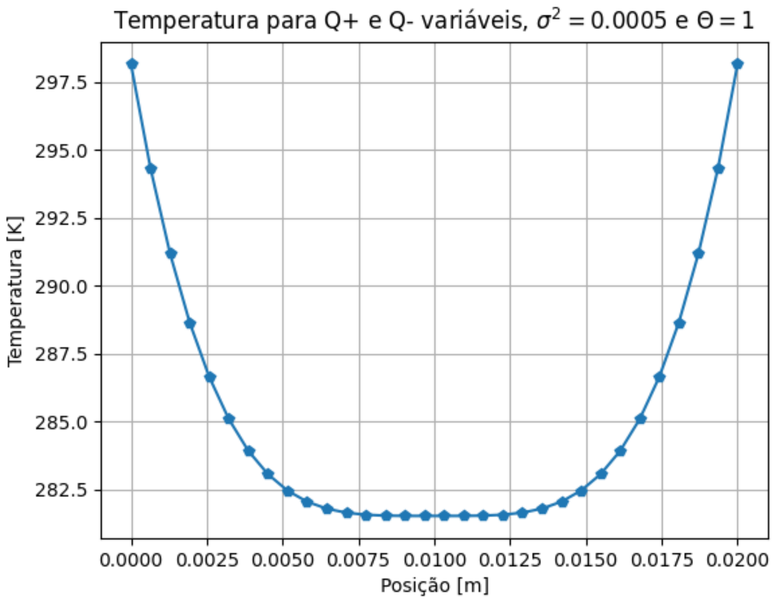

Figura 9 – T(x) para Q(x) gaussiano e com resfriamento

É notório que a dissipação de calor não acontece de forma natural, já que a temperatura final ficou menor que a determinada nas condições iniciais

4.4 Equilíbrio com variação de material

Por fim, estuda-se o caso em que o chip não é feito do mesmo material, ou seja, $k(x)$ não é constante e depende da posição:

$$k(x) = \begin{cases} k_s & \text{para } \frac{L}{2} - d < x < \frac{L}{2} + d \\ k_a \end{cases} \tag{4.5}$$

Para fazer a análise, considera-se um chip feito de silício ($k_s = 3,6W/(m \cdot K)$) envolto por alumínio ($k_a = 60W/(m \cdot K)$). Considerando $d = 0.0025m$ e uma geração Q_+ e dissipação Q_- constantes de calor. Desse modo, a distribuição de temperatura no chip fica:

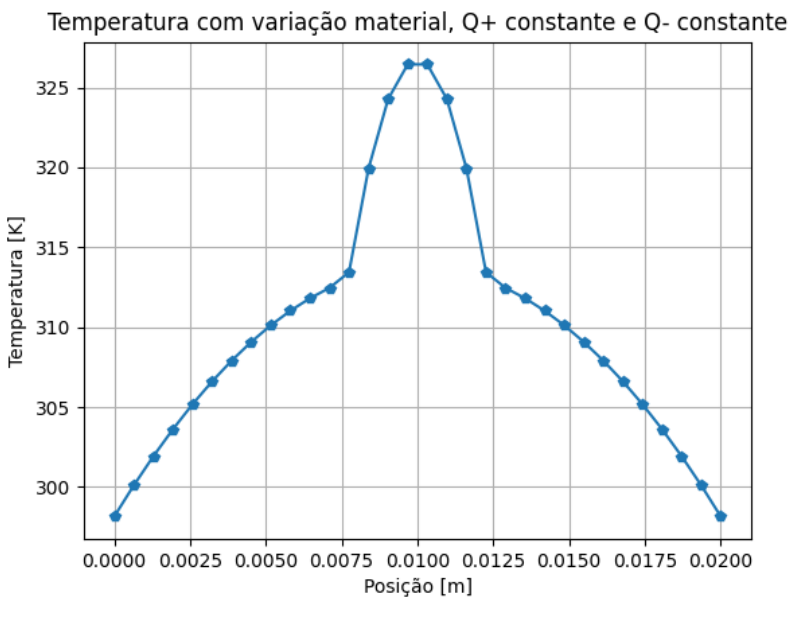

Figura 10 – T(x) com k(x) variável e Q(x) constante

Nota-se curvas bastantes distintas para o local em que o chip é envolto por alumínio, em que a temperatura é significativamente mais alta no centro, onde o material é silício. De fato, isso é esperado uma vez que a condutividade térmica do silício é mais baixa que a do alumínio, sendo mais sensível à mudança de temperatura quando exposto ao calor.

