Ariel Guerreiro - 11257838
Felipe Azank - 11258137

# 1º Exercício Programa

PMR3401

São Paulo, 2022

# Conteúdo

# 1 Método de Runge Kutta

## 1.1 Descrição do problema

O circuito elétrico de um forno pode ser conferido na Figura 1. O objetivo deste exercício é encontrar as correntes elétricas $i_1(t)$, $i_2(t)$ e a carga elétrica $q(t)$ do capacitor C ao longo do tempo, dada uma fonte de tensão $e(t)$.

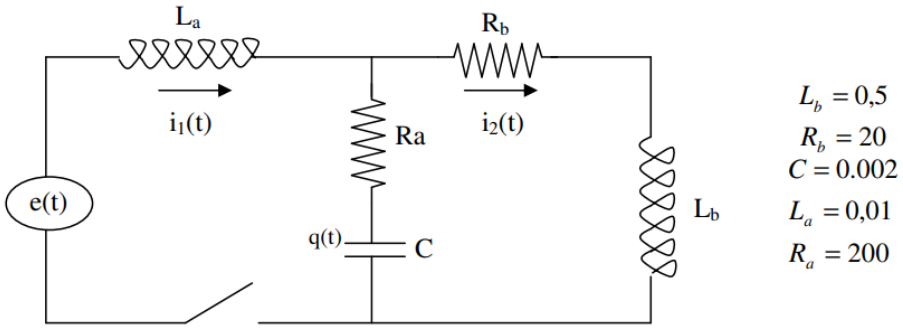



Figura 1: Circuito elétrico do forno

É possível representar o circuito por meio das seguintes equações:

$$\begin{cases} L_a \frac{di_1}{dt} + R_a(i_1 - i_2) + \frac{q(t)}{C} = e(t) \\ -\frac{q(t)}{C} - R_a(i_1 - i_2) + R_b i_2 + L_b \frac{di_2}{dt} = 0 \\ q(t) = \int_0^t (i_1(t') - i_2(t'))dt' + q(0) \end{cases} \tag{1}$$

Onde $e(t) = \cos(600t)/L_a$ e $i_1(0) = i_1(0) = q(0) = 0$

## 1.2 Equacionamento e Modelagem computacional

É possível reescrever o sistema de equações de modo a evidenciar as derivadas a serem calculadas. Em especial, a expressão da carga do capacitor foi derivada:

$$\begin{cases} \frac{di_1}{dt} = \frac{1}{L_a}\big( - R_a(i_1 - i_2) - \frac{q(t)}{C} + e(t)\big) \\ \frac{di_2}{dt} = \frac{1}{L_b}\big(\frac{q(t)}{C} + R_a(i_1 - i_2) - R_b i_2 \\ \frac{dq}{di} = i_1(t) - i_2(t) \end{cases} \tag{2}$$

Para a resolução deste sistema de equações diferenciais, é possível utilizar o método de Runge Kutta de 4ª Ordem (RK Clássico) em sua forma matricial para resolver o sistema de forma iterativa, partindo da condição inicial e calculando o instante seguinte em função do anterior e da função de derivada $[f]$ e do passo $h$. O cálculo dos valores $Y_{i+1}$, em um instante $(i+1)$, é dado por:

$$[Y]_{i+1} = [Y]_i + \frac{h}{6}\Big([K_1] + 2[K_2] + 2[K_3] + [K_4]\Big) \tag{3}$$

Onde:

$$\begin{cases} [K_1] = [F(x_i, [Y]_i)] \\ [K_2] = [F(x_i + \frac{h}{2}, [Y]_i + \frac{h}{2}[K_1])] \\ [K_3] = [F(x_i + \frac{h}{2}, [Y]_i + \frac{h}{2}[K_2])] \\ [K_4] = [F(x_i + h, [Y]_i + h[K_3])] \end{cases}$$

Para o problema do circuito elétrico, temos $x_i = t_i$ e:

$$[Y]_i = \begin{bmatrix} i_1(t_i) \\ i_2(t_i) \\ q(t_i) \end{bmatrix} \qquad\qquad [Y]_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

Temos também o vetor $[F(x_i, [Y]_i)] = \frac{d}{dt}[Y]_i$, determinado por meio das equações em (2):

$$[F(x_i, [Y]_i)] = \begin{bmatrix} \frac{1}{L_a}(e(t) - R_a(Y[1] - Y[2]) - Y[0]/C) \\ \frac{1}{L_b}(Ra(Y[1] - Y[2]) - R_b Y[2] + Y[0]/C) \\ Y[1] - Y[2] \end{bmatrix} \tag{5}$$

Onde $Y[0],\ Y[1],\ Y[2]$ representam $i_1(t_i),\ \imath_2(t_i),\ q(t)$ em um instante $t_i$.

O algoritmo foi implementado na linguagem de programação Python, com o auxílio das bibliotecas Numpy e Matplotlib. A listagem do código que implementa o algoritmo RK4 conforme descrito pela equação (3) pode ser conferido no Listing 1, enquanto o código utilizado para resolver o circuito elétrico pode ser conferido no Listing 2.

## 1.3 Resultados e Discussões

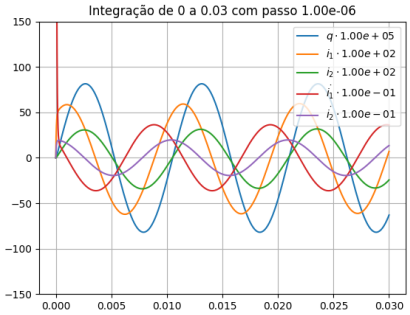
### 1.3.1 Análise da influência do passo

Para as propriedades descritas na Figura 1, buscou-se avaliar a influência do passo "h"na solução do sistema de equações no intervalo de 0 a 0,03s, buscando avaliar o comportamento da solução, que é esperado como periódico, além da magnitude das propriedades. Neste problema, os passos adotados podem ser conferidos na Tabela 1.

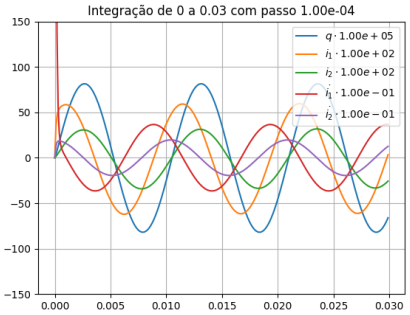
| | Pequeno | Médio | Grande |
|---|---|---|---|
| Passo (h) | 1e-6 | 0.0001 | 0.01 |

Tabela 1: Valores de passo utilizados

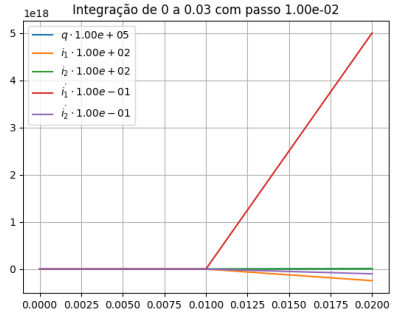
Os gráficos dos resultados, contendo $i_1(t), i_1\dot{(t)}, i_2(t), i_2\dot{(t)}, q(t)$ podem ser conferidos na Figura 2.




(a) Passo pequeno




(b) Passo médio




(c) Passo grande

Figura 2: Resultados do RK4 com os diferentes passos

É possível observar que os resultados com passos médio e pequeno estão muito próximos, mas com um custo computacional e tempo de execução muito superior para o passo pequeno. Por outro lado, é possível verificar que o passo grande não foi capaz de convergir para uma solução, o que se justifica pela frequência da tensão elétrica, e pelo intervalo de solução. O período da tensão fornecida ao sistema é aproximadamente $0,01$, portanto, é de se esperar que um passo da mesma magnitude não seja capaz de representar adequadamente o comportamento oscilatório do sistema.

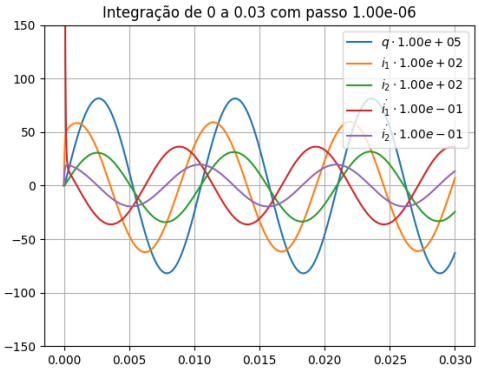
Por meio dos resultados obtidos, conclui-se que o valor ideal de passo, nas condições impostas, é de $h = 1e - 4$, por se tratar de um valor capaz de representar o sistema corretamente, ao mesmo tempo que não ocorre em um custo computacional alto como o passo pequeno.

### 1.3.2 Comparação com outros parâmetros elétricos

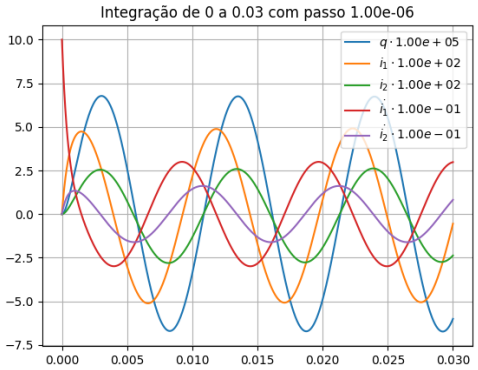
Resolveu-se o sistema novamente, para:

- Mesmas constantes da Figura 1, com $L_a = 0,1$
- Mesmas constantes da Figura 1, com $R_a = 2000$

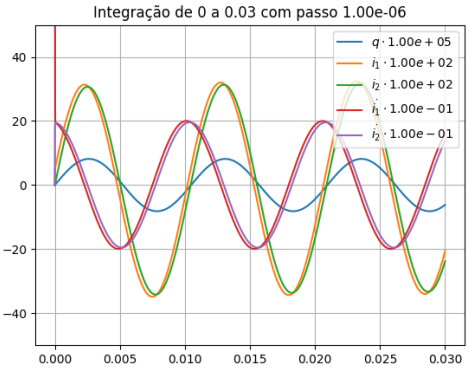
Inicialmente, adotou-se o passo médio. Porém, para a condição de $R_a = 2000$, o sistema só foi capaz de convergir com o passo pequeno. Dessa forma, utilizou-se o passo pequeno para todas as situações. Os resultados podem ser conferidos na Figura 3.




(a) Condições originais




(b) $L_a = 0,1$




(c) $R_a = 2000$

Figura 3: Comportamento do sistema para cada condição imposta

É possível destacar que o aumento da indutância $L_a$ levou a uma grande diminuição da magnitude dos sinais, visto que a tensão elétrica fornecida ao sistema foi reduzida. Em relação ao sistema original, a diferença de fase entre as correntes se manteve aproximadamente igual, e a relação do valor de carga no capacitor com as correntes elétricas aumentou.

Para o aumento da resistência $R_a$, nota-se, em relação ao sistema original, uma diminuição na magnitude das correntes, mas com valores superiores ao caso anterior. O efeito mais relevante dessa alteração é a redução da diferença de fase entre as correntes e o decréscimo da carga do capacitor em relação à corrente. Estes comportamentos são esperados, pois o aumento de $R_a$ leva à uma queda de tensão no capacitor e também ao aumento do fator de potência, tornando as características resistivas do sistema mais relevantes do que as características indutivas e capacitivas.

# 2 Método de Diferenças Finitas

## 2.1 Descrição do problema

O circuito elétrico descrito na seção 1 é responsável por acionar um forno industrial, cuja estrutura pode ser conferida na Figura 4. Por se tratar de um forno elétrico, a diferença de potencial imposta gera uma corrente elétrica no sistema. A resistência elétrica proveniente dos materiais do forno e fundido geram, por efeito Joule, dissipação de calor, gerando uma distribuição de temperaturas. O objetivo deste problema é modelar e avaliar o comportamento elétrico e térmico do forno por meio de técnicas de Métodos de Diferenças Finitas (MDF).

