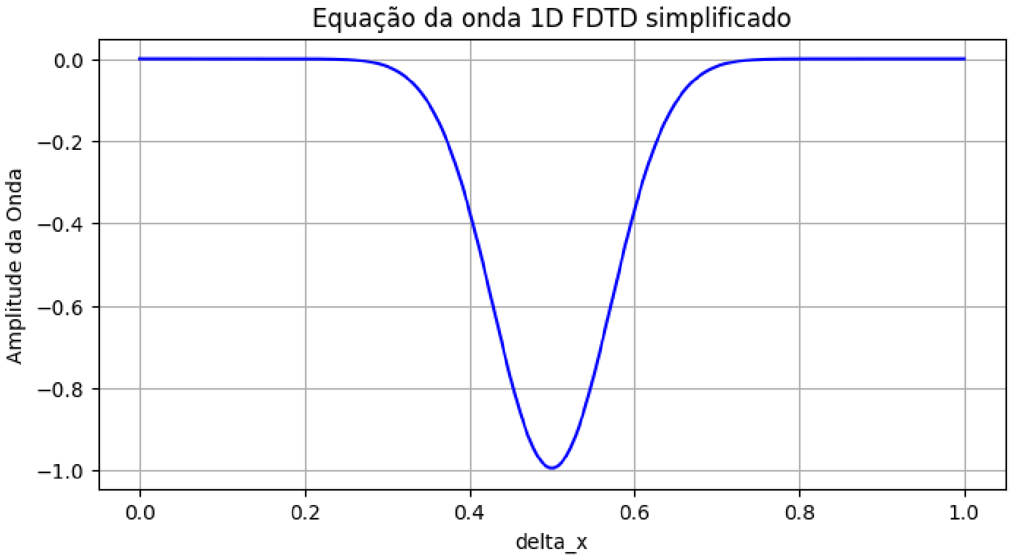
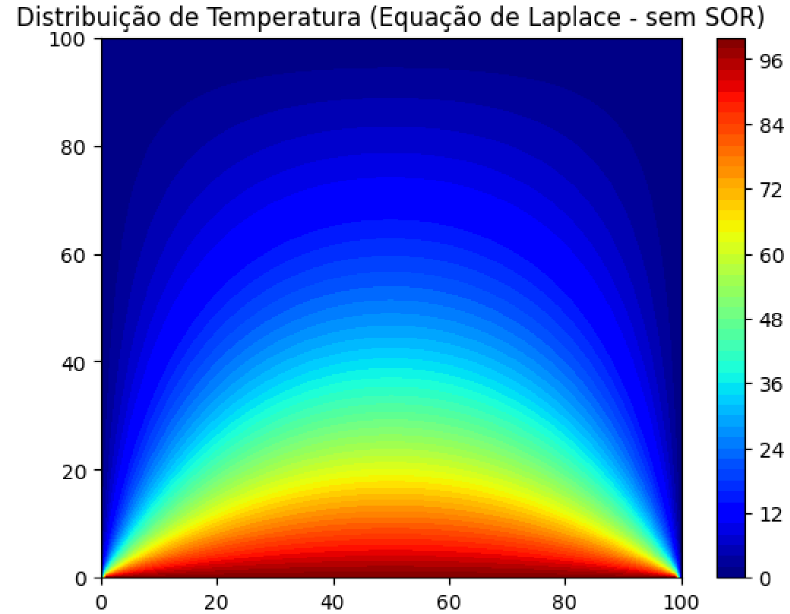
```python
import numpy as np
import math
import matplotlib.pyplot as plt

# Parâmetros físicos e numéricos
comprimento = 1.0
velocidade = 1.0
pontos = 200
dx = comprimento / (pontos - 1)
dt = 0.5 * dx / velocidade
passos_tempo = 400
C = (velocidade * dt / dx) ** 2

# Criação do vetor de posições
x = np.linspace(0, comprimento, pontos)

# Condição inicial (pulso gaussiano no centro)
E_atual = np.exp(-100 * (x - comprimento / 2) ** 2)
E_anterior = E_atual.copy()
E_proximo = np.zeros_like(E_atual)

# =========================================================
# Primeiro passo no tempo (velocidade inicial ZERO)
# Fórmula: E¹(i) = E⁰(i) + 0.5*C*(E⁰(i+1)-2E⁰(i)+E⁰(i-1))
# =========================================================
for i in range(1, pontos - 1):
    E_atual[i] = E_anterior[i] + 0.5 * C * (E_anterior[i + 1] - 2 * E_anterior[i] + E_anterior[i - 1])

# =========================================================
# Loop no tempo
# Fórmula principal:
# Eⁿ⁺¹(i) = 2Eⁿ(i) - Eⁿ⁻¹(i)
#          + C*(Eⁿ(i+1) - 2Eⁿ(i) + Eⁿ(i-1))
# =========================================================
for passo in range(passos_tempo):

    # Cálculo apenas para pontos internos
    for i in range(1, pontos - 1):
        E_proximo[i] = (2 * E_atual[i] - E_anterior[i] +
                        C * (E_atual[i + 1] - 2 * E_atual[i] + E_atual[i - 1]))

    # Condições de contorno (extremidades presas)
    E_proximo[0] = 0.0
    E_proximo[-1] = 0.0

    # Avança no tempo
    E_anterior = E_atual.copy()
    E_atual = E_proximo.copy()

# Gráfico final
plt.figure(figsize=(8, 4))
plt.plot(x, E_atual, color='blue')
plt.title('Equação da onda 1D FDTD simplificado')
plt.xlabel('Posição x [m]')
plt.ylabel('Amplitude da Onda')
plt.grid(True)
plt.show()
```