Da mesma forma, considerando o calor gerado e dissipado como funções gaussianas com $\sigma^2 = 0.0005$ e $\theta^2 = 1$, obteve-se o gráfico:

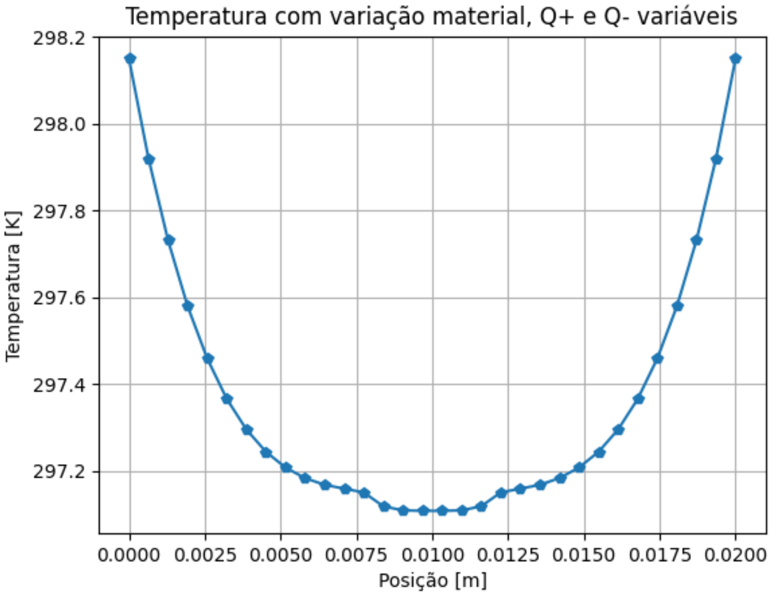

Figura 11 – T(x) com k(x) variável e Q(x) gaussiano

Nesse caso, como $\theta >> \sigma$, o calor dissipado é alto e faz com que o chip possua temperatura menor que a inicial. Assim, o centro fica mais frio, ao contrário do que ocorreu no exemplo anterior, uma vez que a condutividade térmica no centro do chip é menor e, por isso, sofre maior variação de temperatura em contato com o calor.

5 Conclusão

Em coerência com o proposto pelo trabalho, foi possível produzir um código para resolução de uma equação diferencial, utilizando métodos numéricos. Durante o processo, o resultado obtido passou por uma validação em um caso que a solução real era conhecida. Assim, concluiu-se que o código apresenta um erro variável para cada caso, mas, de qualquer forma, pequeno em relação as grandezas do problema.

Além disso, com o código validado, realizou-se algumas simulações para encontrar a temperatura do chip em determinadas condições ambientais. Inicialmente, aplicou-se a simulação, que utiliza do método de elementos finitos, para resolver um caso em que a distribuição de calor é uniforme.

Em sequência, complexidade foi sendo adicionada gradativamente e os resultados permaneceram em coerência com o esperado. Nesse sentido, é imperativo concluir a efetividade da utilização dos métodos aplicados. Vale ressaltar somente, o fato de que para erros muito pequenos, na ordem de 10^{-16}, um padrão quadrático não foi seguido, o que pode apontar para algum problema de arredondamento.

De qualquer forma, fica claro a ampla aplicação dos métodos usados para diversas outras áreas. O presente trabalho contou somente com a utilização para lei de Fourier, mas é possível aplicar também para cálculo estrutural em vigas flexionadas por exemplo.

Em suma, retoma-se a importância das ferramentas utilizadas na prática de engenharia, já que os métodos viabilizam a idealização de produtos ainda mais complexos e efetivos. Com isso, é possível cada vez mais superar problemas concretos da sociedade, utilizando a tecnologia.

