# Problema de Resfriamento de Componente Eletrônico

Introdução a Equações Diferenciais Ordinárias

**Autores:**
Lucas Rodrigues Estorck Pinto - 202310349511
Tainá Martins Macário - 202310070411

**Professor:**
Rafael Barbosa Libotte

Rio de Janeiro, Nova Friburgo

6 de maio de 2025

# Conteúdo

# 1 Introdução

O resfriamento de componentes eletrônicos é essencial para seu bom funcionamento e prolongação de sua vida útil. Tal processo, em um ambiente controlado, pode ser modelado pela Lei de Resfriamento de Newton, que foi fornecida na forma da Equação 1.

$$\frac{dT}{dt} = -k(T - T_a) \tag{1}$$

Onde:

- $T(t)$ é a temperatura do componente no instante $t$ (em $^\circ C$);
- $T_a$ é a temperatura ambiente (constante);
- $k$ é uma constante positiva relacionada a taxa de resfriamento (em $s^{-1}$).

Neste trabalho, exploraremos a forma de equação diferencial fornecida analítica e numericamente, buscando compreender seu comportamento e funcionamento. A solução analítica fornece um resultado preciso e exato para a temperatura em função do tempo, mas, em algumas equações de alta complexidade, não se torna possível utilizar esta forma, fazendo com que os métodos numéricos sejam essenciais.

Assim, no contexto apresentado, utilizaremos dois métodos numéricos amplamente disseminados: o método de Euler e o método de Runge-Kutta de quarta ordem. Ambos serão aplicados com diferentes tamanhos de passo, a fim de avaliar sua precisão e eficiência na aproximação da solução analítica. A eficiência dos métodos e da solução analítica serão comparados graficamente, calculando também o desvio percentual destes para aprofundar a análise.

Esta análise permitirá a discussão de qual método se desempenhou melhor e como o tamanho do passo influencia na precisão das soluções numéricas. Este trabalho ilustra, principalmente, a aplicação de equações diferenciais em problemas reais, evidenciando também a importância dos métodos numéricos na engenharia e na ciência.

## 2 Resolução Analítica

Inicialmente, a equação diferencial será resolvida analiticamente, para que posteriormente haja a comparação entre a resolução analítica e a resolução numérica. Tendo em conta a Equação 1, os parâmetros iniciais disponibilizados para a resolução do problema foram:

- $T_a = 25°\text{C}$
- $T(0) = 90°\text{C}$
- $k = 0,1$

Como a EDO é de primeira ordem e separável, podemos separar as variáveis T e t:

$$\frac{dT}{dt} = -k(T - T_a)$$
$$\frac{dT}{T-T_a} = -k \cdot dt$$

Assim, podemos integrar ambos os lados para remover os diferenciais:

$$\int \frac{dT}{T - T_a} = \int -k \cdot dt$$

Resolvendo ambas as integrais:

$$\ln |T - T_a| = -kt + C$$

Para remover o logaritmo natural, vamos aplicar exponencial dos dois lados, posteriormente simplificando onde é necessário e isolando para $T(t)$:

$$e^{\ln |T-T_a|} = e^{-kt+C}$$
$$T(t) = C \cdot e^{-kt} + T_a$$

Agora, basta aplicar os valores de parâmetros iniciais apresentados para obtermos a forma final da EDO. Utilizando $T_a = 25$ e $k = 0.1$, obtemos:

$$T(t) = C \cdot e^{-0.1t} + 25$$

Por fim, para encontrar a constante $C$, basta aplicarmos $T(0) = 90$:

$$90 = C \cdot e^{-0.1 \cdot 0} + 25$$
$$90 = C \cdot 1 + 25$$
$$C = 90 - 25$$
$$C = 65$$

Desta forma, ao aplicarmos os parâmetros conhecidos e a constante obtida, chegamos à solução analítica exibida na Equação 2.