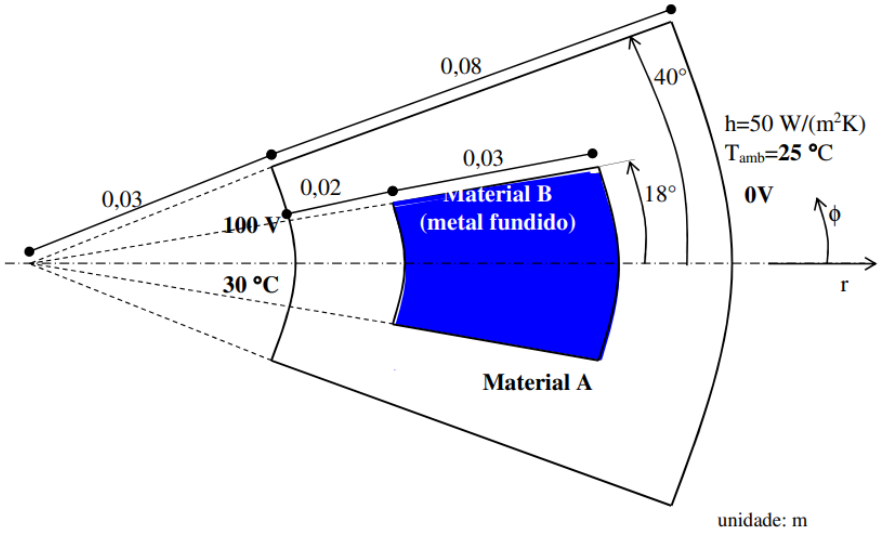



Figura 4: Peça do forno a ser modelada

A equação que rege o comportamento do potencial elétrico na peça é a equação de Laplace, que pode ser conferida em coordenas cilíndricas na Equação 6, onde $\sigma$ é a condutividade elétrica do meio, com $\sigma_A = 5 \cdot 10^{-6}$ e $\sigma_B = 1 \cdot 10^{-5}$.

$$\sigma \nabla^2 V = 0 \Rightarrow \sigma \left[ \frac{\partial^2 V}{\partial r^2} + \frac{1}{r}\frac{\partial V}{\partial r} + \frac{1}{r^2}\frac{\partial^2 V}{\partial \phi^2} \right] = 0 \tag{6}$$

Com a distribuição de potencial elétrico conhecida, é possível determinar a densidade de corrente elétrica $J$ (Equação 7) e a potência elétrica dissipada por efeito Joule $\dot{q}$ (Equação 8).

$$J = -\sigma \nabla V = -\sigma \left( \frac{\partial V}{\partial r}, \frac{1}{r} \frac{\partial V}{\partial \phi} \right) \tag{7}$$

$$\dot{q} = -\frac{|J|^2}{\sigma} \tag{8}$$

Conhecida a potência elétrica dissipada, é possível então encontrar a distribuição de temperaturas no sistema por meio da Equação 9, onde $k$ é a condutividade térmica do material, com $k_A = 110W/(Km)$ e $k_B = 500W/(Km)$

$$k \left[ \frac{\partial^2 V}{\partial r^2} + \frac{1}{r} \frac{\partial V}{\partial r} + \frac{1}{r^2} \frac{\partial^2 V}{\partial \phi^2} \right] + \dot{q} = 0 \tag{9}$$

## 2.2 Equacionamento e Modelagem computacional

### 2.2.1 Comportamento elétrico

Inicialmente, para resolver a equação diferencial parcial descrita e encontrar os valores do potencial elétrico, utilizou-se MDF para discretizar a equação em uma malha de pontos, aproximando as derivadas para diferenças divididas. Para a discretização em coordenadas polares, é necessário definir um valor de discretização $\Delta r$ para o raio e $\Delta \phi$ para o ângulo.

Pela simetria da em torno do eixo $r$, é possível simplificar a solução para somente metade da peça da Figura 4, espelhando o resultado para a outra parte. É possível definir 10 condições em que a equação deve ser discretizada, de acordo com as propriedades físicas do problema. A discretização adotada pode ser conferida na Figura 5.

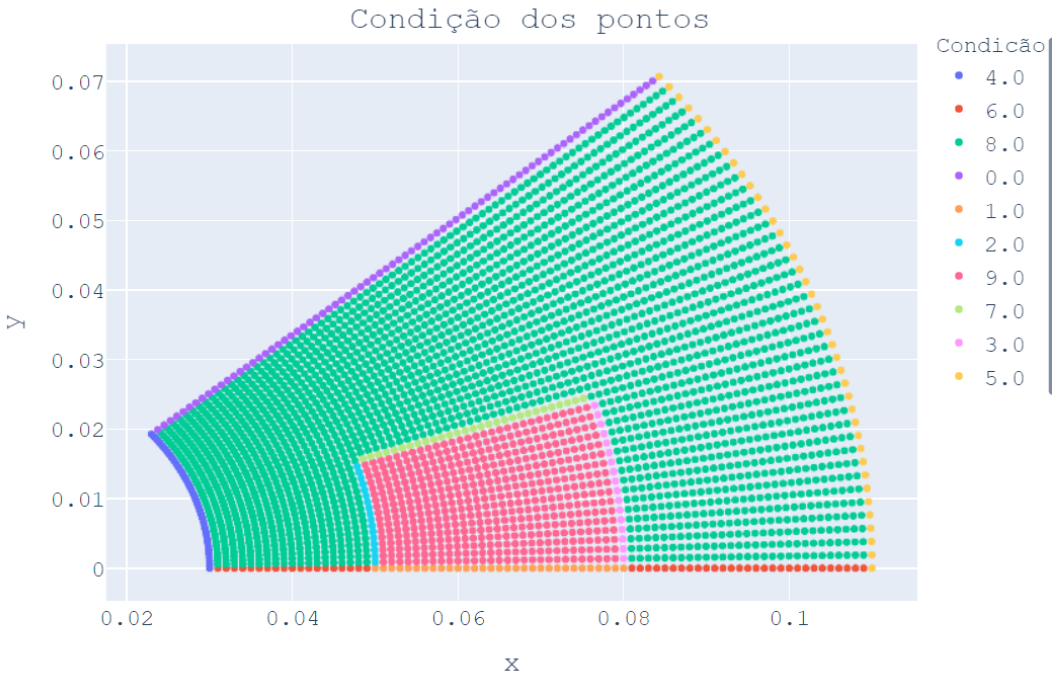



Figura 5: Discretização da peça do forno

A modelagem de cada condição pode ser conferida a seguir:

**Condição 0: borda superior de A**

A primeira e a segunda derivadas na direção radial podem ser aproximadas pelas expressões de diferenças finitas centrais. Neste trabalho, adotou-se o índice $i$ para representar a coordenada radial e o índice $j$ para a coordenada angular.

$$\left.\frac{\partial V}{\partial r}\right|_{i,j} = \frac{V_{i+1,j} - V_{i-1,j}}{2\Delta r} \qquad \left.\frac{\partial^2 V}{\partial^2 r}\right|_{i,j} = \frac{V_{i+1,j} - 2V_{i+1,j} + V_{i-1,j}}{\Delta r^2}$$

Por não possuir o valor $V_{i,j+1}$ na borda superior do material A, é necessário aproximar a expressão da derivada na direção $\phi$ pela regra de Taylor. Pelas condições de contorno do problema, temos que $\left.\frac{\partial V}{\partial \phi}\right|_{i,j} = 0$ nesta região. Por Taylor:

$$V_{i,j-1} = V_{i,j} - \Delta r \left.\frac{\partial V}{\partial \phi}\right|_{i,j} + \frac{\Delta r^2}{2}\left.\frac{\partial^2 V}{\partial \phi^2}\right|_{i,j} \Rightarrow \left.\frac{\partial^2 V}{\partial \phi^2}\right|_{i,j} = \frac{2(V_{i,j-1} - V_{i,j})}{\Delta r^2}$$

Substituindo as aproximações das derivadas na Equação 6, temos que, para a condição 0, com $r_{i,j}$ sendo o raio na coordenada (i, j):

$$V_{i,j} = \frac{V_{i+1,j}\left(\Delta r\Delta\phi^2 r_{ij} + 2\Delta\phi^2 r_{ij}^2\right) + 4V_{i,j-1}\Delta r^2 + V_{i-1,j}\left(-\Delta r\Delta\phi^2 r_{ij} + 2\Delta\phi^2 r_{ij}^2\right)}{4\left(\Delta r^2 + \Delta\phi^2 r_{ij}^2\right)} \tag{10}$$

**Condição 1: borda inferior de B (região de simetria)**

O processo de simetria é semelhante ao desenvolvido para a Condição 1. As derivadas centrais para a direção r podem ser utilizadas, mas a derivada central na direção $\phi$ pode ser simplificada ao se observar a simetria da figura ($V_{i,j-1} = V_{i,j+1}$):

$$\frac{\partial^2 V}{\partial\phi^2}\Big|_{i,j} = \frac{V_{i+1,j} - 2V_{i+1,j} + V_{i-1,j}}{\Delta\phi^2} \Rightarrow \frac{\partial^2 V}{\partial\phi^2}\Big|_{i,j} = \frac{2V_{i+1,j} - 2V_{i+1,j}}{\Delta\phi^2}$$

Substituindo as expressões das derivadas na Equação 6, temos:

$$V_{i,j} = \frac{V_{i+1,j}\left(\Delta r\Delta\phi^2 r_{ij} + 2\Delta\phi^2 r_{ij}^2\right) + 4V_{i,j+1}\Delta r^2 + V_{i-1,j}\left(-\Delta r\Delta\phi^2 r_{ij} + 2\Delta\phi^2 r_{ij}^2\right)}{4\left(\Delta r^2 + \Delta\phi^2 r_{ij}^2\right)} \tag{11}$$

**Condição 2: borda esquerda de B**

Para a borda esquerda de B, é necessário desenvolver a equação para o meio A e para o meio B, usando a condição de continuidade imposta do problema, já que $V_{i,j-1}$ está no meio A e $V_{i,j+1}$. Expandindo por Taylor, pelo meio A, é possível encontrar uma expressão de $\frac{\partial^2 V}{\partial^2 r}\Big|_{i,j}$ para o meio A:

$$\frac{\partial^2 V}{\partial^2 r}\Big|_{i,j} = \frac{2}{\Delta r^2}\left(V_{i-1,j} - V_{i,j} + \frac{\partial V}{\partial r}\Big|_{i,j}^{A}\right) \tag{12}$$

Utilizando a fórmula de diferença central para a coordenada $\phi$ e as fórmulas para as derivadas em $r$, é possível desenvolver uma expressão para $\frac{\partial V}{\partial r}\Big|_{i,j}^{A}$:

$$\frac{\partial V}{\partial r}\Big|_{i,j}^{A}\left(\frac{2}{\Delta r} + \frac{1}{r}\right) = \frac{-2}{\Delta r^2}(V_{i-1,j} - V_{i,j}) - \frac{1}{r_{i,j}^2}\frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{\Delta\phi^2}$$

De forma análoga, é possível, por expansão em Taylor, encontrar uma expressão para $\frac{\partial V}{\partial r}\Big|_{i,j}^{B}$ pelo meio B:

$$\frac{\partial V}{\partial r}\Big|_{i,j}^{B}\left(\frac{-2}{\Delta r} + \frac{1}{r}\right) = \frac{-2}{\Delta r^2}(V_{i+1,j} - V_{i,j}) - \frac{1}{r_{i,j}^2}\frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{\Delta\phi^2}$$

Temos, da condição de continuidade dos meios, $\sigma_A \frac{\partial V}{\partial r}\Big|_{i,j}^{A} = \sigma_B \frac{\partial V}{\partial r}\Big|_{i,j}^{B}$. Definindo $\alpha = \sigma_A(\frac{2}{\Delta r} + \frac{1}{r})$ e $\beta = \sigma_B(\frac{-2}{\Delta r} + \frac{1}{r})$, é possível chegar na seguinte expressão para $V_{i,j}$:

$$V_{i,j} = \frac{2r_{ij}\Delta\phi^2(\alpha V_{i-1,j} - \beta V_{i+1,j}) + (\alpha - \beta)\Delta r^2(V_{i,j+1} + V_{i,j-1})}{2(\alpha - \beta)(r_{ij}^2\Delta\phi^2 + \Delta r^2)} \tag{13}$$

**Condição 3: borda direita de B**

Para a borda direita de B, também é necessário fazer a condição de continuidade entre os meios, com desenvolvimento análogo à condição 2. Para simplificação, é possível adotar a Equação 13 na borda direita, adotando $\alpha = \sigma_B(\frac{-2}{\Delta r} + \frac{1}{r})$ e $\beta = \sigma_A(\frac{2}{\Delta r} + \frac{1}{r})$.

**Condição 4: borda esquerda de A e Condição 5: borda direita de A**

Temos, pelas condições de contorno, que, na borda esquerda de A, $V_{i,j} = 100\ V$ e na borda direita de A, $V_{i,j} = 0\ V$.

**Condição 6: borda inferior de A (região de simetria)**

Para a região de simetria da borda inferior de A, o equacionamento é idêntico ao desenvolvido para a condição 1. Dessa forma, a equação final para este caso também é a Equação 11.

**Condição 7: borda superior de B**

A borda superior de B tem desenvolvimento análogo às condições 2 e 3, mas, nesta condição, a continuidade é dada por $\sigma_A \frac{\partial V}{\partial \phi}\Big|_{i,j}^{A} = \sigma_B \frac{\partial V}{\partial \phi}\Big|_{i,j}^{B}$. Expandindo por Taylor para determinar uma expressão de $\frac{\partial^2 V}{\partial \phi^2}\Big|_{i,j}$ e substituindo na Equação 6, tanto para A quanto para B, e utilizando a equação de continuidade, chegamos na seguinte equação:

$$V_{i,j} = \frac{(\Delta\phi^2\Delta r\sigma_A r_{ij} + \Delta\phi^2\Delta r\sigma_B r_{ij} + 2\Delta\phi^2\sigma_A r_{ij}^2 + 2\Delta\phi^2\sigma_B r_{ij}^2)V_{i+1,j} + 4\Delta r^2\sigma_A V_{i,j+1}}{4(\sigma_A + \sigma_B)(\Delta\phi^2 r_{ij}^2 + \Delta r^2)} + \frac{(-\Delta\phi^2\Delta r\sigma_A r_{ij} - \Delta\phi^2\Delta r\sigma_B r_{ij} + 2\Delta\phi^2\sigma_A r_{ij}^2 + 2\Delta\phi^2\sigma_B r_{ij}^2)V_{i-1,j} + 4\Delta r^2\sigma_B V_{i,j-1}}{4(\sigma_A + \sigma_B)(\Delta\phi^2 r_{ij}^2 + \Delta r^2)} \tag{14}$$

**Condição 8: interior de A**

Para o interior do material A, é possível utilizar as diferenças finitas centrais nas coordenadas r e $\phi$. Dessa forma, a equação fica:

$$V_{i,j} = \frac{V_{i+1,j}\left(\Delta r\Delta\phi^2 r_{ij} + 2\Delta\phi^2 r_{ij}^2\right) + 2V_{i,j+1}\Delta r^2 + 2V_{i,j-1}\Delta r^2 + V_{i-1,j}\left(-\Delta r\Delta\phi^2 r_{ij} + 2\Delta\phi^2 r_{ij}^2\right)}{4\left(\Delta r^2 + \Delta\phi^2 r_{ij}^2\right)} \tag{15}$$

**Condição 9: interior de B**

De forma análoga à condição 8, é possível utilizar diretamente as fórmulas de diferenças finitas centrais. Para esta condição, o equacionamento final também é a Equação 15.