Apêndices

APÊNDICE A – Código completo

```
import numpy as np
import matplotlib.pyplot as plt
def main():
    print('Código desenvolvido por Giovanna Girotto NUSP: 11803416 e Vittor Braide Costa NUSP:11806920')
    print('Para resolução do EP em questão, utilizou-se de três funções já criadas "decomposicao_LU", "solucao_sistema" desenvolvidas no EP1 e "integracao_gaussiana" desenvolvida no EP2')
    print('No desenvolvimento do EP3, foram criadas as funções "fourier_fem" que corresponde a aplicação dos método dos elementos finitos, "validacao", para executar os dois exercícios de validação propostos no enunciado, e "plot_temp_pos" para plotar os gráficos Temperatura x Posição')
    print('A demonstração dos resultados ocorre assim como proposto no enunciado, em que são apresentados gráficos do erro em função de h^2 e da Temperatura no chip adicionando-se complexidade para 3 diferentes casos')
    #EXERCÍCIO DE VALIDAÇÃO
    N=[7,15,31,63]
    validacao(N,0,0,1,lambda x: 0, lambda x: np.exp(x)+1,lambda x:0,lambda x: np.exp(x),lambda x: (x-1)*(np.exp(-x)-1),'$u(x)=(x-1)(e^{-x}-1)$')
    validacao(N,0,0,1,lambda x: 0, lambda x: 12*x*(1-x)-2,lambda x:0,lambda x: 1,lambda x: (x**2)*(1-x)**2,'$u(x)=12x(1-x)-2$')
    #EXERCÍCIOS PARA K CONSTANTE
    #CONDIÇÕES NAO HOMOGENEAS - L=20mm, n=100, k=3.6 W/mK a=b=293.15K, P=30V V=20*20*2 mm^3
    #Caso 1 - Sendo Q+ e Q- constantes
    #Caso 1a - Q+=P/V e Q-=0
    L=0.02
    n=30
    k=3.6
    T_caso_1a=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,lambda x:30/(8*(10**(-7))),lambda x:0,lambda x: k)
    plot_temp_pos(L,n,[T_caso_1a],'Temperatura em função da posição para Q+ constante e Q- nulo','')

    #Caso 1b - Q+=P/V e Q-=15/V
    T_caso_1b=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,lambda x:30/(8*(10**(-7))),lambda x: 15/(8*(10**(-7))),lambda x: k)
    plot_temp_pos(L,n,[T_caso_1b],'Temperatura em função da posição para Q+ constante e Q- constante','')

    #Caso 2 - Sendo Q+ dado pela expressão disponível no enunciado e Q-
```

```
nulo
Q0=30/(8*(10**(-7)))
T_caso_2a=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,lambda x: Q0*np.exp(-((x-L/2)**2)/1),lambda x:0,lambda x: k)
T_caso_2b=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,lambda x: Q0*np.exp(-((x-L/2)**2)/(10**(-4))),lambda x:0,lambda x: k)
T_caso_2c=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,lambda x: Q0*np.exp(-((x-L/2)**2)/(10**(-6))),lambda x:0,lambda x: k)
plot_temp_pos(L,n,[T_caso_2a,T_caso_2b,T_caso_2c],'Temperatura em função da posição para Q+ variável e Q- nulo',['$\sigma^2=1$','$\sigma^2=10^{-4}$','$\sigma^2=10^{-6}$'])

#Caso 3 - Q+ e Q- dados pelo expressão disponível no enunciado
Q0_pos=30/(8*(10**(-7)))
Q0_neg=15/(8*(10**(-7)))
theta_qua=0.0005
s_qua=1
Q_ger=lambda x: Q0_pos*np.exp(-((x-L/2)**2)/s_qua)
Q_dis=lambda x: Q0_neg*(np.exp(-((x)**2/theta_qua))+np.exp(-((x-L)**2)/theta_qua))
T_caso_3a=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,Q_ger,Q_dis,lambda x: k)
plot_temp_pos(L,n,[T_caso_3a],'Temperatura para Q+ e Q- variáveis, $\sigma^2=1$ e $\Theta^2=0.0005$','')