$$\boxed{T(t) = 65e^{-0.1t} + 25} \tag{2}$$

# 3 Resolução Numérica

Nesta seção, será explicado o código utilizado para a solução numérica do problema. A simulação e os cálculos foram realizados na linguagem Python, versão 3.13.3, utilizando como principais bibliotecas a NumPy (2.2.5) para as operações numéricas e a Matplotlib (3.10.1) para gerar os gráficos comparativos. O ambiente de execução foi um sistema operacional OpenSUSE Tumbleweed, com o kernel otimizado Liquorix 6.14.6, proporcionando um desempenho estável. [1] [2] [3]

## 3.1 Código Explicado

Inicialmente, foram importadas as bibliotecas já explicadas. NumPy foi importada com a sigla `np` para facilitar sua citação ao longo do código, enquanto a Matplotlib foi importada como `plt` pela mesma razão.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import matplotlib
4  matplotlib.use('TkAgg')
```

Após a importação, definiu-se os parâmetros iniciais já fornecidos no problema e a constante calculada previamente para plotar o gráfico da solução inicial.

```
5  # Parâmetros do problema
6  T0 = 90.0  # Temperatura inicial (ŕC)
7  Ta = 25.0  # Temperatura ambiente (ŕC)
8  C = T0 - Ta  # Constante da solução (T0 - Ta)
9  k = 0.1    # Constante de resfriamento (s^-1)
10 t_max = 30.0  # Tempo máximo (s)
11 t_pontos = [0, 10, 20, 30]
```

A primeira função definida foi a analítica, pois é a mais 'simples' e a que já está previamente definida, fora do código. Perceba que as variáveis utilizadas para defini-la são similares as utilizadas no relatório previamente.

```
12 # Solução analítica: T(t) = (T0 - Ta) * exp(-k * t) + Ta
13 def solucao_analitica(t):
14     return C * np.exp(-k * t) + Ta
15
16 # Validar solução analítica
17 print("Validação da Solução Analítica (T(t) = 65 * exp(-0.1 * t) +
       25):")
18 for t in t_pontos:
19     T_ana = solucao_analitica(t)
20     print(f"t = {t:<2}: T_ana = {T_ana:.8f}")
```

Agora, inicia-se a definição dos métodos numéricos já citados, sendo eles o Método de Euler e o Método de Runge-Kutta de $4^{\circ}$ ordem. Ambos estes métodos recebem como parâmetro o passo de integração $h$,

o tempo máximo $t_{max}$, a temperatura inicial $T_0$, a constante de resfriamento $k$ e a temperatura ambiente $T_a$.

O método de Euler aplica uma aproximação linear para estimar a temperatura em cada passo de tempo, sendo este método mais simples e computacionalmente mais leve, o que causa um custo na precisão do método, especialmente ao definir passos grandes. Enquanto isso, o método RK4 (Runge-Kutta de $4°$ ordem) realiza quatro cálculos intermediários em cada iteração para estimar a inclinação da curva, logicamente resultando em uma solução mais precisa mesmo quando se fornece passos grandes [4]. Ambos os métodos foram projetados para retornar arrays com os valores de tempo e as respectivas temperaturas simuladas.

Os métodos para tempos específicos geram resultados para fins comparativos em tempos pré-definidos, enquanto os métodos para gráficos são feitos para gerarem curvas de solução.

```
21          # Método de Euler para tempos específicos
22          def metodo_euler_pontos(t_pontos, h, T0, k, Ta):
23              T = [T0]  # Garante que o primeiro ponto é exatamente a condição
                    inicial
24              t_prev = t_pontos[0]
25              T_prev = T0
26
27              for t_target in t_pontos[1:]:
28                  # Calcula quantos passos são necessários para chegar ao
                        próximo ponto
29                  steps = int(np.ceil((t_target - t_prev) / h))
30                  adjusted_h = (t_target - t_prev) / steps
31
32                  for _ in range(steps):
33                      T_next = T_prev + adjusted_h * (-k * (T_prev - Ta))
34                      t_prev += adjusted_h
35                      T_prev = T_next
36
37                  T.append(T_prev)
38
39              return np.array(t_pontos), np.array(T)
40
41          # Método de Runge-Kutta de 4ł ordem para tempos específicos
42          def runge_kutta_4_pontos(t_pontos, h, T0, k, Ta):
43              T = [T0]
44              t_prev = t_pontos[0]
45              T_prev = T0
46
47              for t_target in t_pontos[1:]:
48                  # Calcula quantos passos são necessários para chegar ao
                        próximo ponto
49                  steps = int(np.ceil((t_target - t_prev) / h))
50                  adjusted_h = (t_target - t_prev) / steps
51
52                  for _ in range(steps):
53                      k1 = -k * (T_prev - Ta)
```