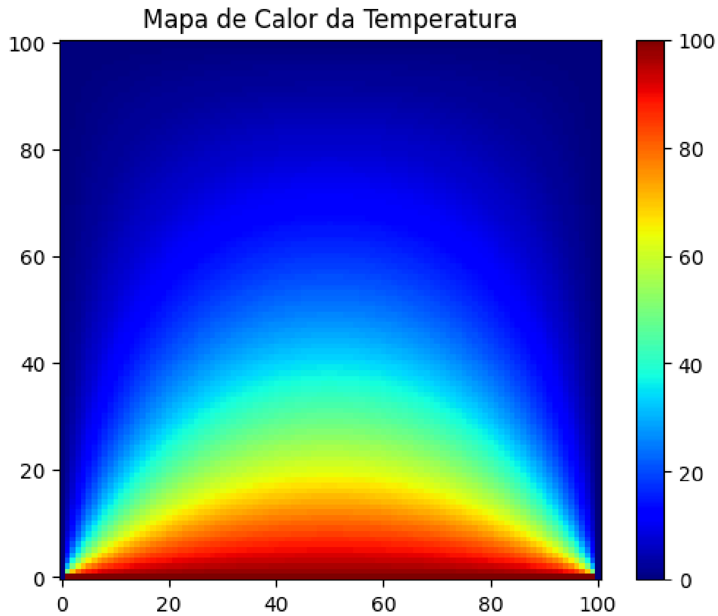



```
import numpy as np
import matplotlib.pyplot as plt
import time

inicio = time.time()

# Tamanho do Grid
N = 100

# Condições de contorno
T_sup = 100
T_dir = 0
T_est = 0
T_inf = 0
erro = 1e-6

# Inicializa o array de temperatura
Temp = np.zeros([N + 1, N + 1], float)
Temp[0, :] = T_sup   # Borda superior
Temp[-1, :] = T_inf  # Borda inferior
Temp[:, 0] = T_est   # Borda esquerda
Temp[:, -1] = T_dir  # Borda direita

# Iteração de Gauss-Seidel
delta = 1.0
while delta > erro:
    delta = 0.0
    for i in range(1, N):
        for j in range(1, N):
            T_old = Temp[i, j]
            Temp[i, j] = 0.25 * (
                Temp[i + 1, j] +
                Temp[i - 1, j] +
                Temp[i, j + 1] +
                Temp[i, j - 1]
            )
            diff = abs(Temp[i, j] - T_old)
            if diff > delta:
                delta = diff

fim = time.time()
print(f"Tempo de execução: {fim - inicio:.4f} segundos")

# Visualização da solução
X, Y = np.meshgrid(np.arange(0, N + 1), np.arange(0, N + 1))
plt.contourf(X, Y, Temp, 50, cmap='jet')
plt.colorbar()
plt.title("Distribuição de Temperatura (Equação de Laplace - sem SOR)")

plt.figure()
plt.imshow(Temp, origin='lower', cmap='jet')
plt.colorbar()
```

```
plt.title("Mapa de Calor da Temperatura")
plt.show()
```

```
Tempo de execução: 242.8173 segundos
```