Cada expressão de $V_{i,j}$ foi utilizada para calcular o valor da tensão real de forma iterativa, utilizando-se o Método de Liebmann com sobre-relaxação, que pode ser conferido na Equação 16. Nela, o $V_{i,j}$ é o valor calculado pela expressão correspondente de cada caso e $V_{i,j}^{velho}$ o valor armazenado da iteração anterior. Na inicialização, todos os valores iniciam-se zerados. O cálculo do erro relativo e o critério de parada para um erro desejado $\varepsilon_{des}$ também podem ser conferidos na Equação 16.

$$V_{i,j}^{novo} = \lambda V_{i,j} + (1-\lambda)V_{i,j}^{velho} \qquad |(\varepsilon_a)_{max}| = \max_{\substack{1\leq j\leq m \\ 1\leq i\leq n}} \left| \frac{V_{i,j}^{novo} - V_{i,j}^{velho}}{V_{i,j}^{novo}} \right| < \varepsilon_{des} \tag{16}$$

Após encontrar a tensão elétrica de cada um dos pontos da malha, é possível então definir o valor da densidade de corrente $J$. Para isso definiu-se as variáveis $Q_r = \sigma\frac{\partial V}{\partial r}$ e $Q_\phi = \sigma\frac{1}{r_{ij}}\frac{\partial V}{\partial \phi}$, com $J = -(Q_r, Q_\phi)$.

Para $Q_r$, utilizou-se as aproximações das derivadas pela fórmula de diferença central de 2 pontos. Para os pontos em que não é possível utilizar a diferença central (condições **2, 3**, por se tratarem de regiões de fronteira de materiais, **4, 5** por se tratarem de bordas), utilizou-se as diferenças progressiva e regressiva de 3 pontos, respectivamente. As fórmulas podem ser conferidas a seguir:

$$Q_r = \begin{cases} \sigma\frac{-V_{i+2,j}+4V_{i+1,j}-3V_{i,j}}{2\Delta r} & \text{(progressiva)} \\ \sigma\frac{3V_{i,j}-4V_{i-1,j}+V_{i-2,j}}{2\Delta r} & \text{(regressiva)} \\ \sigma\frac{V_{i+1,j}-V_{i-1,j}}{2\Delta r} & \text{(central)} \end{cases} \tag{17}$$

Analogamente, para $Q_\phi$, adotou-se a diferença regressiva nas condições **7**, por se tratar de fronteira de material, e **0**, por se tratar da borda superior. Utilizou-se a

diferença central nos interiores dos materiais. Para a região de simetria da figura (condições **1, 6**), temos que $V_{i,j-1} = V_{i,j+1}$, dessa forma, $Q_\phi = 0$ na linha de simetria. As expressões da derivada regressiva e central são análogas às presentes na Equação 17, porém variando o índice $j$.

Por fim, com os valores de $Q_r$ e $Q_\phi$ é possível definir o vetor $J$ em cada um dos pontos da malha e, por meio da Equação 8, definir também os valores de $\dot{q}(i, j)$, que serão utilizados para avaliar o comportamento térmico da peça.

Com o valor da densidade de corrente $J$, temos o valor da corrente elétrica média que passa no sistema. A corrente é definida pela seguinte expressão:

$$I_m = \int_A J \cdot n dS \tag{18}$$

Temos, para o sistema, que a direção normal se trata da direção radial do sistema. Dessa forma, é possível, utilizando-se da Regra dos Trapézios para integração numérica, calcular a corrente elétrica pela seguinte expressão, considerando a espessura do material como unitária:

$$I_m = 2 \cdot r_{ij} \int J_r d\phi \Rightarrow I_m = 2 \cdot r_{ij} \sum_{i=1}^{m} \frac{(J_r^i + J_r^{i-1})\Delta\phi}{2} \tag{19}$$

Para encontrar o valor da resistência elétrica do bloco, é necessário duplicar o valor de $I_m$ encontrado, pois a integral só é realizada em metade da peça com relação a $\phi$. Temos também que $\Delta V = 100\ V$, permitindo o cálculo da resistência por $R = \frac{\Delta V}{2I_m}$.

### 2.2.2 Comportamento térmico

A equação que rege o comportamento térmico é a Equação 9. É possível observar a semelhança com a Equação 6, com o acréscimo do termo $\dot{q}$, calculado por meio da densidade de corrente. É necessário observar que o sinal de $\dot{q}$ encontrado nas equações elétricas é o inverso do sinal utilizado nas equações térmicas. Do ponto de vista elétrico, $\dot{q}$ é energia saindo do sistema por meio do Efeito Joule, enquanto, do ponto de vista térmico, $\dot{q}$ é calor sendo adicionado ao sistema.

É possível utilizar as mesmas 10 condições descritas pela Figura 3, e adaptar as equações encontradas para o caso térmico.

Para a **condição 8 - interior de A**, por exemplo, temos, utilizando as aproximações anteriores para as derivadas:

$$\frac{T_{i+1,j} - 2T_{i+1,j} + T_{i-1,j}}{\Delta r^2} + \frac{1}{r_{ij}}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{r_{i,j}^2}\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta\phi^2} + \frac{\dot{q_{ij}}}{k_A} = 0 \Rightarrow$$

$$T_{i,j} = \frac{T_{i+1,j}\left(\Delta r\Delta\phi^2 r_{ij} + 2\Delta\phi^2 r_{ij}^2\right) + 2T_{i,j+1}\Delta r^2 + 2T_{i,j-1}\Delta r^2 + T_{i-1,j}\left(-\Delta r\Delta\phi^2 r_{ij} + 2\Delta\phi^2 r_{ij}^2\right)}{4\left(\Delta r^2 + \Delta\phi^2 r_{ij}^2\right)} + \\ + \frac{2\Delta r^2\Delta\phi^2 r_{ij}^2 \dot{q_{ij}}}{4k_A\left(\Delta r^2 + \Delta\phi^2 r_{ij}^2\right)}$$

Nota-se que a expressão anterior é a Equação 15 acrescida de um fator com o $\dot{q}$. De forma análoga, para as condições (**0, 1, 6, 8, 9**), verificou-se que é possível utilizar as mesmas equações desenvolvidas para o caso elétrico, acrescentando-se o fator $\frac{2\Delta r^2\Delta\phi^2 r_{ij}^2 \dot{q_{ij}}}{k_i}$, onde $k_i$ é a condutividade térmica do material correspondente.

Para as condições de fronteira entre materiais (**2, 3, 7**), ao se adotar a condição de continuidade dos meios ${}_A\frac{\partial T}{\partial r}\Big|_{i,j}^{A} = k_B\frac{\partial T}{\partial r}\Big|_{i,j}^{B}$ para 2 e 3 e $k_A\frac{\partial T}{\partial\phi}\Big|_{i,j}^{A} = k_B\frac{\partial T}{\partial\phi}\Big|_{i,j}^{B}$ para 7, despreza-se a energia gerada nesta fronteira. Dessa forma, para estas três condições, as equações utilizadas são as mesmas desenvolvidas para a parte elétrica, ajustando-se as constantes de $\sigma_a$ e $\sigma_b$ para $k_A$ e $k_B$.

Para a condição **4**, borda esquerda de A, é necessário somente alterar o valor da condição $T_{i,j} = 30°C$.

Para a condição **5**, da borda direita de A, não há mais uma condição de contorno de *Dirichelet*, com um valor fixo, mas uma condição de *Von Neumann*, pois há o fenômeno de convecção na parede externa da peça. Temos, pela continuidade do fluxo térmico:

$$\dot{q}_{cond} = \dot{q}_{conv} \Rightarrow k_A\frac{\partial T}{\partial r}\Big|_{i,j} = h(T_{i,j} - T_{amb})$$

É possível, pela expansão por Taylor, encontrar também uma expressão para a segunda derivada de T:

$$\frac{\partial^2 T}{\partial r^2}\Big|_{i,j} = \frac{2}{\Delta r^2}\left(-T_{i,j} + T_{i-1,j} - \frac{\Delta r h\left(-T_{amb} + T_{i,j}\right)}{k_A}\right)$$

Com as expressões anteriores e usando a fórmula da segunda diferença central para a coordenada $\phi$, chegamos na seguinte expressão:

$$T_{i,j} = \frac{T_{amb}\left(\Delta r^2\Delta\phi^2 h r_{ij} + 2\Delta r\Delta\phi^2 h r_{ij}^2\right) + T_{i,j+1}\Delta r^2 k_A + T_{i,j-1}\Delta r^2 k_A}{\Delta r^2\Delta\phi^2 h r_{ij} + 2\Delta r^2 k_A + 2\Delta r\Delta\phi^2 h r_{ij}^2 + 2\Delta\phi^2 k_A r_{ij}^2} + \\ + \frac{2T_{i-1,j}\Delta\phi^2 k_A r_{ij}^2 + \Delta r^2\Delta\phi^2 \dot{q} r_{ij}^2}{\Delta r^2\Delta\phi^2 h r_{ij} + 2\Delta r^2 k_A + 2\Delta r\Delta\phi^2 h r_{ij}^2 + 2\Delta\phi^2 k_A r_{ij}^2}$$

Com as expressões da temperatura e por meio do método de Liebmann, é possível, de forma análoga à parte elérica, encontrar os valores das temperaturas na malha. Com os valores de temperatura é possível então encontrar o vetor fluxo de calor no interior da peça.

O fluxo de calor é definido de forma análoga ao vetor densidade de corrente elétrica e sua fórmula pode ser conferida a seguir:

$$Q = -k\nabla T = -k\left(\frac{\partial T}{\partial r}, \frac{1}{r_{ij}}\frac{\partial T}{\partial \phi}\right) \tag{20}$$

Pela semelhança com a Equação 7, podemos definir $Q_r = k\frac{\partial T}{\partial r}$ e $Q_\phi = k\frac{1}{r_{ij}}\frac{\partial T}{\partial \phi}$, com $Q = -(Q_r, Q_\phi)$. O cálculo de $Q_r$ e $Q_\phi$ é feito com as mesmas expressões e condições de suas variáveis equivalentes para o caso elétrico.

Por fim, é necessário calcular a quantidade de calor que flui pela parede de convecção (borda direita de A). A quantidade de calor é calculada por:

$$q_{conv} = \int_A Q \cdot n dS \tag{21}$$

Por sua semelhança com a Equação 19, é possível usar o mesmo método para calcular a integral:

$$q_{conv} = 2 \cdot r_{ij}\int_A Q_r d\phi \Rightarrow q_{conv} = 2 \cdot r_{ij}\sum_{i=1}^{m}\frac{(Q_r^i + Q_r^{i-1})\Delta\phi}{2} \tag{22}$$

## 2.3 Resultados e Discussões

Após a realização das simulações e análise dos valores gerados, diversos resultados e conclusões foram passíveis de serem realizadas, o que nos permite responder as questões elaboradas. Contudo, nota-se que, dado a escala de grandeza dos valores de condutividade elétrica do material, alguns dos resultados não representam a situação real de um

forno industrial. Vendo assim, uma necessidade de alterar os valores de condutividade para aproximar a casos mais próximos do real, em especial na elaboração dos resultados térmicos. Tendo em vista os valores em $\sigma_A$ e $\sigma_B$ alteram consideravelmente a magnitude da potência elétrica dissipada por efeito Joule ($\dot{q}$).

Tendo isso em consideração, se atualizarmos o valor da condutividade elétrica para valores maiores $\sigma_{Anovo} = \sigma_A \cdot 10^6$ e $\sigma_{Bnovo} = \sigma_B \cdot 10^6$ , teremos gráficos e resultados com valores mais expressivos e lógicos. Portanto, foi-se utilizado esses valores para os demais resultados das simulações.

### 2.3.1 Definição para o tamanho de malha

Com a realização de diferentes testes, foi-se experimentado amplamente os resultados dos experimentos para 3 valores diferentes de malhas, interpretadas como pequena, média e grande:

| **Tipo de Malha** | **Valor de $d_r$** | **Valor de $d_\phi$** | **Tamanho da malha** |
|---|---|---|---|
| Pequena | 0.005 | 2 graus | (17,21) |
| Média | 0.001 | 1 grau | (81,41) |
| Grande | 0.0005 | 0.5 grau | (161,81) |

Para a malha pequena, apesar de sua alta velocidade de execução, seus resultados em relação aos valores vetoriais (densidade de corrente $J(r, \phi)$ e fluxo de calor $Q(r, \phi)$) mostram-se extremamente deformados em relação a direção em decorrência do baixo número de iterações. Ademais, com o tamanho reduzido da malha, os valores da potência dissipada não se diferenciam tão expressivamente com a mudança de material. Portanto, esse valor de malha foi desconsiderado.