```
                        k2 = -k * (T_prev + adjusted_h/2 * k1 - Ta)
                        k3 = -k * (T_prev + adjusted_h/2 * k2 - Ta)
                        k4 = -k * (T_prev + adjusted_h * k3 - Ta)
                        T_next = T_prev + (adjusted_h/6) * (k1 + 2*k2 + 2*k3 + k4
                            )
                        t_prev += adjusted_h
                        T_prev = T_next

                    T.append(T_prev)

                return np.array(t_pontos), np.array(T)

            # Método de Euler para gráficos
            def metodo_euler(h, t_max, T0, k, Ta):
                t = np.arange(0, t_max + h, h)
                T = np.zeros(len(t))
                T[0] = T0
                for i in range(1, len(t)):
                    T[i] = T[i-1] + h * (-k * (T[i-1] - Ta))
                return t, T

            # Método de Runge-Kutta de 4ł ordem para gráficos
            def runge_kutta_4(h, t_max, T0, k, Ta):
                t = np.arange(0, t_max + h, h)
                T = np.zeros(len(t))
                T[0] = T0
                for i in range(1, len(t)):
                    k1 = -k * (T[i-1] - Ta)
                    k2 = -k * (T[i-1] + h/2 * k1 - Ta)
                    k3 = -k * (T[i-1] + h/2 * k2 - Ta)
                    k4 = -k * (T[i-1] + h * k3 - Ta)
                    T[i] = T[i-1] + (h/6) * (k1 + 2*k2 + 2*k3 + k4)
                return t, T
```

Para avaliar a precisão das soluções numéricas definiu-se uma função que calcula o desvio relativo percentual entre os resultados numéricos e analíticos da equação diferencial. A fórmula definida foi a que está na Equação 3.

$$Desvio = 100 \cdot \left| \frac{T_{num} - T_{ana}}{T_{ana}} \right| \tag{3}$$

Onde, intuitivamente, $T_{num}$ representam os valores de temperatura calculados numericamente e $T_{ana}$ a solução analítica correspondente. Além disto, gerou-se um array de tempo com alta resolução, utilizando linspace para obter uma curva suave da solução analítica ao longo do intervalo, comparando a visualização e comparação gráfica com os métodos numéricos.

```
            # Calcula o desvio relativo percentual
            def desvio_relativo(T_num, T_ana):
                return 100 * np.abs((T_num - T_ana) / T_ana)

            # Solução analítica para gráficos
```

```
t_analitico = np.linspace(0, t_max, 1000)
T_analitico = solucao_analitica(t_analitico)
```

Agora sim, com o objetivo de comparar os desempenhos, calcula-se as soluções utilizando os dois métodos, aplicando tamanhos diferentes de passos para cada um. Para o método de Euler utilizou-se $h = [1.0,\ 0.65,\ 0.1]$ enquanto no RK4 utilizou-se $h = [2.0,\ 1.0,\ 0.65]$. Pela diferença de precisão dos métodos, é justificável a diferenciação dos passos, inclusive, torna-se possível analisar o impacto deste parâmetro e como isso refletirá nos desvios percentuais.