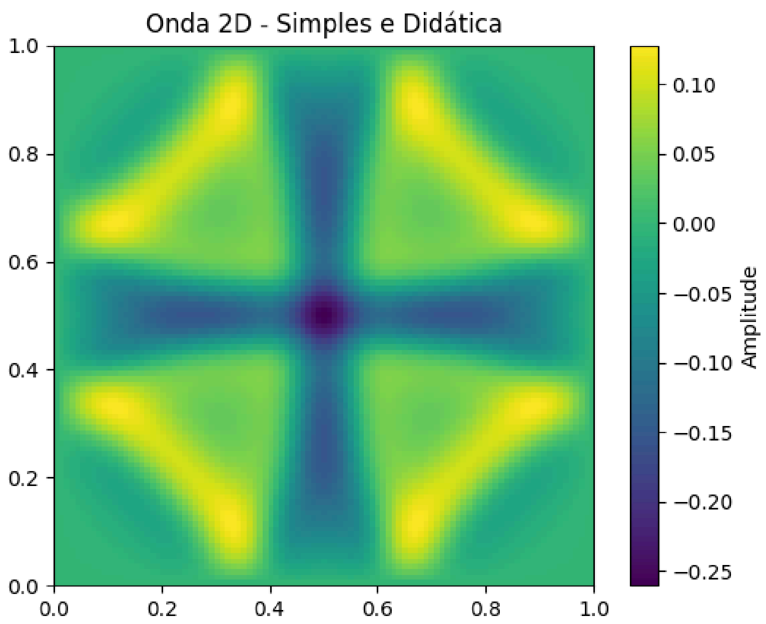


QUESTÃO 02

QUESTÃO 03

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

# Tamanho da região simulada (quadrada)
L = 1.0

# Quantidade de pontos no eixo x e y
N = 101

# Espaço entre os pontos (malha)
dx = L / (N - 1)
dy = dx

# Velocidade da onda
v = 1.0

# Passo de tempo (garante estabilidade numérica)
```

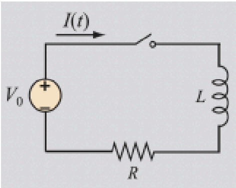
```
dt = 0.5 * dx / v

# Constantes que aparecem na fórmula numérica
Cx = (v * dt / dx) ** 2
Cy = Cx  # como dx = dy, é igual

# Matrizes para os 3 instantes no tempo
onda_prev = np.zeros((N, N))   # tempo t - dt
onda_atual = np.zeros((N, N))  # tempo t
onda_next = np.zeros((N, N))   # tempo t + dt

# Coordenadas
x = np.linspace(0, L, N)
y = np.linspace(0, L, N)

# Pulso inicial no centro
for i in range(N):
    for j in range(N):
        onda_prev[i,j] = np.exp(-200*((x[i]-0.5)**2 + (y[j]-0.5)**2))
        onda_atual[i,j] = onda_prev[i,j]

# Evolução da onda no tempo
for passo in range(200):

    for i in range(1, N-1):
        for j in range(1, N-1):

            onda_next[i,j] = (
                2*onda_atual[i,j] - onda_prev[i,j]
                + Cx * (onda_atual[i+1,j] - 2*onda_atual[i,j] + onda_atual[i-1,j])
                + Cy * (onda_atual[i,j+1] - 2*onda_atual[i,j] + onda_atual[i,j-1])
            )

    # Bordas fixas (parede onde a onda bate e volta)
    for i in range(N):
        onda_next[i,0] = 0
        onda_next[i,N-1] = 0
    for j in range(N):
        onda_next[0,j] = 0
        onda_next[N-1,j] = 0