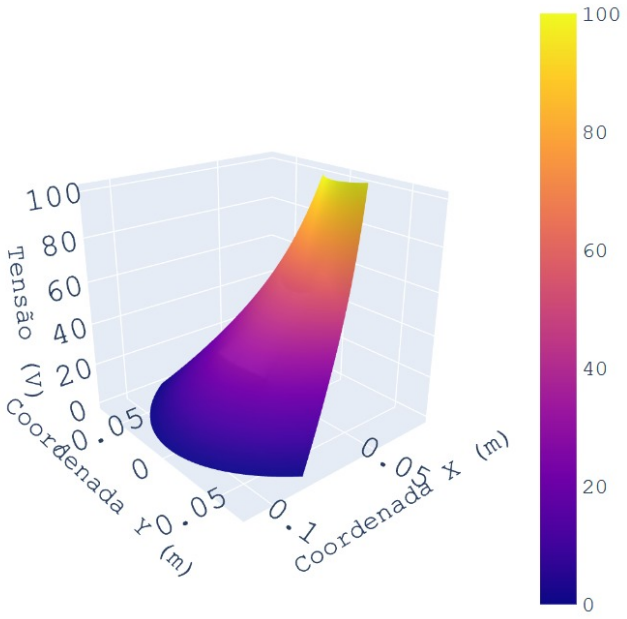
Para a malha grande, a velocidade de execução do código em comparação com os testes da malha média e pequena foram muito maiores em quase 2 ordens de grandeza. Contudo, na análise dos resultados (discutidos a seguir), suas simulações foram extremamente parecidas com o caso médio, o que levou o caso da malha grande a ser descartado.

Por fim, foi possível escolher a malha média, uma vez que, ao apresentar valores intermediários, consegue ser executada com velocidades próximas a malha pequena, mas apresentando resultados extremamente similares com a malha grande, justificando a escolha.

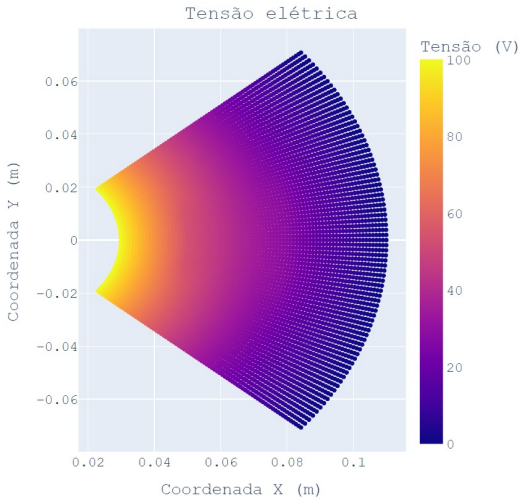
### 2.3.2 Resultados do comportamento elétrico

#### 2.3.2.1 Distribuição do Potencial Elétrico $V(r, \phi)$ e impacto da discretização

Partindo da matriz nula (com valores iguais a zero) e aplicando o **Método de Liebman** com as condições de contorno desenvolvidas em 2.2.1, foi possível obter a seguinte distribuição geral da tensão $V(r, \phi)$ :




(a) Gráfico 3D do Potencial Elétrico $V(r, \phi)$




(b) Distribuição do Potencial Elétrico $V(r, \phi)$

Com base na Figura 6a, é possível notar que o valores encontrados seguem as condições de Dirichlet impostas no equacionamento (partindo de uma tensão $V = 100\,V$ para valores de $r = 0.03m$, até uma tensão de $V = 0\ V$ em valores $r = 0.11m$).

Ademais, há de se perceber uma leve inclinação no início do material B, evidenciada na Figura 7, quando comparado com o material A. Isso pode ser explicado pela diferença da condutividade entre eles: como o material B apresenta uma condutividade maior e, portanto, uma menor resistividade à tensão elétrica, a queda de tensão torna-se menos intensa no interior desse material, comprovando a teoria do sistema elétrico por meio da simulação prática.

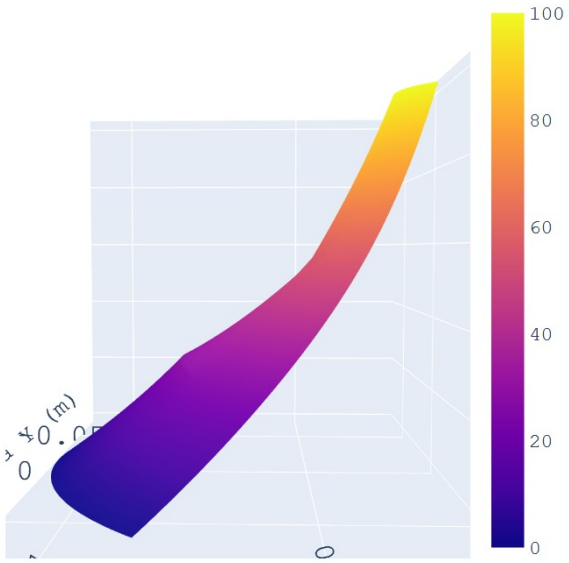



Figura 7: Gráfico do Potencial Elétrico $(Vr, \phi)$ evidenciando a diferença de queda de tensão entre materiais

#### 2.3.2.2 Distribuição da Potência Elétrica Dissipada

Com a obtenção dos valores do potencial em todo o espaço, foi possível estimar os valores da densidade de corrente elétrica e de potência dissipada utilizando novamente o Método das diferenças finitas utilizando condições vistas em 2.2.1.Tendo isso em consideração, foi possível produzir as distribuições da potência dissipada, vista na Figura 8:

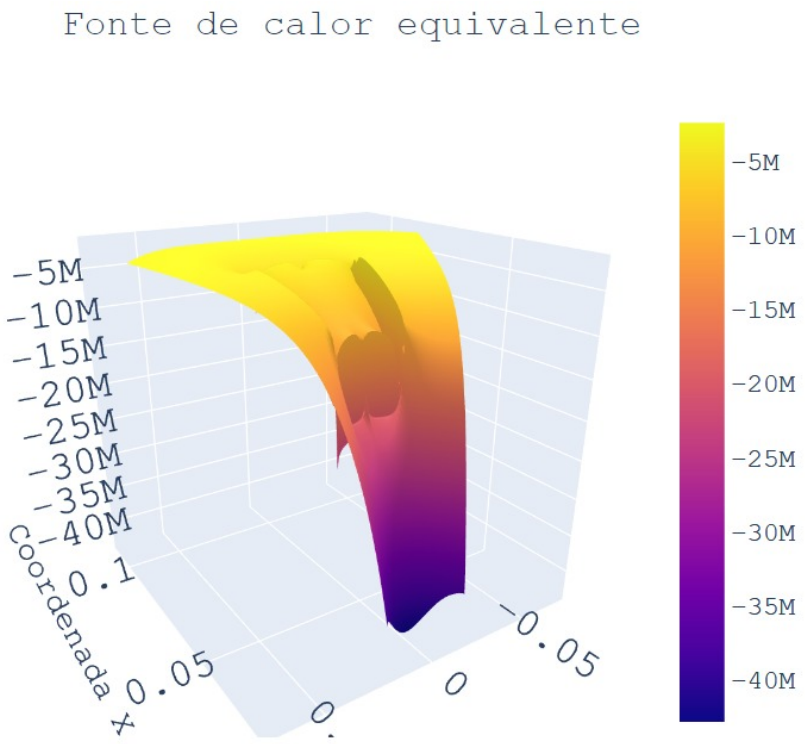



Figura 8: Gráfico da Potência Dissipada $\dot{q}(r, \phi)$ $[W/m^3]$)

Ao analisar o gráfico, é possível notar que sua forma faz sentido físico, tendo em vista que o material B, por apresentar maior condutibilidade elétrica, acaba por dissipar mais energia, o que pode ser verificado no gráfico. Percebe-se que os valores de potência dissipada são maiores em ponto com maior tensão (e consequentemente corrente).

Ademais, deve-se perceber que, para o gráfico, o valor de potência dissipada é evidenciado com valor negativo, uma vez que a energia está saindo do sistema elétrico. Contudo, quando utilizarmos a distribuição para gerar os valores de temperatura, esse valor deve ser positivo no sentido de adicionar energia no sistema térmico.

#### 2.3.2.3 Densidade de Corrente Elétrica

Assim como a potência dissipada, o vetor de Densidade de Corrente Elétrica para cada ponto foi elaborado pela utilização do MDF com as condições da primeira derivada, realizando o processo separadamente para a componente em $r : Q_r$ e $\phi : Q_\phi$. Após o fim das iterações sobre a matriz, tivemos o resultado dado pelo Quiver plot da figura 9.

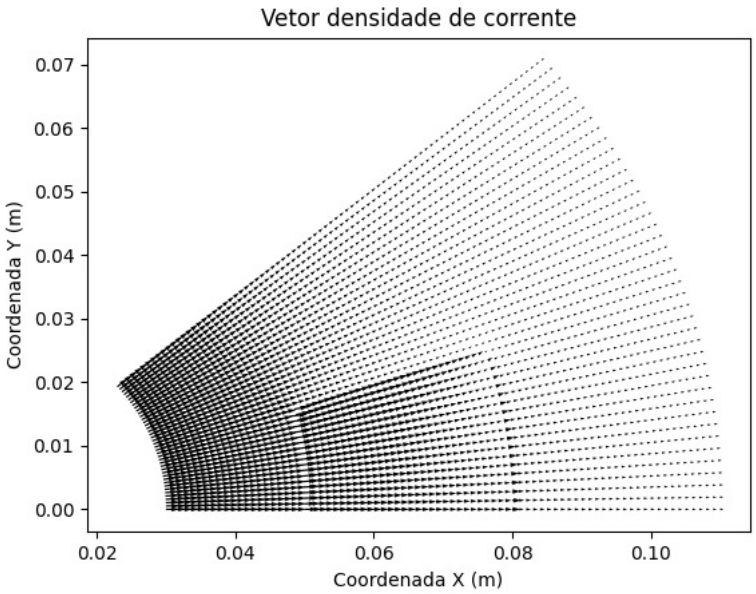



Figura 9: Quiver plot do vetor densidade de corrente elétrica $J(r, \phi)$ em $A/m^2$

Com base no gráfico, é possível perceber que a densidade de corrente flui da região de maior potencial para a região de menor potencial (fisicamente acurado). Outrossim é possível notar o maior módulo dos vetores na região de B quando comparado com os vetores em A com o mesmo valor de raio, processo que também justifica o valor maior na potência dissipada, comentada anteriormente.

Por fim, é também possível perceber que a componente no sentido de $Q_r$ é consideravelmente maior do que a componente $Q_\phi$ no sentido angular, tendo em vista que as condições que realizam o movimento de corrente (diferença de potencial), estão distribuídas uniformemente em relação aos ângulos ($V = 100$ para todo $\phi$ no início do fogão, e igual à 0V no contorno).

#### 2.3.2.4 Estimativa do valor médio de corrente e resistência média

Após a conclusão da estimativa de $J$, foi possível a construção de um método para o cálculo da integral que obtém o valor da corrente média $I_m$. Como dito na secção 2.2.1, o cálculo da integral pela estimação da soma dos trapézios (já duplicada) foi de:

$$I_m(r_{max}) = 1189.2595A \tag{23}$$

$$I_m(r_{min}) = 1152.6803A \tag{24}$$

Como o cálculo da integral poderia ser realizado para $r = 0.03m$ e $r = 0.11m$, foi-se calculado ambas e o resultado final considerado foi a média das duas. Para os valores do enunciado original, portanto:

$$I_m = 1170.9698A \tag{25}$$

Com o valor de $I_m$, podemos, por extensão, encontrar o valor da resistência equivalente:

$$R_{eq} = \frac{\Delta V}{I_m} = \frac{100}{1170.9698} = 0.08539\Omega \tag{26}$$

### 2.3.3 Resultados do comportamento térmico

#### 2.3.3.1 Distribuição da Temperatura no forno $T(r, \phi)$

Com a distribuição estimada da potência dissipada pelo sistema elétrico, além das condições de contorno das temperaturas e da convecção, é possível, na mesma função utilizada para o caso elétrico (visto em:5), definir a distribuição da temperatura no material, que pode ser conferida na Figura 10.