```
# Calcula soluções numéricas para gráficos
t_euler_1, T_euler_1 = metodo_euler(1.0, t_max, T0, k, Ta)
t_euler_05, T_euler_05 = metodo_euler(0.65, t_max, T0, k, Ta)
t_euler_01, T_euler_01 = metodo_euler(0.1, t_max, T0, k, Ta)
t_rk4_2, T_rk4_2 = runge_kutta_4(2.0, t_max, T0, k, Ta)
t_rk4_1, T_rk4_1 = runge_kutta_4(1.0, t_max, T0, k, Ta)
t_rk4_05, T_rk4_05 = runge_kutta_4(0.65, t_max, T0, k, Ta)
```

A seguir foi gerado o gráfico comparativo entre as soluções analíticas e numéricas que utilizam diferentes passos $h$. Esse gráfico permitirá observar visualmente a convergência dos métodos numéricos à solução analítica à medida que o passo $h$ diminui. O gráfico será apresentado em seu tópico pertinente. Foi gerado, também, um gráfico do erro absoluto das soluções para buscar compreender melhor como estes erros se portam.

```
# Gráfico de temperatura
plt.figure(figsize=(10, 6))
plt.plot(t_analitico, T_analitico, 'k-', label='Solução Analítica',
    linewidth=2)
plt.plot(t_euler_1, T_euler_1, 'r--', label='Euler h=1.0')
plt.plot(t_euler_05, T_euler_05, 'g--', label='Euler h=0.65')
plt.plot(t_euler_01, T_euler_01, 'b--', label='Euler h=0.1')
plt.plot(t_rk4_2, T_rk4_2, 'c-.', label='RK4 h=2.0')
plt.plot(t_rk4_1, T_rk4_1, 'm-.', label='RK4 h=1.0')
plt.plot(t_rk4_05, T_rk4_05, 'y-.', label='RK4 h=0.65')
plt.xlabel('Tempo (s)')
plt.ylabel('Temperatura (řC)')
plt.title('Resfriamento de Componente Eletrônico')
plt.legend()
plt.grid(True)
plt.savefig('grafico_resfriamento.png')
print("Gráfico de temperatura salvo como 'grafico_resfriamento.png'")
plt.show()
plt.close()

# Gráfico de erro absoluto (escala logarítmica)
plt.figure(figsize=(10, 6))
plt.semilogy(t_euler_1, np.abs(T_euler_1 - solucao_analitica(
    t_euler_1)), 'r--', label='Erro Euler h=1.0')
plt.semilogy(t_euler_05, np.abs(T_euler_05 - solucao_analitica(
    t_euler_05)), 'g--', label='Erro Euler h=0.65')
plt.semilogy(t_euler_01, np.abs(T_euler_01 - solucao_analitica(
    t_euler_01)), 'b--', label='Erro Euler h=0.1')
```

```
124             plt.semilogy(t_rk4_2, np.abs(T_rk4_2 - solucao_analitica(t_rk4_2)), '
                    c-.', label='Erro RK4 h=2.0')
125             plt.semilogy(t_rk4_1, np.abs(T_rk4_1 - solucao_analitica(t_rk4_1)), '
                    m-.', label='Erro RK4 h=1.0')
126             plt.semilogy(t_rk4_05, np.abs(T_rk4_05 - solucao_analitica(t_rk4_05))
                    , 'y-.', label='Erro RK4 h=0.65')
127             plt.xlabel('Tempo (s)')
128             plt.ylabel('Erro Absoluto (řC)')
129             plt.title('Erro Absoluto das Soluções Numéricas')
130             plt.legend()
131             plt.grid(True)
132             plt.savefig('grafico_erro_absoluto.png')
133             print("Gráfico de erro absoluto salvo como 'grafico_erro_absoluto.png
                    '")
134             plt.show()
135             plt.close()
```

Com os dados das soluções numéricas e analítica calculados, foram selecionados pontos específicos no tempo para avaliar a precisão dos métodos computacionais aplicados. Os tempos utilizados foram de $t = [0,\ 10,\ 20,\ 30]$, permitindo a verificação do comportamento do erro ao longo do intervalo simulado. As comparações foram organizadas em uma estrutura de dicionário, armazenando os desvios relativos percentuais para cada combinação de método e passo. Essa abordagem facilita a análise individual dos resultados e sua apresentação gráfica/tabelada posterior. A extração dos valores em tempos exatos foi feita de forma direta por meio de indexação dos arrays, garantindo consistência comparativa.