    # Avança no tempo
    onda_prev = onda_atual.copy()
    onda_atual = onda_next.copy()

# Gráfico final
plt.imshow(onda_atual.T, origin='lower', cmap='viridis', extent=[0,L,0,L])
plt.colorbar(label="Amplitude")
plt.title("Onda 2D - Simples e Didática")
plt.show()
```



LISTA 02

QUESTÃO 01

Questão 1. *Um indutor e um resistor não-linear de resistência* $R = 500 + 250I^2\Omega$ *estão conectados em série com uma fonte de tensão CC e uma chave.*

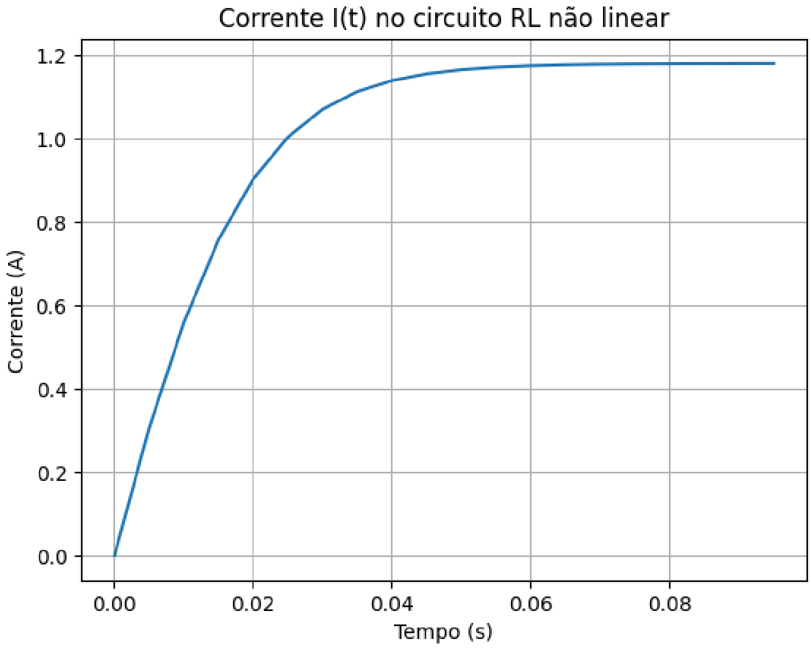


A chave está inicialmente aberta, sendo então fechada no tempo $t = 0$. *A corrente* I *no circuito para* $t > 0$ *é determinada a partir da solução da equação*

$$\frac{dI}{dt} = \frac{V_0}{L} - \frac{R}{L}I$$

Para $V_0 = 1000\ V$ *e* $L = 15H$, *determine e trace a corrente em função do tempo em* $0 \geqslant t \geqslant 0,1\ s$. *Use 0,005 como passo de integração.*

```
import matplotlib.pyplot as plt

# Constantes
t_inicial = 0.0
t_final = 0.1
passo = 0.005
V0 = 1000.0
L = 15.0

# Função da EDO
def f_corrente(t, I):
    R = 500.0 + 250.0 * I**2
    return (V0 / L) - (R / L) * I

# Método de Runge-Kutta de 4ª ordem
t = t_inicial
I = 0.0
tempos = []
correntes = []

while t <= t_final:
    tempos.append(t)
    correntes.append(I)
    k1 = f_corrente(t, I)
    k2 = f_corrente(t + passo/2, I + (passo/2)*k1)
    k3 = f_corrente(t + passo/2, I + (passo/2)*k2)
    k4 = f_corrente(t + passo, I + passo*k3)
    I = I + (passo/6)*(k1 + 2*k2 + 2*k3 + k4)
    t = t + passo



plt.plot(tempos, correntes)
plt.title("Corrente I(t) no circuito RL não linear")
plt.xlabel("Tempo (s)")
plt.ylabel("Corrente (A)")
plt.grid()
plt.show()
```



QUESTÃO 02

Questão 2. *Resolva o seguinte problema de valor inicial*

$$\frac{dy}{dx} = yx^2 - 1,1y$$

em que $y(0) = 1$ *no intervalo de* $x \in [0, 2]$. *Use o método de RK de primeira e segunda ordem com* h *= 0,5 e 0,25. Calcule o erro de truncamento total.*

```
# Dados da questão:
#y(0) = 1, onde x0 = 0 e y0 = 1
# Intervalo entre 0 até 2
# Passos h = 0.5 e h = 0.25