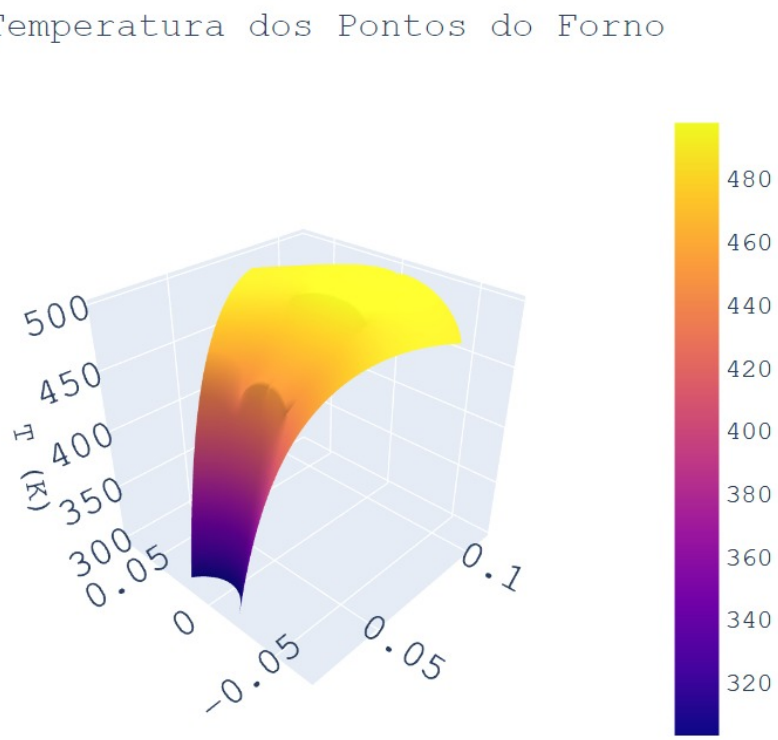



Figura 10: Plot da distribuição da temperatura $T(r, \phi)$ em $K$

Primeiramente, é possível notar, na Figura 11, o impacto claro de $\dot{q}$ na temperatura pela semelhança clara do formato das distribuições.

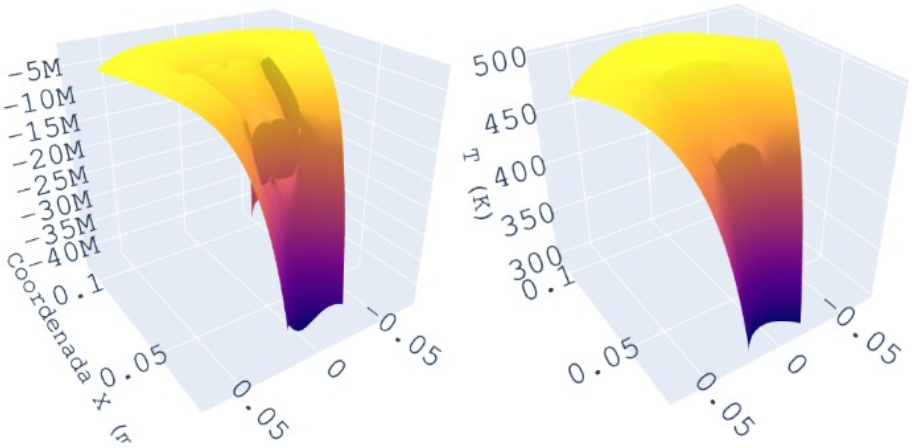



Figura 11: Comparação entre $T(r, \phi)$ e $\dot{q}(r, \phi)$

Ademais, é possível perceber que a distribuição de temperaturas simulada apresenta pleno sentido físico, uma vez que respeita a condição de contorno em $r = 0.03m$ e

demonstra um crescimento claro com o aumento da potência dissipada até o limite $r_{max} = 0.11$.

Por fim, vê-se o comportamento coerente no material B, apresentando uma temperatura maior do que os pontos no material A de mesmo raio, tendo em vista na região B apresenta maior potência dissipada, como explicado na secção anterior (2.3.2.2).

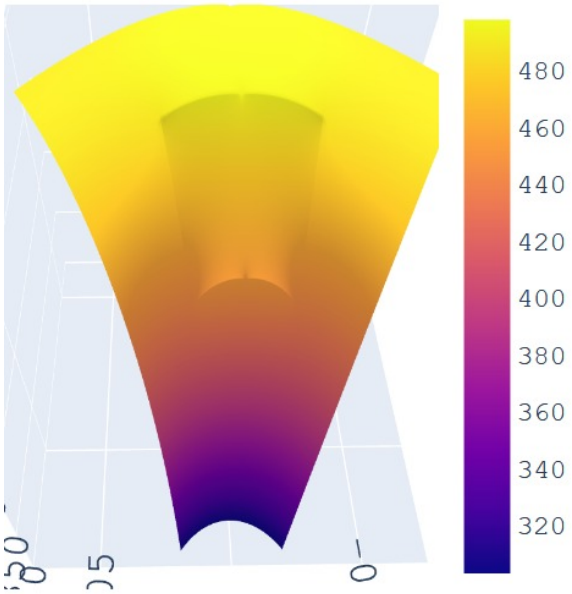



Figura 12: Diferença entre materiais e impacto na temperatura

#### 2.3.3.2 Distribuição dos vetores de fluxo de calor

Com os resultados de $T(r, \phi)$, é possível, utilizando a mesma função desenvolvida na definição da Densidade de Corrente a partir de $V(r, \phi)$, determinar a distribuição vetorial do fluxo de calor, bastando apenas alterar o fator multiplicativo das derivadas de $\sigma$ para $k$.

Assim, pode-se obter os seguintes resultados do *quiver plot* na Figura 13:

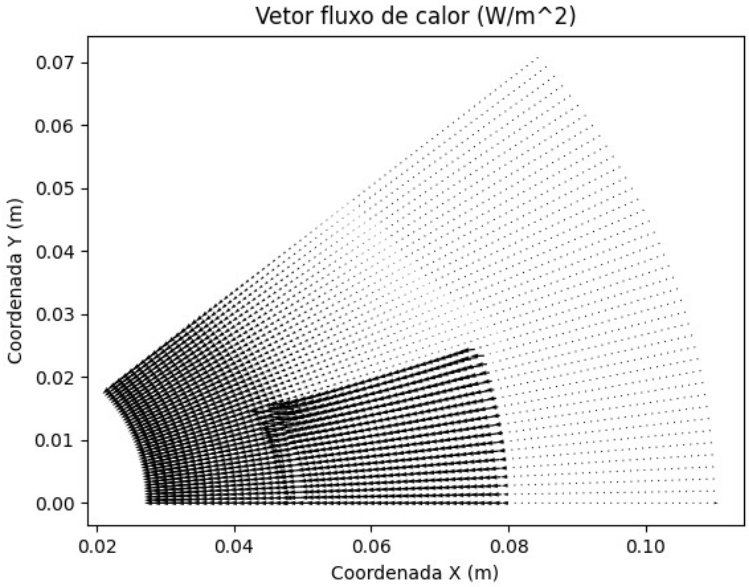



Figura 13: Gráfico vetorial do fluxo de calor $Q(r, \phi)$

Ao analisar o resultado simulado, foi possível analisar que o fluxo de calor apresenta um módulo grande (que precisou ser normalizado no gráfico quiver para ter mais legibilidade). Além disso, vemos que a direção do fluxo térmico apresenta sentido físico correto, tendo em vista que, no material B, seu módulo é superior para o mesmo raio no material A, o que pode ser explicado pelas magnitudes das condutividades térmicas, pois $k_A < k_B$, implicando em maiores valores de transferência de energia térmica.

#### 2.3.3.3 Quantidade de calor que flui pela parede de convecção

Por fim, com o valor de fluxo térmico determinado, a definição da quantidade de calor que flui pela parede de convecção torna-se um problema análogo a definição de corrente média por meio da densidade de corrente elétrica. Isso ocorre uma vez que as integrais são aplicadas da mesma forma e pode ser desenvolvidas para a mesma superfície, permitindo assim, que seja possível realizar o cálculo com a mesma função programada anteriormente, apenas alterando os fatores multiplicativos de $\sigma_A$ para $k_A$.

Com isso, obtemos os resultados, para $k_A = 100$ e $k_B = 500$:

$$q_{conv} = 7559.571W \tag{27}$$

# 3 Conclusões

Por meio deste exercício-programa, foi possível aplicar os conceitos de métodos numéricos aplicados à engenharia para a resolução de problemas com equações diferenciais e de equações diferenciais parciais.

Na primeira parte do problema, foi possível aplicar o método de Runge-Kutta para solucionar equações diferenciais que ditam o comportamento elétrico de um circuito. Nota-se a facilidade e a versatilidade do método, capaz de resolver diversas equações diferenciais que muitas vezes não possuem soluções analíticas de uma forma prática e eficiente. Ao se utilizar o método, é possível alterar propriedades físicas do sistema e rapidamente obter o novo comportamento, característica desejável em projetos de engenharia.

Vale-se destacar a importância do passo utilizado no algoritmo, pois sua escolha influencia diretamente na resolução obtida. Um passo muito grande não é capaz de gerar soluções coerentes e muitas vezes pode divergir. Por outro lado, um passo muito pequeno gera um algoritmo de execução lenta e cujo aumento de detalhe na resolução não seja relevante para a análise. É necessário então assumir uma solução de compromisso, com um passo capaz de fornecer os dados esperados, em um tempo de execução razoável.

Na segunda parte do problema, foi possível implementar um sistema complexo, regido por uma equação diferencial parcial em uma malha bidimensional, de forma numérica com o auxílio do Método de Elementos Finitos. Este exercício possibilitou a avaliação do comportamento da solução numérica sob o efeito de diversos tamanhos de malha e discretização, mostrando que, assim como para o passo no algoritmo de Runge Kutta, é necessário assumir uma solução de compromisso.

O uso das técnicas de MDF se mostrou versátil para a resolução do problema, pois, uma vez que as variáveis primárias são modeladas e calculadas, o cálculo das variáveis secundárias é simples e direto. Dessa forma, o método permite que se obtenha diversas informações relevantes sobre um sistema complexo e de difícil solução analítica, e, por se tratar de uma tarefa computacional, permite a análise da influência dos parâmetros de forma fácil e prática. A simulação do forno também permitiu visualizar e entender como as características elétricas e térmicas de diferentes materiais podem ser utilizadas de modo a causar diferentes efeitos para cumprir um objetivo requisitado no projeto.

Outrossim, até mesmo os erros encontrados e solucionados durante a criação dos

programas agregaram grande aprendizado para a formação na engenharia. Evidenciando, por exemplo, a propagação de erros nos equacionamentos, ao serem somados em grandes malhas, geram resultados finais muito destoantes da realidade.

# 4 Listagem de códigos

## 4.1 Parte 1 - Método de Runge Kutta

### 4.1.1 Implementação do RK4

Listing 1: Implementação do RK4

```
import numpy as np
import matplotlib.pyplot as plt

class RK4():
    '''
    Implementao do RK4 clssico
    '''
    def __init__(self: object, func: np.array, step: float, x0: float, y0: np.array)
        :
        self.func = func #espera funcao de x e y
        self.step = step #assume step fixo
        self.x0 = x0 #valor da variavel independente
        self.y0 = y0 #vetor da variavel dependente



    def _compute_step(self, x, y):

        #calcula constantes seguindo RK4 classico
        k1 = self.func(x, y)

        k2 = self.func(x + self.step/2, y + self.step/2 * k1)

        k3 = self.func(x + self.step/2, y + self.step/2 * k2)

        k4 = self.func(x + self.step, y + self.step * k3)

        return y + self.step/6 * (k1 + 2 * k2 + 2 * k3 + k4)

    def solve(self, n_steps):

        #formato do vetor: vec_y[instante, variavel]
        vec_y = np.zeros(shape=(n_steps, self.y0.shape[0]))
```

```
32          vec_y[0, :] = self.y0
33
34          vec_x = np.zeros(shape=(n_steps,))
35          vec_x[0] = self.x0
36
37          for i in range(0, n_steps-1, 1):
38
39              vec_y[i + 1, :] = self._compute_step(vec_x[i], vec_y[i, :])
40              vec_x[i + 1] = vec_x[i] + self.step
41
42          return vec_x, vec_y
43
44
45
46  if __name__ == '__main__':
47
48      def f(x, y):
49          return np.array([
50              -0.5*y[0],
51              4 - 0.3*y[1] - 0.1*y[0]
52          ])
53
54      x0 = 0
55      y0 = np.array([4, 6])
56      h = 0.5
57
58      runge = RK4(f, h, x0, y0)
59      x, y = runge.solve(3)
60
61      plt.plot(x, y[:, 0])
62      plt.plot(x, y[:, 1])
63      plt.show()
```