```
136             # Calcula soluções numéricas nos tempos exatos
137             _, T_euler_1_pontos = metodo_euler_pontos(t_pontos, 1.0, T0, k, Ta)
138             _, T_euler_05_pontos = metodo_euler_pontos(t_pontos, 0.65, T0, k, Ta)
139             _, T_euler_01_pontos = metodo_euler_pontos(t_pontos, 0.1, T0, k, Ta)
140             _, T_rk4_2_pontos = runge_kutta_4_pontos(t_pontos, 2.0, T0, k, Ta)
141             _, T_rk4_1_pontos = runge_kutta_4_pontos(t_pontos, 1.0, T0, k, Ta)
142             _, T_rk4_05_pontos = runge_kutta_4_pontos(t_pontos, 0.65, T0, k, Ta)
143
144
145             # Verificação do tempo inicial
146             print("\nVerificação do tempo inicial t=0:")
147             print(f"Solução analítica: {solucao_analitica(0):.8f}")
148             print(f"Euler h=1.0: {T_euler_1_pontos[0]:.8f}")
149             print(f"RK4 h=2.0: {T_rk4_2_pontos[0]:.8f}")
150
151             # Imprime tabela de temperaturas para depuração
152             print("\nValores de Temperatura (para depuração):")
153             print(f"{'t (s)':<10} {'Analítica':<15} {'Euler h=1.0':<15} {'Euler h
                    =0.65':<15} {'Euler h=0.1':<15} {'RK4 h=2.0':<15} {'RK4 h
                    =1.0':<15} {'RK4 h=0.65':<15}")
154             for i, t in enumerate(t_pontos):
155                 T_ana = solucao_analitica(t)
156                 print(f"{t:<10} {T_ana:<15.8f} {T_euler_1_pontos[i]:<15.8f} {
                        T_euler_05_pontos[i]:<15.8f} {T_euler_01_pontos[i]:<15.8f} "
157                       f"{T_rk4_2_pontos[i]:<15.8f} {T_rk4_1_pontos[i]:<15.8f} {
                            T_rk4_05_pontos[i]:<15.8f}")
158
```