# y' = y*(x²) - 1.1*y

def f(x, y):
    return y * x * x - 1.1 * y

# Método de Euler (RK1)
def rk1(h):
    x = 0.0
    y = 1.0
    while x < 2.0:
        y = y + h * f(x, y)
        x = x + h
    return y

# Método de RK2 (Heun sem corretor)
def rk2(h):
    x = 0.0
    y = 1.0

    while x < 2.0:
        k1 = f(x, y)
        k2 = f(x + h, y + h * k1)
        y = y + 0.5*(k1 + k2)*h
        x = x + h
    return y

# Cálculos para h = 0.5 e h = 0.25
for passo in [0.5, 0.25]:

    y_rk1 = rk1(passo)
    y_rk2 = rk2(passo)
```

```
    erro_truncamento = abs(y_rk1 - y_rk2)


    print("Passo h =", passo)
    print("  RK (1ª ordem)  = ", y_rk1)
    print("  RK2 (2ª ordem)  = ", y_rk2)
    print("  Erro total ≈ ", erro_truncamento)
    print('------------------------------')
```

```
Passo h = 0.5
  RK (1ª ordem)  =  0.38715468749999987
  RK2 (2ª ordem)  =  1.4754070184938046
  Erro total ≈  1.0882523309938048
------------------------------
Passo h = 0.25
  RK (1ª ordem)  =  0.7641088317523717
  RK2 (2ª ordem)  =  1.5630651931973547
  Erro total ≈  0.798956361444983
------------------------------
```

QUESTÃO 03

Questão 3. *Considere a EDO de primeira ordem a seguir*

$$\frac{dy}{dx} = yx - x^3$$

com $y(0) = 1$ *e* $I = [0, 1.8]$. *Resolva a equação manualmente usando o método de Runge-Kutta de segunda ordem pela tecnica de Heun sem iteração, ponto médio e Ralston, com 3 tamanhos de passos distintos. Calcule o erro de truncamento total em cada caso.*

```
# dados da questão:
# x0 = 0 e y0 e intervalo de 0 até 1.8

import math

# y'
def f(x, y):
    return y * x - x**3

def heun(h):
  x = 0.0
  y = 1.0

  while x < 1.8:
    k1 = f(x, y)
    k2 = f(x + h, y + h)
    y = y + 0.5 * (k1 + k2) * h
    x = x + h

    return y

def ralston(h):
  x = 0.0
  y = 1.0

  while x < 1.8:
    k1 = f(x, y)
    k2 = f(x + 0.75*h, y + k1*0.75*h)

    y = y + (h/3.0)*(k1 + 2*k2)
    x = x + h

    return y

def ponto_medio(h):
  x = 0.0
  y = 1.0

  while x < 1.8:
    k1 = f(x, y)
```

```
        k2 = f(x + (h/2), y + (k1*h/2))
        y = y + h*k2
        x = x + h

        return y


    passos = [0.2, 0.6, 0.7]
    valor_heun = []
    valor_ralston = []
    valor_ponto_medio = []

    for h in passos:

      valor_heun.append(heun(h))
      valor_ralston.append(ralston(h))
      valor_ponto_medio.append(ponto_medio(h))

    for i in range (len(passos)):

      print(f'\nh = {passos[i]}')
      print(f'Heun: y(1.8): {valor_heun[i]}')
      print(f'Ralston: y(1.8): {valor_ralston[i]}')
      print(f'Ponto Médio: y(1.8): {valor_ponto_medio[i]}')
      print('\n')

      if i > 0:
        print('Erros aproximados: ')
        print('-----------------------------------')
        print(' heun: ', (abs(valor_heun[i]) - abs(valor_heun[i-1])))
        print(' ralston: ', abs(valor_ralston[i]) - abs(valor_ralston[i-1]))
        print(' ponto medio: ', abs(valor_ponto_medio[i]) - abs(valor_ponto_medio[i-1]))
```

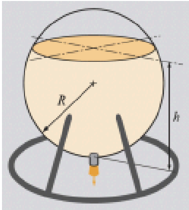
```
h = 0.2
Heun: y(1.8): 1.0232
Ralston: y(1.8): 1.01955
Ponto Médio: y(1.8): 1.0198



h = 0.6
Heun: y(1.8): 1.2232
Ralston: y(1.8): 1.14355
Ponto Médio: y(1.8): 1.1638