theta_qua=1
s_qua=0.0005
T_caso_3b=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,Q_ger,Q_dis,lambda x: k)
plot_temp_pos(L,n,[T_caso_3b],'Temperatura para Q+ e Q- variáveis, $\sigma^2=0.0005$ e $\Theta=1$','')

#PROBLEMA COM K VARIÁVEL
#Caso 1 - Q+ constante e Q- constante
ks=3.6
ka=60
d=0.0025
T_k_variavel1=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,lambda x:30/(8*(10**(-7))),lambda x:15/(8*(10**(-7))),lambda x: ks if L/2-d<x<L/2+d else ka)
plot_temp_pos(L,n,[T_k_variavel1],'Temperatura com variação material, Q+ constante e Q- constante','')
#Caso 2 - Q+ dado pelo enunciado e Q- constante
s_qua=10**(-4)
Q_ger=lambda x: Q0_pos*np.exp(-((x-L/2)**2)/s_qua)
T_k_variavel2=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,Q_ger,lambda x:15/(8*(10**(-7))),lambda x: ks if L/2-d<x<L/2+d else ka)
plot_temp_pos(L,n,[T_k_variavel2],'Temperatura com variação material, Q
```

```
    + variável e Q- constante','')
    #Caso 3 - Q+ e Q- dados pelo enunciado
    theta_qua=1
    s_qua=0.0005
    Q_ger=lambda x: Q0_pos*np.exp(-((x-L/2)**2)/s_qua)
    Q_dis=lambda x: Q0_neg*(np.exp(-((x)**2/theta_qua))+np.exp(-((x-L)**2)/theta_qua))
    T_k_variavel3=fourier_fem(25+273.15,25+273.15,L,n,lambda x:0,Q_ger,Q_dis,lambda x: ks if L/2-d<x<L/2+d else ka)
    plot_temp_pos(L,n,[T_k_variavel3],'Temperatura com variação material, Q+ e Q- variáveis','')


def fourier_fem(a,b,L,n,q,Q_ger,Q_dis,k):
    h=L/(n+1)
    f=lambda x: Q_ger(x)-Q_dis(x)
    #x0=0, x_n+1=L
    X=np.arange(0,L+h,h)
    D_1,D_2,D_3,D_4,D_5,D_6=np.zeros(n-1),np.zeros(n),np.zeros(n),np.zeros(n+1),np.zeros(n),np.zeros(n)
    alfa1, alfa2, alfa0=np.zeros(n),np.zeros(n),np.zeros(n)
    #Definindo Q2,n; Q3,n; Q4,n; Q4,n+1; Q5,n; Q6,n
    D_2[n-1]=((1/h)**2)*integracao_gaussiana(X[n-1],X[n],lambda x: ((x-X[n-1])**2)*q(x))
    D_3[n-1]=((1/h)**2)*integracao_gaussiana(X[n],X[n+1],lambda x: ((X[n+1]-x)**2)*q(x))
    D_4[n-1]=((1/h)**2)*integracao_gaussiana(X[n-1],X[n],lambda x: k(x))
    D_4[n]=((1/h)**2)*integracao_gaussiana(X[n],X[n+1],lambda x: k(x))
    D_5[n-1]=(1/h)*integracao_gaussiana(X[n-1],X[n],lambda x: (x-X[n-1])*f(x))
    D_6[n-1]=(1/h)*integracao_gaussiana(X[n],X[n+1],lambda x: (X[n+1]-x)*f(x))
    #Definindo Q1,n; Q2,i; Q3,n; Q4,n; Q5,n; Q6,n;
    #X[i] tem uma peculiaridade já que existe X[0]
    for i in range(1,n):
        D_1[i-1]=((1/h))**2*integracao_gaussiana(X[i],X[i+1],lambda x: (X[i+1]-x)*(x-X[i])*q(x))
        D_2[i-1]=((1/h))**2*integracao_gaussiana(X[i-1],X[i],lambda x: ((x-X[i-1])**2)*q(x))
        D_3[i-1]=((1/h)**2)*integracao_gaussiana(X[i],X[i+1],lambda x: ((X[i+1]-x)**2)*q(x))
        D_4[i-1]=((1/h)**2)*integracao_gaussiana(X[i-1],X[i],lambda x: k(x))
        D_5[i-1]=(1/h)*integracao_gaussiana(X[i-1],X[i],lambda x: (x-X[i-1])*f(x))
        D_6[i-1]=(1/h)*integracao_gaussiana(X[i],X[i+1],lambda x: (X[i+1]-x)*f(x))
```

```
    for i in range(1,n):
        alfa1[i-1]=D_4[i-1]+D_4[i]+D_2[i-1]+D_3[i-1]
        alfa2[i-1]=-D_4[i]+D_1[i-1]
        #loop para i=2,3,...,n
        alfa0[i]=-D_4[i]+D_1[i-1]
    alfa0[0]=0