```
159             # Calcula e imprime tabela de desvios relativos
160             desvios = {
161                 't': t_pontos,
162                 'Euler h=1.0': [],
163                 'Euler h=0.65': [],
164                 'Euler h=0.1': [],
165                 'RK4 h=2.0': [],
166                 'RK4 h=1.0': [],
167                 'RK4 h=0.65': []
168             }
169
170             for i, t in enumerate(t_pontos):
171                 T_ana = solucao_analitica(t)
172                 desvios['Euler h=1.0'].append(desvio_relativo(T_euler_1_pontos[i
                        ], T_ana))
173                 desvios['Euler h=0.65'].append(desvio_relativo(T_euler_05_pontos[
                        i], T_ana))
174                 desvios['Euler h=0.1'].append(desvio_relativo(T_euler_01_pontos[i
                        ], T_ana))
175                 desvios['RK4 h=2.0'].append(desvio_relativo(T_rk4_2_pontos[i],
                        T_ana))
176                 desvios['RK4 h=1.0'].append(desvio_relativo(T_rk4_1_pontos[i],
                        T_ana))
177                 desvios['RK4 h=0.65'].append(desvio_relativo(T_rk4_05_pontos[i],
                        T_ana))
178
179             print("\nTabela de Desvio Relativo Percentual:")
180             print(f"{'t (s)':<10} {'Euler h=1.0':<15} {'Euler h=0.65':<15} {'
                    Euler h=0.1':<15} {'RK4 h=2.0':<15} {'RK4 h=1.0':<15} {'RK4 h
                    =0.65':<15}")
181             for i in range(len(t_pontos)):
182                 print(f"{t_pontos[i]:<10} {desvios['Euler h=1.0'][i]:<15.8f} {
                        desvios['Euler h=0.65'][i]:<15.8f} {desvios['Euler h=0.1'][i
                        ]:<15.8f} "
183                       f"{desvios['RK4 h=2.0'][i]:<15.8f} {desvios['RK4 h=1.0'][i
                            ]:<15.8f} {desvios['RK4 h=0.65'][i]:<15.8f}")
```

# 4 Resultados e discussão

A análise dos resultados obtidos numericamente evidencia a diferença de desempenho entre os métodos numéricos utilizados para diferentes tamanhos de passo $h$ em relação a solução analítica da equação de resfriamento.

Na Tabela 1 é possível observar que todos os métodos apresentam um desvio inicial de 0% quando $t = 0$, comprovando que estes acertaram a condição inicial $T_0 = 90°\text{C}$. Isto é coerente pois a solução analítica e a numérica partem do mesmo ponto inicial, sem acumular erros até então.

| | Euler | | | RK4 | | |
|---|---|---|---|---|---|---|
| $t$ (s) | $h = 1.0$ | $h = 0.65$ | $h = 0.1$ | $h = 2.0$ | $h = 1.0$ | $h = 0.65$ |
| 0 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 | 0.00000000 |
| 10 | 2.55164559 | 1.24831672 | 0.24546347 | 0.00077036 | 0.00004428 | 0.00000265 |
| 20 | 2.64614116 | 1.31226424 | 0.25942587 | 0.00082311 | 0.00004716 | 0.00000283 |