Erros aproximados:
-----------------------------------
 heun:  0.19999999999999996
 ralston:  0.12400000000000011
 ponto medio:  0.1439999999999999

h = 0.7
Heun: y(1.8): 1.29645
Ralston: y(1.8): 1.177471875
Ponto Médio: y(1.8): 1.2149874999999999


Erros aproximados:
-----------------------------------
 heun:  0.07325000000000004
 ralston:  0.03392187499999988
 ponto medio:  0.05118749999999994
```

Clique duas vezes (ou pressione "Enter") para editar

```
Comece a programar ou gere código com IA.
```

QUESTÃO 04

Questão 4. *Um tanque esférico de raio $R = 4m$ é esvaziado por meio de um pequeno buraco circular de raio $r = 0,02$ m localizado no fundo.*



1

O topo do tanque está aberto. O nível d'água instantâneo no tanque, h (medido a partir do fundo do tanque, no dreno), pode ser determinado a partir da solução da seguinte EDO

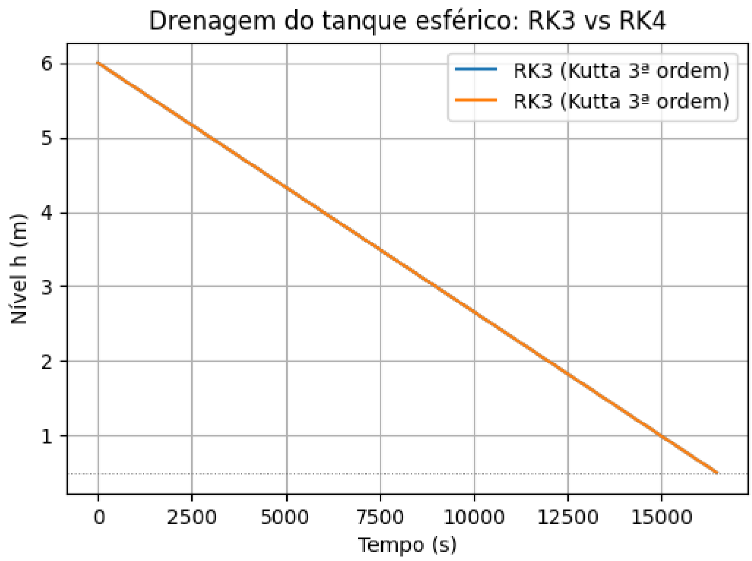
$$\frac{dh}{dt} = -\frac{r^2\sqrt{2gh}}{2hR - h^2}$$

onde $g = 9,81 m/s^2$. Se o nível d'água inicial em $t = 0$ é $h = 6$ m, determine o tempo necessário para drenar o tanque até um nível de 0,5 m. Use o método de Runge-Kutta de terceira e quarta ordem e compare os resultados graficamente.

```
import math
import matplotlib.pyplot as plt

# dados da questão
# t = x
# h = y

# valores fornecidos na questão
R = 4.0
r = 0.02
g = 9.81
y0 = 6.0
y_final = 0.5

def f(x, y):
  numerador = (r**2)*(math.sqrt(2*g*h))
  denominador = (2*h*R) - (h**2)

  valor = - ( numerador / denominador )

  return valor

def rk3(h):
    x = 0.0
    y = y0
    tempos = []
    niveis = []

    while y > y_final:
      tempos.append(x)
      niveis.append(y)
      k1 = f(x, y)
      k2 = f(x + (h/2), y + (k1 * (h/2)))
      k3 = f(x + h, y - (k1 * h) + (2 * k2 * h))

      y = y + (h/6) * (k1 + 4 * k2 + k3)
      x = x + h

    return tempos, niveis, x


def rk4(h):
  x = 0.0
  y = y0
  tempos = []
  niveis = []
```

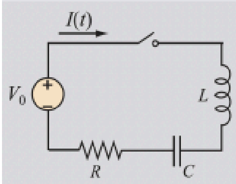
```
    while y > y_final:
      tempos.append(x)
      niveis.append(y)

      k1 = f(x, y)
      k2 = f(x + (h/2), y + (k1 * (h/2)))
      k3 = f(x + (h/2), y + (k2 * (h/2)))
      k4 = f(x + h, y + (k3 * h))