### 4.1.2 Implementação da solução do circuito

Listing 2: Implementação da solução do circuito

```
1  import numpy as np
2  from runge_kutta import RK4
3  import matplotlib.pyplot as plt
```

```
4
5  # variaveis globais
6  Lb = 0.5
7  Rb = 20
8  C = 0.002
9  La = 0.01
10 Ra = 200
11
12
13 def e(t):
14     #funcao da corrente eletrica
15     return np.cos(t * 600) / La
16
17
18 def f(t, Y):
19     '''
20     O vetor de variaveis dependentes  dado por:
21
22             | ------ |
23             | q(t) |
24     Y(t) = | i_1(t) |
25             | i_2(t) |
26             | ------ |
27     '''
28
29     return np.array([
30         Y[1] - Y[2],
31         1 / La * (e(t) - Ra * (Y[1] - Y[2]) - Y[0] / C),
32         1 / Lb * (Ra * (Y[1] - Y[2]) - Rb * Y[2] + Y[0] / C)
33     ])
34
35
36 if __name__ == '__main__':
37     delta_t = 0.0001
38     t0 = 0
39
40     tmax = 0.1
41
42     n_steps = int(tmax / delta_t) #define numero de steps de integracao
43
44     Y0 = np.array([
```

```
45          0,
46          0,
47          0
48      ])
49
50      edo = RK4(func=f, step=delta_t, x0=t0, y0=Y0)
51
52      #resolve sistema por RK4 com o numero de steps definido
53      t_int, y_int = edo.solve(n_steps)
54
55      #calcula a derivada de [Y] por meio de [F]
56      ydot_int = np.array([f(t_int[i], y_int[i, :]) for i in range(len(t_int))])
57
58      #define constantes para multiplicar cada uma das respostas
59      plot_scalars = {
60          'q': 1e5,
61          'i_1': 1e2,
62          'i_2': 1e2,
63          '\dot{i_1}': 0.1,
64          '\dot{i_2}': 0.1,
65      }
66
67      #plot dos valores
68
69      plt.plot(t_int, y_int[:, 0] * plot_scalars['q'])
70      plt.plot(t_int, y_int[:, 1] * plot_scalars['i_1'])
71      plt.plot(t_int, y_int[:, 2] * plot_scalars['i_2'])
72
73      #plot das derivadas
74      plt.plot(t_int, ydot_int[:, 1] * plot_scalars['\dot{i_1}'])
75      plt.plot(t_int, ydot_int[:, 2] * plot_scalars['\dot{i_1}'])
76
77      plt.legend([f'${key} \cdot {value:.2e}$' for key, value in plot_scalars.items()
            ])
78
79      top_limit = 150
80      plt.ylim(top=top_limit, bottom=-top_limit)
81
82      plt.title(f'Integrao de {t0} a {tmax} com passo {delta_t:.2e}')
83      plt.grid(True)
84
```

```
85      plt.show()
```

## 4.2 Parte 2 - Método de Diferenças Finitas

### 4.2.1 Implementação do método de Liebman

Listing 3: Implementação Liebman

```
1  import numpy as np
2
3  def calcula_erro(M_novo, M_atual):
4      '''
5      Funo que calcula o erro relativo entre a iterao atual e a iterao
6      anterior, devolvendo uma matriz com o valor do erro em cada entrada
7      da matriz M
8
9      **Entradas**:
10     M_novo: matriz da nova iterao
11     M_atual: matriz da iterao anterior
12
13     **Sadas**:
14     erro: matriz com o erro relativo de cada um dos valores da varivel
15
16     '''
17     erro = np.zeros(M_atual.shape)
18
19     for i in range(M_atual.shape[0]):
20         for j in range(M_atual.shape[1]):
21             if(M_novo[i, j] == 0 and M_atual[i, j] == 0):
22                 #evita a diviso por zero
23                 erro[i, j] = 0
24             elif(M_novo[i, j] == 0 and M_atual[i, j] != 0):
25                  #no deve ocorrer, caso ocorra o cdigo para imediatamente
26                 import pdb; pdb.set_trace()
27             else:
28                 erro[i, j] = np.abs((M_novo[i, j] - M_atual[i, j])/M_novo[i, j])
29     return erro
30
31 def calcula_iteracao(M_atual, func, lamb, dr, dtheta, q_dots):
32     '''
```

```
33      Funo que calcula uma iterao do mtodo de liebmann, j aplicando a
34      sobrerelaxao a cada item calculado
35
36      **Entradas**:
37      M_atual: numpy array com os valores iniciais da iterao, permanece inalterada
38      func: funo que calcula novo valor da varivel
39      lamb: fator de sobrerelaxao
40      dr: variao "delta r"
41      dtheta: variao "delta theta"
42      q_dots: valor de q_ponto, usado no caso trmico
43
44      **Sadas**
45      M_step: numpy array com os novos valores aps o clculo da iterao
46
47      '''
48      M_step = np.copy(M_atual)
49
50      for i in range(M_atual.shape[0]):
51          for j in range(M_atual.shape[1]):
52              M_step[i, j] = func(M_step, i, j, dr, dtheta, q_dots)
53              M_step[i, j] = lamb*M_step[i, j] + (1 - lamb)*M_atual[i, j]
54
55      return M_step
56
57  def liebmann(M, func, lamb, erro_des, dr, dtheta, max_steps=1e5,q_dots=None):
58      '''
59      Implementao do mtodo de Liebmann com sobrerelexao
60
61      **Entradas**:
62      M: numpy array que armazenar os valores da varavel calculada
63      func: funo que calcula novo valor da varivel
64      lamb: fator de sobrerelaxao
65      erro_des: erro relativo desejado
66      dr: variao "delta r"
67      dtheta: variao "delta theta"
68      max_steps: nmero mximo de iteraes, interrompe execuo caso
69      a convergncia no tenha ocorrido
70      q_dots: valor de q_ponto, usado no caso trmico
71
72      **Sadas**
73      M_atual: Matriz com os valores da varivel calculada aps a convergncia
```

```
    '''

    M_atual = np.copy(M)
    erro = [10*erro_des]
    i = 1

    while(np.max(erro) > erro_des):
        M_novo = calcula_iteracao(M_atual, func, lamb, dr, dtheta,q_dots)
        erro = calcula_erro(M_novo, M_atual)
        M_atual = np.copy(M_novo)
        i += 1

        print(f"Iterao {i}: erro {np.max(erro):.5f}", end='\r')

        if(i > max_steps):
            break

    print(f"Erro {np.max(erro):.4f} atingido com {i} steps")
    return M_atual
```

### 4.2.2 Implementação das equações pelo Método de Diferenças Finitas

Listing 4: Implementação MDF com condições do problema

```
import numpy as np

#propriedades fisicas relevantes
props_elet = {
    "sigma_A": 5e-6*1e6,
    "sigma_B": 1e-5*1e6,
    "k_A": 110,
    "k_B": 500,
    "h": 50
}

"""
Todas as funcoes precisam das temperaturas da matriz
de temperatura e da coordenada atual, alem das prop
fisicas e eletricas. Para calcular os valores da tenso
eltrica, o q_dot fornecido deve ser nulo, caso contrrio,
```

```
ser calculado o valor da temperatura no ponto

de forma geral:
**Entradas**
M: matriz dos valores a serem calculados
i e j: coordenadas do ponto desejado
    i: coordenada do raio
    j: coordenada do ngulo
dr: variao do raio "delta r"
dtheta: variao do ngulo"delta theta"
qdot: valor do q ponto, usado no caso trmico

**Saida**
valor da tenso eltrica ou da temperatura no ponto

As funes abaixo seguem a nomenclatura criada
para este problema, com as seguintes condies:
- 0: borda superior de A
- 1: borda inferior de B (regiao de simetria)
- 2: borda esquerda de B
- 3: borda direita de B
- 4: borda esquerda de A
- 5: borda direita de A
- 6: borda inferior de A (regiao de simetria)
- 7: borda superior de B
- 8: interior de A
- 9: interior de B
"""

#0: borda superior de A
def sup_A(M, i, j, dr, dtheta, qdot):
    raio = 0.03 + i*dr
    angulo = j*dtheta #em radianos
    k_A = props_elet['k_A']

    coefs = np.array([
        4*(dr**2 + dtheta**2 * raio**2),
        -dr* dtheta**2 * raio + 2 * dtheta**2 * raio**2,
        dr* dtheta**2 * raio + 2 * dtheta**2 * raio**2,
        4 * dr**2,
    ])
```

```
58
59      pontos = np.array([M[i-1,j], M[i+1,j], M[i,j-1]]).reshape(3,1)
60
61      return (np.float(coefs[1:] @ pontos)-(qdot/k_A)*2*(dr**2 * dtheta**2 * raio**2))
            /np.float(coefs[0])
62
63  #1: borda inferior de B (regiao de simetria)
64  def inf_B(M, i, j, dr, dtheta,qdot):
65      raio = 0.03 + i*dr
66      k_B = props_elet['k_B']
67
68      sigma = props_elet['sigma_B']
69
70      coefs = np.array([
71          4*(dr**2 + dtheta**2 * raio**2),
72          -dr * dtheta**2 * raio + 2 * dtheta**2 * raio**2,
73          dr * dtheta**2 * raio + 2 * dtheta**2 * raio**2,
74          2*dr**2,
75          2*dr**2,
76      ])
77
78      pontos = np.array([M[i-1,j], M[i+1,j], M[i,j+1], M[i,j+1]]).reshape(4,1)
79
80      return (np.float(coefs[1:] @ pontos)-(qdot/k_B)*4*(dr**2 * dtheta**2 * raio**2))
            /np.float(coefs[0])
81
82  #2: borda esquerda de B
83  def esq_B(M, i, j, dr, dtheta,qdot):
84      raio = 0.03 + i*dr
85
86      k_A = props_elet['k_A']
87      k_B = props_elet['k_B']
88
89      if qdot==0:
90          sigmaA = props_elet['sigma_A']
91          sigmaB = props_elet['sigma_B']
92
93          alpha = sigmaA * (-2/dr + 1/raio)
94          beta = sigmaB * (2/dr + 1/raio)
95
96      else:
```

```
97
98          alpha = k_A * (-2/dr + 1/raio)
99          beta = k_B * (2/dr + 1/raio)
100
101
102     coefs = np.array([
103         2*(alpha - beta)*(raio**2 * dtheta**2 + dr **2),
104         2 * raio**2 * dtheta**2 * alpha,
105         -2 * raio**2 * dtheta**2 * beta,
106         (alpha - beta)*dr**2,
107         (alpha - beta)*dr**2,
108     ])
109
110     pontos = np.array([M[i-1,j], M[i+1,j], M[i,j-1], M[i,j+1]]).reshape(4,1)
111
112     return (np.float(coefs[1:] @ pontos)/np.float(coefs[0]))
113
114 #3: borda direita de B
115 def dir_B(M, i, j, dr, dtheta,qdot):
116     raio = 0.03 + i*dr
117     k_A = props_elet['k_A']
118     k_B = props_elet['k_B']
119
120     if qdot==0:
121         sigmaA = props_elet['sigma_A']
122         sigmaB = props_elet['sigma_B']
123
124         alpha = sigmaB*(-2/dr + 1/raio)
125         beta = sigmaA*(2/dr + 1/raio)
126
127     else:
128         # k_A = props_elet['k_A']
129         # k_B = props_elet['k_B']
130
131         alpha = k_A*(-2/dr + 1/raio)
132         beta = k_B*(2/dr + 1/raio)
133
134
135     coefs = np.array([
136         2*(alpha - beta)*(raio**2 * dtheta**2 + dr **2),
137         2 * raio**2 * dtheta**2 * alpha,
```

```
        -2 * raio**2 * dtheta**2 * beta,
        (alpha - beta)*dr**2,
        (alpha - beta)*dr**2,
    ])

    pontos = np.array([M[i-1,j], M[i+1,j], M[i,j-1], M[i,j+1]]).reshape(4,1)

    #VARZEA
    return np.float(coefs[1:] @ pontos)/np.float(coefs[0])

#4: borda esquerda de A
def esq_A(M, i, j, dr, dtheta,qdot):
    if qdot==0:
        return 100
    else:
        return 30 + 273


#5: borda direita de A
def dir_A(M, i, j, dr, dtheta,qdot):
    if qdot==0:
        return 0
    else:
        T_amb = 25
        k_A = props_elet['k_A']
        h = props_elet['h']
        raio = 0.03 + i*dr

        coefs = np.array([
            (dr**2 * dtheta**2 * h*raio) + 2* dr**2 * k_A + 2* dr* dtheta**2 * h *
                raio**2
            + 2* dtheta**2 * k_A * raio**2,
            2* dtheta**2 * k_A * raio**2,
            dr**2 * k_A,
            dr**2 * k_A,
        ])

        try:
            pontos = np.array([M[i-1,j], M[i,j-1], M[i,j+1]]).reshape(3,1)

            temp = np.float(coefs[1:] @ pontos)
```

```
178
179             temp += T_amb*(dr**2 * dtheta**2 * h * raio + 2* dr * dtheta**2 * h *
                    raio**2)
180             temp += qdot * dr**2 * dtheta**2 * raio**2
181
182         except:
183             return M[i,j-1]
184
185         return temp/np.float(coefs[0])
186
187
188 #6: borda inferior de A (regiao de simetria)
189 def inf_A(M, i, j, dr, dtheta,qdot):
190     raio = 0.03 + i*dr
191     k_A = props_elet['k_A']
192
193     coefs = np.array([
194         4*(dr**2 + dtheta**2 * raio**2),
195         -dr * dtheta**2 * raio + 2 * dtheta**2 * raio**2,
196         dr * dtheta**2 * raio + 2 * dtheta**2 * raio**2,
197         2*dr**2,
198         2*dr**2,
199     ])
200     pontos = np.array([M[i-1,j], M[i+1,j], M[i,j+1], M[i,j+1]]).reshape(4,1)
201
202     return (np.float(coefs[1:] @ pontos)-(qdot/k_A)*2*(dr**2 * dtheta**2 * raio**2))
            /np.float(coefs[0])
203
204 #7: borda superior de B
205 def sup_B(M, i, j, dr, dtheta,qdot):
206     raio = 0.03 + i*dr
207
208     if qdot==0:
209         A = props_elet['sigma_A']
210         B = props_elet['sigma_B']
211     else:
212         A = props_elet['k_A']
213         B = props_elet['k_B']
214
215     coefs = np.array([
216         4*(A + B)*(dtheta**2 * raio**2 + dr**2),
```

```
        (A + B)*(-dtheta**2 * dr * raio + 2 * dtheta**2 * raio**2),
        (A + B)*(dtheta**2 * dr * raio + 2 * dtheta**2 * raio**2),
        4 * dr**2 * A,
        4 * dr**2 * B,
    ])

    pontos = np.array([M[i-1,j], M[i+1,j], M[i,j-1], M[i,j+1]]).reshape(4,1)


    return np.float(coefs[1:] @ pontos)/np.float(coefs[0])

#8: interior de A
def inter_A(M, i, j, dr, dtheta,qdot):
    raio = 0.03 + i*dr
    k_A = props_elet['k_A']

    coefs = np.array([
        4*(dr**2 + dtheta**2 * raio**2),
        -dr * dtheta**2 * raio + 2 * dtheta**2 * raio**2,
        dr * dtheta**2 * raio + 2 * dtheta**2 * raio**2,
        2*dr**2,
        2*dr**2,
    ])

    pontos = np.array([M[i-1,j], M[i+1,j], M[i,j-1], M[i,j+1]]).reshape(4,1)

    return (np.float(coefs[1:] @ pontos)-(qdot/k_A)*2*(dr**2 * dtheta**2 * raio**2))
        /np.float(coefs[0])


#9: interior de B
def inter_B(M, i, j, dr, dtheta,qdot):
    raio = 0.03 + i*dr
    k_B = props_elet['k_B']

    coefs = np.array([
        4*(dr**2 + dtheta**2 * raio**2),
        -dr * dtheta**2 * raio + 2 * dtheta**2 * raio**2,
        dr * dtheta**2 * raio + 2 * dtheta**2 * raio**2,
        2*dr**2,
        2*dr**2,
```

```
    ])

    pontos = np.array([M[i-1,j], M[i+1,j], M[i,j-1], M[i,j+1]]).reshape(4,1)

    return (np.float(coefs[1:] @ pontos)-(qdot/k_B)*2*(dr**2 * dtheta**2 * raio**2))
        /np.float(coefs[0])


if __name__ == '__main__':
    from resolve_potencial import *

    dr = 0.0005
    dtheta = np.deg2rad(0.5)
    lamb = 1.5
    erro_des = 1e-4

    #M = cria_malha(dr, dtheta)
```