| 30 | 1.70254796 | 0.85572016 | 0.17120389 | 0.00054181 | 0.00003115 | 0.00000187 |

Tabela 1: Tabela de Desvio Relativo Percentual

| | Analítica | Euler | | | RK4 | | |
|---|---|---|---|---|---|---|---|
| $t$ (s) | $T(t)$ | $h = 1.0$ | $h = 0.65$ | $h = 0.1$ | $h = 2.0$ | $h = 1.0$ | $h = 0.65$ |
| 0 | 90.00000000 | 90.00000000 | 90.00000000 | 90.00000000 | 90.00000000 | 90.00000000 | 90.00000000 |
| 10 | 48.91216368 | 47.66409861 | 48.30158496 | 48.79210218 | 48.91254048 | 48.91218534 | 48.91216497 |
| 20 | 33.79679341 | 32.90248255 | 33.35322918 | 33.70911578 | 33.79706765 | 33.79608935 | 33.79679437 |
| 30 | 28.23615944 | 27.75542529 | 27.99453693 | 28.18781804 | 28.23631243 | 28.23616824 | 28.23615997 |

Tabela 2: Valores de temperatura obtidos por diferentes métodos.

Ao longo do tempo, o comportamento dos métodos diverge. Primeiramente, o método de Euler, especialmente quando submetido à um passo maior ($h = 1.0$), atinge um desvio de até 2.65%, evidenciando a limitação que este método possui para passos maiores, o que já se espera de um esquema de primeira ordem. Ao reduzir o passo para $h = 0.65$, o erro cai para cerca de 1.31% e continua caindo conforme a diminuição do passo $h$ o que apenas comprova o previsto teoricamente da proporcionalidade linear erro-passo.

Agora, pensando sobre o método de Runge-Kutta de 4° ordem, é visível que mesmo com passos altos, como $h = 2.0$, este possui um erro máximo de -0.00082%, extremamente inferior ao erro de Euler com o passo reduzido (Figura 1). Conforme diminui-se o passo até chegar em $h = 0.65$, o desvio cai para $10^{-6}$, o que torna os resultados indistinguíveis da solução analítica tanto numerica quanto graficamente.

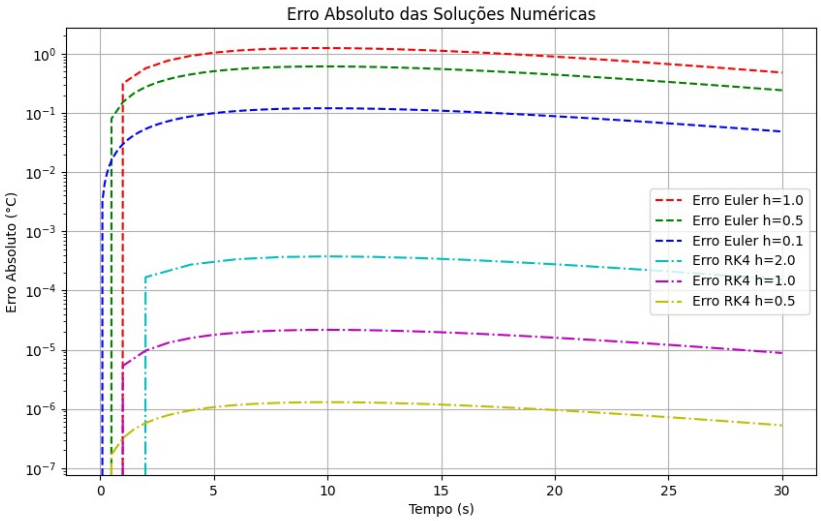



Figura 1: Gráfico de Erros Absolutos

No gráfico mostrado na Figura 2 essa diferença de desempenho pode ser vista, de certa forma. As curvas dos métodos numéricos convergem para a solução analítica à medida em que o passo $h$ diminui. As curvas de Euler com passos maiores se afastam levemente da analítica, enquanto as curvas de RK4 se mantêm muito próximas, mesmo com um $h$ alto. Na Tabela 2 é possível ver que o método RK4 realmente se aproxima muito da solução analítica, o que justifica curvas tão próximas graficamente. Os gráficos na Figura 3 são, na verdade, uma imagem aproximada do primeiro gráfico para mostrar o comportamento das curvas se analisadas mais de perto.

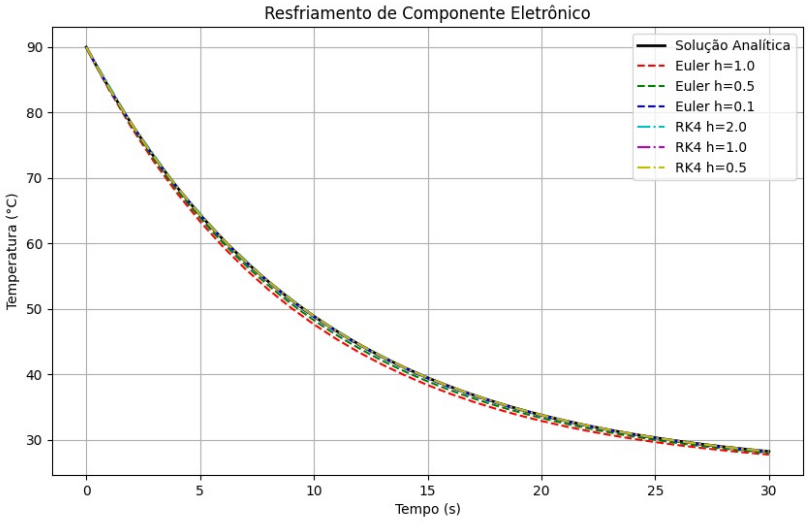
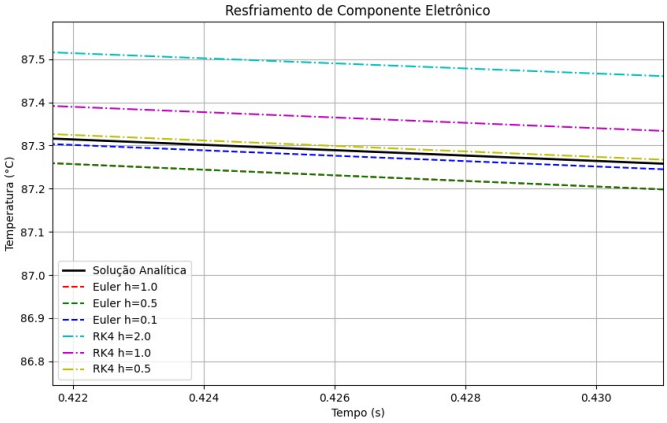
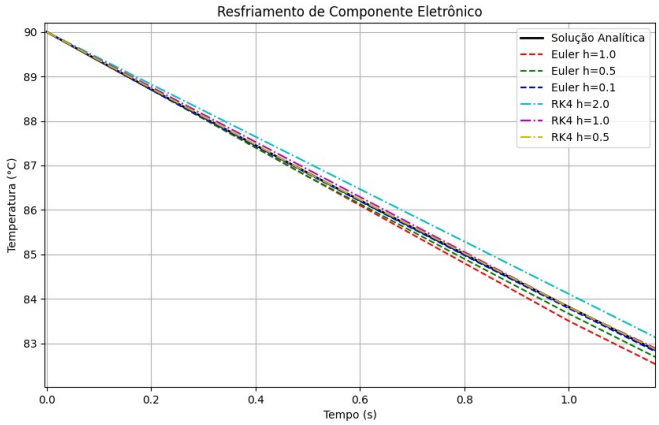



Figura 2: Gráfico Comparativo de Soluções




(a) Zoom 1




(b) Zoom 2

Figura 3: Zoom na Imagem 2