      y = y + (h/6) * (k1 + 2 * k2 + 2 * k3 + k4)
      x = x + h

    return tempos, niveis, x

h = 0.5

tempos_rk3, niveis_rk3, x_rk3 = rk3(h)
tempos_rk4, niveis_rk4, x_rk4 = rk4(h)

print("Passo usado: ", h)
print("Tempo para drenar de {:.2f} m até {:.2f} m:".format(y0, y_final))
print("  RK3 (Kutta 3ª ordem): {:.6f} s".format(x_rk3))
print("  RK4 (Kutta 4ª ordem): {:.6f} s".format(x_rk4))


plt.figure(figsize=(6,4))
plt.plot(tempos_rk3, niveis_rk3, label='RK3 (Kutta 3ª ordem)')
plt.plot(tempos_rk4, niveis_rk4, label='RK3 (Kutta 3ª ordem)')
plt.axhline(y=y_final, color='gray', linewidth=0.7, linestyle=':')
plt.xlabel('Tempo (s)')
plt.ylabel('Nível h (m)')
plt.title('Drenagem do tanque esférico: RK3 vs RK4')
plt.legend()
plt.grid(True)
plt.show()
```

```
Passo usado:  0.5
Tempo para drenar de 6.00 m até 0.50 m:
  RK3 (Kutta 3ª ordem): 16463.000000 s
  RK4 (Kutta 4ª ordem): 16463.000000 s
```



QUESTÃO 05

Questão 5. *Um capacitor de $C = 4,2\mu F$ é colocado em série.*



Conforme mostrado na figura, o circuito contém uma fonte de tensão CC, $V_0 = 1000$ V, um indutor de $L = 15$ H e uma resistência não-linear $R = R_0 + R_1 I^2$ Ω, onde $R_0 = 500$ Ω e $R_1 = 250$ Ω$/A^2$. A chave está inicialmente aberta, sendo então fechada no tempo t = 0. A carga Q no capacitor em t ¿ 0 é determinada a partir da solução da equação

$$\frac{d^2Q}{dt^2} + \frac{R_0}{L}\frac{dQ}{dt} + \frac{R_1}{L}\left(\frac{dQ}{dt}\right)^3 + \frac{Q}{LC} = \frac{V_0}{L}$$

Inicialmente, $Q = 0$ e $\frac{dQ}{dt} = 0$.

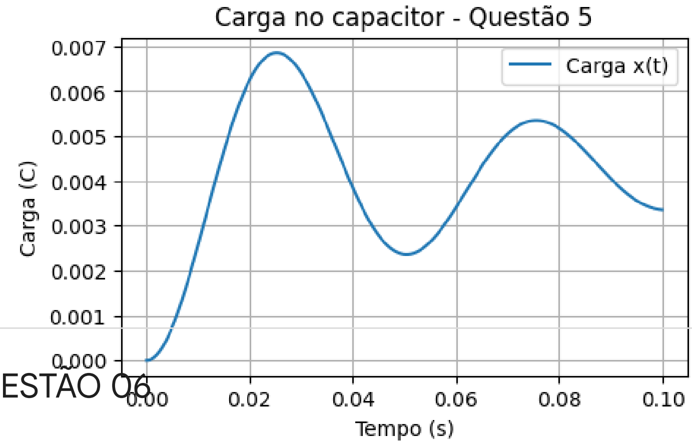
a) Reduza a EDO de segunda ordem em um sistema de duas EDOs de primeira ordem e determine a carga Q em função do tempo em resolvendo o sistema com o método de Runge-Kutta de quarta ordem.

b) Use os resultados obtidos na letra a) para traçar um gráfico com a corrente no circuito. A corrente é dada pela derivada temporal da carga, $I = \frac{dQ}{dt}$.