### 4.2.3 Implementação da simulação dos valores do forno unindo o método de Liebman e o método de Diferenças Finitas

Listing 5: Implementação da solução do forno

```
import numpy as np
from liebmann import *
from funcoes_potencial import *

#propriedades geometricas do sistema
props_geo = {
    "R_A" : [0.03, 0.08 + 0.03],
    "R_B" : [0.03 + 0.02, 0.03 + 0.05],
    "Theta_A" : [0, np.deg2rad(40)],
    "Theta_B" : [0, np.deg2rad(18)],
}

def cria_malha(dr, dtheta):
    '''
    Funo que cria a malha para o problema, de acordo os valores de delta r e
    delta theta fornecidos. Pela simetria do problema, so  necessrio resolver
    metade da geometria da pea, ento a malha gerada  equivalente a somente
```

```
    metade da pea

    **Entradas**
    dr: variao do raio "delta r"
    dtheta: variao do ngulo"delta theta"

    **Sadas**:
    M: matriz inicialmente zerada, do tamanho necessrio
    '''
    #cria matriz inicial

    n_r = (max(props_geo['R_A']) - min(props_geo['R_A']))/dr
    n_theta = (max(props_geo["Theta_A"]) - min(props_geo["Theta_A"]))/dtheta

    M = np.zeros((int(n_r+1), int(n_theta+1)))

    return M


def define_condicao(i, j, dr, dtheta):

    '''Funo que verifica em que condio o ponto i,j da matriz ,
    para calcular o valor da varivel com a expresso correta. A
    correspondncia entre o valor numrico e a condio pode ser
    conferida na funo 'calcula_tenso'

    **Entradas**:
    i, j: coordenadas do ponto
    dr, dtheta: variaes do raio e ngulo

    **Sada**
    valor numrico correspondente  condio do ponto
    '''

    raio = i*dr + min(props_geo['R_A'])
    theta = np.rad2deg(j*dtheta + min(props_geo['Theta_A']))
    #angulos sao usados em graus nessa funcao para facilitar a analise

    if(theta >= 40):
        if(raio == 0.03):
            return 4
```

```
59          if(raio == 0.11):
60              return 5
61          else:
62              return 0 # borda superior material A
63      elif(theta == 0):
64          if(0.05 < raio < 0.08):
65              return 1 #borda inferior do material B - duplicar
66          elif(raio == 0.05):
67              return 1 #borda esquerda do material B (antes 2)
68          elif(raio == 0.08):
69              return 1 #borda direita do material B (antes 3)
70          elif(raio == 0.03):
71              return 4 #borda esquerda do material A - ajuste de matriz
72          elif(raio == 0.11):
73              return 5 #borda direita do material A - ajuste de matriz
74          else:
75              return 6 #borda inferior do material A - duplicar
76      elif(theta == 18):
77          if(0.05 < raio < 0.08):
78              return 7 #borda superior do material B
79          if(raio == 0.11):
80              return 5 #borda direita do material A
81          elif(raio == 0.03):
82              return 4 #borda esquerda do material A
83          else:
84              return 8 #interior do material A
85      elif(theta > 18):
86          if(raio == 0.11):
87              return 5 #borda direita do material A
88          elif(raio == 0.03):
89              return 4 #borda esquerda do material A
90          else:
91              return 8 #interior do material A
92      else:
93          if(0.05 < raio < 0.08):
94              return 9 #interior B
95          elif(raio == 0.05):
96              return 2
97          elif(raio == 0.08):
98              return 3
99          elif(raio == 0.03):
```

```
100             return 4
101         elif(raio == 0.11):
102             return 5
103         else:
104             return 8
105 
106 
107 def calcula_tensao(M, i, j, dr, dtheta,q_dots):
108     '''
109     Funcao que calcula a tenso (caso q_dots seja None)
110     ou a temperatura do ponto. A funo verifica qual
111     a condio do ponto e em seguida chama a funo correspondente
112 
113     10 condicoes necessarias
114     - 0: borda superior de A
115     - 1: borda inferior de B (regiao de simetria)
116     - 2: borda esquerda de B
117     - 3: borda direita de B
118     - 4: borda esquerda de A
119     - 5: borda direita de A
120     - 6: borda inferior de A (regiao de simetria)
121     - 7: borda superior de B
122     - 8: interior de A
123     - 9: interior de B
124 
125     **Entradas**
126     M: matriz dos valores a serem calculados
127     i e j: coordenadas do ponto desejado
128     dr e dtheta: variaes do raio e ngulo
129     q_dots = Valor da energia dissipada
130     (passar como None caso seja caso eltrico)
131     '''
132 
133     if q_dots is None:
134         qdot = 0
135     else:
136         qdot = q_dots[i,j]
137 
138     condicao = define_condicao(i, j, dr, dtheta)
139 
140     if(condicao == 0):
```

```
141         temp = sup_A(M, i, j, dr, dtheta, qdot)
142     elif(condicao == 1):
143         temp = inf_B(M, i, j, dr, dtheta, qdot)
144     elif(condicao == 2):
145         temp = esq_B(M, i, j, dr, dtheta, qdot)
146     elif(condicao == 3):
147         temp = dir_B(M, i, j, dr, dtheta, qdot)
148     elif(condicao == 4):
149         temp = esq_A(M, i, j, dr, dtheta, qdot)
150     elif(condicao == 5):
151         temp = dir_A(M, i, j, dr, dtheta, qdot)
152     elif(condicao == 6):
153         temp = inf_A(M, i, j, dr, dtheta, qdot)
154     elif(condicao == 7):
155         temp = sup_B(M, i, j, dr, dtheta, qdot)
156     elif(condicao == 8):
157         temp = inter_A(M, i, j, dr, dtheta, qdot)
158     elif(condicao == 9):
159         temp = inter_B(M, i, j, dr, dtheta, qdot)
160 
161     return temp
162 
163 def resolve_potencial(dr=0.001, dtheta=np.deg2rad(2), lamb=1.75, erro_des=1e-4,
        q_dots=None):
164     '''
165     Funo principal para resolver o problema do potencial eltrico ou da temperatura
166     de cada um dos pontos. A funo cria a malha e usa o mtodo de Liebmann para
            encontrar
167     os valores
168 
169     **Entradas**:
170     dr e dtheta: variaes do raio e ngulo
171     lamb: fator de sobrerelaxao
172     erro_des: erro relativo desejado
173     q_dots: valor de q_ponto, usado no caso trmico
174     '''
175 
176     #cria matriz inicialmente zerada
177     M = cria_malha(dr, dtheta)
178 
179     print(f"Matriz: {M.shape}")
```

```
180
181     M_ans = liebmann(M,
182                      func=calcula_tensao,
183                      lamb=lamb,
184                      erro_des=erro_des,
185                      dr=dr,
186                      dtheta=dtheta,
187                      q_dots=q_dots,
188                      max_steps=1.e4
189                      )
190
191     #cria_plot(M_ans, dr, dtheta)
192     return M_ans
193
194 def calcula_Qr(V, i, j, dr, dtheta, termico=False):
195     '''
196     Calcula a coordenada r do vetor J ou do vetor fluxo de calor,
197     de acordo com o parametro 'termico'
198
199     **Entradas**:
200     V: matriz com os valores de tenso ou temperatura
201     i e j: coordenadas do ponto desejado
202     dr e dtheta: variaes do raio e ngulo
203     termico: 'False' para problema eltrico, 'True' para trmico
204
205     **Sadas**
206     qr: valor da coordenada r do vetor
207     '''
208
209     condicao = define_condicao(i, j, dr, dtheta)
210
211     if termico:
212         multA = props_elet['k_A']
213         multB = props_elet['k_B']
214     else:
215         multA = props_elet['sigma_A']
216         multB = props_elet['sigma_B']
217
218     if(condicao == 4): #borda esquerda de A
219         #progressiva
220         qr = multA * (-V[i+2, j] + 4*V[i+1, j] - 3*V[i, j])/(2*dr)
```

```
221     elif(condicao == 5): #borda direita de A
222         #regressiva
223         qr = multA * (V[i-2, j] - 4*V[i-1, j] + 3*V[i, j])/(2*dr)
224     elif(condicao in [0, 8, 6]): #interior de A
225         qr = multA * (V[i+1, j] - V[i-1,j])/(2*dr)
226     else: #interior de B
227         qr = multB * (V[i+1, j] - V[i-1,j])/(2*dr)
228 
229     return qr
230 
231 def calcula_Qtheta(V, i, j, dr, dtheta, termico=False):
232     '''
233     Calcula a coordenada theta do vetor J ou do vetor fluxo de calor,
234     de acordo com o parametro 'termico'
235 
236     **Entradas**:
237     V: matriz com os valores de tenso ou temperatura
238     i e j: coordenadas do ponto desejado
239     dr e dtheta: variaes do raio e ngulo
240     termico: 'False' para problema eltrico, 'True' para trmico
241 
242     **Sadas**
243     qtheta: valor da coordenada theta do vetor
244     '''
245 
246     condicao = define_condicao(i, j, dr, dtheta)
247     raio = 0.03 + i*dr
248 
249     if termico:
250         multA = props_elet['k_A']
251         multB = props_elet['k_B']
252     else:
253         multA = props_elet['sigma_A']
254         multB = props_elet['sigma_B']
255 
256 
257     if(condicao == 0): #regressiva
258         qtheta = multA * (V[i, j-2] - 4*V[i, j-1] + 3*V[i, j])/(2*dtheta*raio)
259     elif(condicao in [6, 1]): #central simetrica A
260         qtheta = 0
261     elif(condicao in [4, 5, 8]): #central A
```

```
262         try:
263             qtheta = multA* (V[i, j+1] - V[i,j-1])/(2*dtheta*raio)
264         except:
265             qtheta = 0 #despreza pontos extremos de A
266     else: #central B
267         qtheta = multB* (V[i, j+1] - V[i,j-1])/(2*dtheta*raio)
268 
269     return qtheta
270 
271 def calcula_J(V_ans, dr, dtheta, termico=False):
272     '''
273     Calcula vetor J ou Q e armazena o resultado em uma matriz tridimensional,
274     onde J[i, j, 0] corresponde  coordenada r do vetor e J[i, j, 1]  coordenada
275     theta
276 
277     **Entradas**:
278     V_ans: matriz com as tenses (para o J) ou temperaturas (para o Q)
279     dr e dtheta: variaes do raio e ngulo
280     termico: 'False' para problema eltrico, 'True' para trmico
281 
282     **Sadas**
283     J: matriz tridimensional com os vetores correspondentes de cada posio i,j
284     '''
285 
286     J = np.zeros((V_ans.shape[0], V_ans.shape[1], 2)) #guarda os vetores
287     for i in range(J.shape[0]):
288         for j in range(J.shape[1]):
289             J[i, j, 0] = -calcula_Qr(V_ans, i, j, dr, dtheta, termico)
290             J[i, j, 1] = -calcula_Qtheta(V_ans, i, j, dr, dtheta, termico)
291 
292     return J
293 
294 def calcula_qponto(J_ans, dr, dtheta):
295     '''
296     Funo que calcula o valor de q_ponto de cada valor i, j
297     a partir da densidade de corrente, para utilizar no problema trmico
298 
299     **Entradas**:
300     J: matriz tridimensional com os vetores correspondentes de cada posio i,j
301     dr e dtheta: variaes do raio e ngulo
302 
```

```
    **Sadas**:
    qdot: matriz com os valores da energia dissipada em cada posio i, j
    '''
    qdot = np.zeros((J_ans.shape[0], J_ans.shape[1]))

    for i in range(qdot.shape[0]):
        for j in range(qdot.shape[1]):
            condicao = define_condicao(i, j, dr, dtheta)

            if(condicao in [0, 4, 5, 6, 8]):
                sigma = props_elet['sigma_A']
            else:
                sigma = props_elet['sigma_B']

            qdot[i, j] = -(np.linalg.norm(J_ans[i, j, :]))**2/sigma

    return qdot

def calcula_corrente(J_ans, dtheta, rmax=False):
    '''
    Funo de integrao, usada para calcular a corrente ou o fluxo por
    convecco. A integral foi aproximada por meio do mtodo dos trapzios.

    **Entradas**:
    J_ans: matriz tridimensional com os vetores correspondentes de cada posio i,j
    dtheta: variao do ngulo
    rmax: 'False' para raio 0.03, 'True' para raio 0.11
    '''
    #aproxima integral para soma
    soma = 0
    if rmax:
        raio = 0.11
        for j in range(1,J_ans.shape[1]):
            soma += J_ans[-1,j,0] + J_ans[-1,j-1,0] #verificar o -1
    else:
        raio = 0.03
        for j in range(1,J_ans.shape[1]):
            soma += J_ans[0,j,0] + J_ans[0,j-1,0]

    return soma*2*raio*dtheta/2
```

```
344 if __name__ == "__main__":
345     resolve_potencial()
```