    alfa1[n-1]=D_4[n-1]+D_4[n]+D_2[n-1]+D_3[n-1]
    alfa2[n-1]=0
    beta=D_5+D_6
    c=solucao_sistema([alfa0,alfa1,alfa2],beta)
    sum_phi=np.zeros(n)
    for i in range(1,n+1):
        #calculo phi_i
        for j in range(1,n+1):
            if X[i]>X[j-1] and X[i]<=X[j]:
                phi=(1/h)*(X[i]-X[j-1])
            elif X[i]>X[j] and X[i]<=X[j+1]:
                phi=(1/h)*(X[j+1]-X[i])
            else:
                phi=0
            sum_phi[i-1]+=c[j-1]*phi
        #Adicionar casos não homogêneos
        sum_phi[i-1]+=a+((b-a)/L)*X[i]
    sum_phi=np.insert(sum_phi,0,a)
    sum_phi=np.insert(sum_phi,n+1,b)
    return sum_phi

def validacao(N,a,b,L,q,Q_ger,Q_dis,k,u,str_sol_exata):
    h_qua=np.zeros(len(N))
    error=np.zeros(len(N))
    X_exata=np.arange(0,L+0.01,0.01)
    sol_plot=np.zeros(len(X_exata))
    for l in range(len(X_exata)):
        sol_plot[l]=u(X_exata[l])
    for i in range(len(N)):
        h=L/(N[i]+1)
        sol_exata=np.zeros(N[i]+2)
        X=np.arange(0,L+h,h)
        h_qua[i]=(L/(N[i]+1))**2
        sol_numerica=fourier_fem(a,b,L,N[i],q,Q_ger,Q_dis,k)
        for j in range(0,N[i]+1):
            sol_exata[j]=u(X[j])
        error[i]=max(abs(sol_numerica-sol_exata))
        plt.plot(X_exata,sol_plot,'-',label="Solução exata ($u(x)$)")
        plt.plot(X,sol_numerica,'p',label="Solução numérica ($\overline{u}(
    x)$)")
        plt.legend()
```

```
        plt.title('Paridade da simulação e solução exata '+str_sol_exata+'
 para n=%i '%N[i])
        plt.xlabel('x')
        plt.ylabel('Solução')
        plt.grid()
        plt.show()
    plt.plot(h_qua, error, '-p')
    plt.title('Erro da simulação para solução exata '+str_sol_exata,y=1.08)
    plt.xlabel('$h^2$')
    plt.ylabel('Erro')
    plt.grid()
    plt.show()

def plot_temp_pos(L,n,T,title,legenda):
    h=L/(n+1)
    X=np.arange(0,L+h,h)
    if len(T)>1:
        for i in range(len(T)):
            plt.plot(X,T[i],'-p',label=legenda[i])
            plt.legend()
    else:
        plt.plot(X,T[0],'-p')
    plt.title(title)
    plt.xlabel('Posição [m]')
    plt.ylabel('Temperatura [K]')
    plt.grid()
    plt.show()

def decomposicaoLU(A):
    a,b,c=A[0],A[1],A[2]
    U,L=np.zeros(len(A[0])),np.zeros(len(A[0])) #cria os vetores U e L
 vazios
    #Valores iniciais
    U[0]=b[0]
    L[0]=1
    #Laço para encontrar L,U, assim como disponibilizado no enunciado
    for i in range(1,len(A[0])):
        L[i]=a[i]/U[i-1]
        U[i]=b[i]-L[i]*c[i-1]
    return L,U

def solucao_sistema(A,d):
    LU=decomposicaoLU(A)
    L,U=LU[0],LU[1]
    x,y=np.zeros(len(d)),np.zeros(len(d)) #cria os vetores x e y vazios
    y[0]=d[0]
    for i in range(1,len(d)):
```

```
            y[i]=d[i]-L[i]*y[i-1]
    x[len(d)-1]=y[len(d)-1]/U[len(d)-1]
    for i in range(len(d)-2,-1,-1):
            x[i]=(y[i]-A[2][i]*x[i+1])/U[i]
    return x

def integracao_gaussiana(a,b,f):
    W=(1,1) #pesos
    p=(-1/(np.sqrt(3)),1/(np.sqrt(3))) #nós
    n=2
    I1=0
    for i in range(n):
            x=((b-a)*p[i]+(a+b))/2
            u=(b-a)*W[i]/2
            I1+=u*f(x)
    return I1
main()
```