A compração geral entre os métodos é vísivel também na Tabela 3. De maneira geral, o erro do método de Euler se torna proporcional a $h$ enquanto o erro de RK4 é proporcional a $h^4$, o que mostra um ganho exponencial de precisão com a redução do passo em RK4. Reduzir o passo $h$ melhora a precisão de ambos os métodos, mas RK4 é menos sensível a passos grandes e oferece melhor desempenho com menos iterações. Apesar disso, é necessário avaliar custos computacionais para o método de RK4.

| **Método** | **Passo $h$** | **Erro Máximo (%)** | **Nž de Passos (0–30 s)** |
|---|---|---|---|
| Euler | 1.0 | $\sim 2.65\%$ | 30 |
| Euler | 0.65 | $\sim 1.31\%$ | 60 |
| RK4 | 2.0 | $\sim 0.00082\%$ | 15 |
| RK4 | 0.65 | $\sim 0.000001\%$ | 60 |

Tabela 3: Comparação entre métodos numéricos para a equação de resfriamento

# 5 Conclusão

Este trabalho explorou a aplicação da Lei de Resfriamento de Newton para modelar o resfriamento de um componente eletrônico em um ambiente controlado, utilizando a resolução analítica e dois métodos numéricos. A solução analítica, representada pela equação $T(t) = 65 \cdot e^{-0.1t} + 25$ forneceu um resultado exato para referência no momento de analisar a precisão dos métodos numéricos implementados, sendo eles o método de Euler e o método Runge-Kutta de 4ř ordem (RK4).

Os resultados demonstraram que ambos os métodos numéricos aplicados são capazes de aproximar a solução analítica, mas com certas diferenças em termo de precisão versus eficiência. O método de Euler, sendo mais simples e computacionalmente mais leve, se mostrou sensível ao tamanho do passo $h$ aplicado, com erros maiores para passos mais largos. Por outro lado, o método RK4 apresentou precisão superior mesmo com passos maiores, o que destaca sua robustez e eficácia para problemas que exigem certa exatidão.

A análise gráfica e os desvios percentuais calculados evidenciaram que a redução do passo $h$ melhora a precisão de ambos os métodos, mas o RK4 manteve uma convergência estável e mais próxima da solução analítica, mesmo com o aumento do passo. Tal fato reforça a importância da escolha do método numérico adequado, considerando a necessidade de equilíbrio entre precisão e custo computacional.

Em conclusão, este trabalho ilustrou a relevância das equações diferenciais na modelagem de fenômenos reais, como o próprio resfriamento de componentes eletrônicos, e destacou a utilidade dos métodos numéricos quando soluções analíticas são complexas ou invíaveis. Além disso, a aplicação de dois métodos diferentes em um mesmo problema forneceu um melhor entendimento sobre suas vantagens e limitações.

## Referências Bibliográficas

[1] Matplotlib Community, “Matplotlib documentation,” 2024. Acesso em: 28 abr. 2025.

[2] NumPy Developers, “Numpy documentation,” 2024. Acesso em: 28 abr. 2025.

[3] G. van Rossum *et al.*, “Pep 8 – style guide for python code,” 2001. Acesso em: 28 abr. 2025.

[4] P. on Computational Mathematics, “Runge-kutta method,” 2024. Acesso em: 28 abr. 2025.