### 4.2.4 Implementação das funções que geram os gráficos

Listing 6: Implementação dos gráficos

```
1  from matplotlib.axis import XAxis
2  import matplotlib.pyplot as plt
3  import numpy as np
4  import seaborn as sns
5  import plotly.express as ex
6  import plotly.graph_objects as go
7  import plotly.figure_factory as ff
8  import pandas as pd
9
10 def heatmap_2d(M, dr, dtheta, xlabel, ylabel, title, legend, tipo='C', reflexao=True
       ):
11     '''
12     Funo para gerar um heatmap de 2 dimenses, a partir dos valores armazenados
13     na matriz M
14
15     **Entradas**:
16     M: matriz com os valores das variveis desejadas
17     dr: variao "delta r"
18     dtheta: variao "delta theta"
19     xlabel, ylabel, title: labels para os eixos e ttulo do grfico
20     legend: legenda da varivel desejada
21     tipo: "C" para dados contnuos, "D" para dados discretos
22     reflexao: como calcula-se somente metade da pea, caso este valor seja 'True'
23     ser feita a reflexo no eixo x para mostrar a pea inteira
24     '''
25
26     X = []
27     Y = []
28     Valores = []
29
30     for i in range(M.shape[0]): #raio
31         for j in range(M.shape[1]): #angulo
32             raio = 0.03 + i*dr
```

```
            angulo = j*dtheta

            #conversao de coord. polar para cartesiana
            x = raio*np.cos(angulo)
            y = raio*np.sin(angulo)

            valor = M[i,j]

            X.append(x)
            Y.append(y)
            Valores.append(valor)

            #reflexao no eixo X
            if reflexao:
                X.append(x)
                Y.append(-y)
                Valores.append(valor)

    X, Y = np.array(X), np.array(Y)
    if tipo == 'D':
        Valores = [str(i) for i in Valores]

    df = pd.DataFrame({xlabel: X, ylabel: Y, legend: Valores})

    fig = ex.scatter(df, x=X, y=Y, color=legend)

    fig.update_layout(
        title = dict(
            text=title,
            x=0.5,
            xanchor='center',
            yanchor='top',
        ),

        xaxis_title = xlabel,

        yaxis_title = ylabel,

        legend = dict(
            yanchor='top',
            y=1.02,
```

```
        ),

        font=dict(
            family="Courier New",
            size=18,
        ),

    )

    fig.show()

def surf_3d(M, dr, dtheta, xlabel, ylabel, zlabel, title):
    '''
    Funo para gerar um grfico tridimensional da varivel calculada,
    de tipo 'surf'

    **Entradas**:
    M: matriz com os valores das variveis desejadas
    dr: variao "delta r"
    dtheta: variao "delta theta"
    xlabel, ylabel, zlabel, title: labels para os eixos e ttulo do grfico
    '''

    raios = [0.03 + i*dr for i in range(M.shape[0])]
    angulos = [j*dtheta for j in range(M.shape[1])]

    ang_mesh, raio_mesh = np.meshgrid(angulos, raios)

    x_mesh = raio_mesh*np.cos(ang_mesh)
    y_mesh = raio_mesh*np.sin(ang_mesh)

    fig = go.Figure(data=[go.Surface(z=M, x=x_mesh, y=y_mesh), go.Surface(z=M, x=
        x_mesh, y=-y_mesh)])

    fig.update_layout(
       title = dict(
            text=title,
            x=0.5,
            xanchor='center',
        ),

```

```
114         scene = dict(
115             xaxis_title=xlabel,
116             yaxis_title=ylabel,
117             zaxis_title=zlabel,
118         ),
119 
120         font=dict(
121             family="Courier New",
122             size=18,
123         ),
124     )
125 
126     fig.show()
127 
128 
129 
130 def quiver(J, dr, dtheta, xlabel, ylabel, title, plot='half', arrow_scale=1, lib='
        matplotlib'):
131     '''
132     Funo para gerar um grfico bidimensional de um campo vetorial. Por se tratar de
            um grfico
133     mais pesado, recomenda-se utilizar o matplotlib ao inves do plotly, mas ambas as
             opces esto
134     implementadas
135 
136     **Entradas**:
137     J: matriz tridimensional com os valores dos vetores
138     dr: variao "delta r"
139     dtheta: variao "delta theta"
140     xlabel, ylabel, title: labels para os eixos e ttulo do grfico
141     plot: 'half' (metade - recomendado) ou 'full' (pea inteira)
142     arrow_scale: escala da seta, utilizada somente na verso do plotly
143     lib: 'matplotlib' (recomendada) ou 'plotly'
144     '''
145 
146 
147     raios = [0.03 + i*dr for i in range(J.shape[0])]
148     angulos = [j*dtheta for j in range(J.shape[1])]
149 
150     ang_mesh, raio_mesh = np.meshgrid(angulos, raios)
151 
```

```
152     x_mesh = raio_mesh*np.cos(ang_mesh)
153     y_mesh = raio_mesh*np.sin(ang_mesh)
154 
155     x_vec = np.zeros((J.shape[0], J.shape[1]))
156     y_vec = np.zeros((J.shape[0], J.shape[1]))
157 
158     for i in range(J.shape[0]):
159         for j in range(J.shape[1]):
160             raio = 0.03 + i*dr
161             angulo = j*dtheta
162             #projeta
163             qr = J[i, j, 0]
164             qtheta = J[i, j, 1]
165             x_vec[i, j] = (qr - qtheta)*np.cos(angulo)
166             y_vec[i, j] = (qr + qtheta)*np.sin(angulo)
167 
168     if lib == 'matplotlib':
169         plt.quiver(x_mesh, y_mesh, x_vec, y_vec)
170         plt.xlabel(xlabel)
171         plt.ylabel(ylabel)
172         plt.title(title)
173         plt.show()
174 
175 
176     elif lib == 'plotly':
177 
178         fig1 = ff.create_quiver(x_mesh, y_mesh, x_vec, y_vec, arrow_scale=arrow_scale
                )
179 
180 
181         if plot == 'full':
182             fig2 = ff.create_quiver(x_mesh, -y_mesh, x_vec, -y_vec, arrow_scale=
                    arrow_scale)
183 
184             fig1.add_traces(data = fig2.data)
185             fig1.update_traces(marker=dict(color='blue'))
186 
187         fig1.update_layout(
188         title = dict(
189                 text=title,
190                 x=0.5,
```

```
191                 xanchor='center',
192             ),
193 
194             scene = dict(
195                 xaxis_title=xlabel,
196                 yaxis_title=ylabel,
197             ),
198 
199             font=dict(
200                 family="Courier New",
201                 size=18,
202             ),
203         )
204 
205         fig1.show()
206 
207 
208 if __name__ == '__main__':
209     from resolve_potencial import define_condicao, cria_malha
210 
211     dr = 0.001
212     dtheta = np.deg2rad(1)
213     M = cria_malha(dr, dtheta)
214     print(M.shape)
215 
216     for i in range(M.shape[0]):
217         for j in range(M.shape[1]):
218             M[i, j] = define_condicao(i,j, dr, dtheta)
219 
220     heatmap_2d(M, dr, dtheta, title='Condio dos pontos', xlabel='X (m)', ylabel='Y (
        m)', legend='Condico', tipo='D', reflexao=False)
221     #surf_3d(M, dr, dtheta, title='Condicao dos pontos', xlabel='X (m)', ylabel='Y (
        m)', zlabel='Tensao (V)')
```

### 4.2.5 Uso das funções de resolução para gerar os resultados

Para gerar todos os gráficos e valores do Exercício Programa, basta executar este código

Listing 7: Resolução da simulação do forno

```python
from resolve_potencial import *
from plots import *
import numpy as np
import time

print("PMR3401 - EP1 2022")
print("Ariel Guerreiro - 11257838")
print("Felipe Azank - 11258137\n\n")

print(f"valor da condutividade eltrica em A: {props_elet['sigma_A']}")
print(f"valor da condutividade eltrica em B: {props_elet['sigma_B']}")

'''
Arquivo principal, responsvel por resolver
toda a parte 2 do exerccio-programa
'''

#calcula a tensao de cada ponto da malha
dr = 0.001
dtheta = np.deg2rad(1)

print(f"Discretizao adotada:\ndelta_r = {dr}\ndelta_theta = {dtheta:.5f} rad ({np.
    rad2deg(dtheta)})")

#calculo das tensoes eletricas
V_ans = resolve_potencial(
    dr=dr,
    dtheta=dtheta,
    lamb=1.75,
    erro_des=1e-4
    )

#calculo do vetor densidade de corrente
J_ans = calcula_J(V_ans, dr, dtheta)

#calculo da energia dissipada por efeito Joule
q_ans = calcula_qponto(J_ans, dr, dtheta)

#calcula corrente eletrica pelo raio maximo.
#Como o problema  resolvido para metade da pea,
```

```
#o valor deve ser dobrado
I_ans_rmax = 2*calcula_corrente(J_ans, dtheta, rmax=True)

#ddp
deltaV = 100

#calcula resisncia da pea
R_max = deltaV/I_ans_rmax

print(f"Corrente calculada para Rmax = 0.11: {I_ans_rmax} A")
print(f"Resistncia equivalente: {R_max} Ohms")
print(f"Potncia dissipada MAX: {I_ans_rmax**2 * R_max}")


#calcula corrente eletrica pelo raio minimo.
I_ans = 2*calcula_corrente(J_ans, dtheta, rmax=False)

R_min = deltaV/I_ans

print(f"Corrente calculada para Rmin = 0.03: {I_ans} A")
print(f"Resistncia equivalente: {R_min} Ohms")
print(f"Potncia dissipada: {I_ans**2 * R_min}")


#calcula temperaturas com as mesmas funcoes do potencial
T_ans = resolve_potencial(
    dr=dr,
    dtheta=dtheta,
    lamb=1.75,
    erro_des=1e-4,
    q_dots = q_ans)

#calculo do fluxo de calor
fluxo_ans = calcula_J(T_ans, dr, dtheta, termico=True)

#calculo da quantidade de calor (unidade W) que flui pela parede de conveco:
qt_calor_ans = 2*calcula_corrente(fluxo_ans, dtheta, rmax=True)

print(f"Quantidade de calor que flui pela parede de conveco: {qt_calor_ans} W")


```

```
81
82  #plots das condies
83
84  #heatmap da tensao eletrica
85  heatmap_2d(
86      V_ans, dr, dtheta,
87      title='Tenso eltrica',
88      xlabel='Coordenada X (m)',
89      ylabel='Coordenada Y (m)',
90      legend='Tenso (V)'
91  )
92
93  #surface plot da tensao eletrica
94  surf_3d(
95      V_ans, dr, dtheta,
96      title='Tenso eltrica',
97      xlabel='Coordenada X (m)',
98      ylabel='Coordenada Y (m)',
99      zlabel='Tenso (V)'
100 )
101
102
103
104 #surface plot da fonte de calor equivalente
105 surf_3d(
106     q_ans, dr, dtheta,
107     title='Fonte de calor equivalente',
108     xlabel='Coordenada X (m)',
109     ylabel='',
110     zlabel='',
111 )
112
113 surf_3d(
114     T_ans, dr, dtheta,
115     title='Temperatura dos Pontos do Forno',
116     xlabel='',
117     ylabel='',
118     zlabel='T (K)'
119 )
120
121 #quiver plot da densidade de corrente
```

```
quiver(
    J_ans, dr, dtheta,
    title='Vetor densidade de corrente',
    xlabel='Coordenada X (m)',
    ylabel='Coordenada Y (m)',
    arrow_scale=1e-5,
)

#quiver plot do vetor fluxo de calor
quiver(
    fluxo_ans, dr, dtheta,
    title='Vetor fluxo de calor (W/m^2)',
    xlabel='Coordenada X (m)',
    ylabel='Coordenada Y (m)',
)
```