```
import math
import matplotlib.pyplot as plt

# dados da questão
# Q = x
# dQ/dt = y

# valores fornecidos na questão
C = 4.2e-6
V0 = 1000
L = 15.0
R0 = 500
R1 = 250
x = 0.0
y = 0.0


def f1(x, y):
  # dQ/dt = y
  return y

def f2(x, y):
  termo1 = V0 / L
  termo2 = (R0 / L) * y
  termo3 = (R1 / L) * (y ** 3)
  termo4 = x / (L * C)

  return termo1 - termo2 - termo3 - termo4


def rk4(h):

  x = 0.0
  y = 0.0
  tempo = 0.0
  tempos = []
  valores_x = []
  valores_y = []

  while tempo <= 0.1:
    tempos.append(tempo)
    valores_x.append(x)
    valores_y.append(y)
```

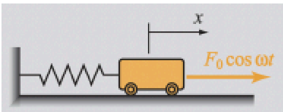
```
        k1x = f1(x, y)
        k2x = f1(x + (h/2), y + (k1x * (h/2)))
        k3x = f1(x + (h/2), y + (k2x * (h/2)))
        k4x = f1(x + h, y + (k3x * h))



        k1y = f2(x, y)
        k2y = f2(x + (h/2), y + (k1y * (h/2)))
        k3y = f2(x + (h/2), y + (k2y * (h/2)))
        k4y = f2(x + h, y + (k3y * h))

        x = x + (h/6) * (k1x + 2 * k2x + 2 * k3x + k4x)
        y = y + (h/6) * (k1y + 2 * k2y + 2 * k3y + k4y)

        tempo = tempo + h

      return tempos, valores_x, valores_y

    h = 0.0001

    tempos, cargas, correntes = rk4(h)

    print('\n letra a) \n')
    print("Passo de integração =", h)
    print("Carga final (x) =", cargas[-1], "C")
    print("Corrente final (y) =", correntes[-1], "A")


    plt.figure(figsize=(5,3))
    plt.plot(tempos, cargas, label="Carga x(t)")
    plt.xlabel("Tempo (s)")
    plt.ylabel("Carga (C)")
    plt.title("Carga no capacitor - Questão 5")
    plt.grid(True)
    plt.legend()
    plt.show()

    print('\n letra b) \n')

    plt.figure(figsize=(5,3))
    plt.plot(tempos, correntes, label="Corrente y(t)", color="orange")
    plt.xlabel("Tempo (s)")
    plt.ylabel("Corrente (A)")
    plt.title("Corrente no circuito - Questão 5")
    plt.grid(True)
    plt.legend()
    plt.show()
```

```
letra a)

Passo de integração = 0.0001
Carga final (x) = 0.0033532588980488146 C
Corrente final (y) = -0.009270532665919474 A
```



QUESTÃO 06

Questão 6. *Considere a vibração forçada do sistema massa-mola mostrado na figura.*



A posição x da massa em função do tempo é dada pela solução da equação

$$\frac{d^2x}{dt^2} = -\frac{k}{m}x + \frac{F_0}{m}\cos\omega t$$

onde $m = 2$ kg é a massa, $k = 800$ N/m é a constante da mola, $F_0 = 50$ N é a amplitude da força harmônica aplicada e $\omega = 3$ rad/s é a frequência dessa força. As condições iniciais são $x(0) = 0,1$ m e $x'(0) = 0,1$ m/s. Resolva a EDO em $0 \leqslant t \leqslant 10$ s e trace x e em função de t. Use um passo de integração de 0,01 s.

Tempo (s)

```
import math
import matplotlib.pyplot as plt

# dados da questão
# x = posição
# y = velocidade

# valores fornecidos na questão
m = 2.0
k = 800.0
f0 = 50.0
w = 3.0


def f1(x, y):

  return y

def f2(x, y, tempo):
  termo1 = - (k/m) * x
  termo2 = (f0/m) * math.cos(w * tempo)

  return termo1 + termo2


def rk4(h):

  x = 0.1
  y = 0.1
  tempo = 0.0
  tempos = []
  valores_x = []
  valores_y = []
```

```
    while tempo <= 10:
      tempos.append(tempo)
      valores_x.append(x)
      valores_y.append(y)


      k1x = f1(x, y)
      k1y = f2(x, y, tempo)

      k2x = f1(x + (h/2) * k1x, y + (h/2) * k1y)
      k2y = f2(x + (h/2) * k1x, y + (h/2) * k1y, tempo + h/2)

      k3x = f1(x + (h/2) * k2x, y + (h/2) * k2y)
      k3y = f2(x + (h/2) * k2x, y + (h/2) * k2y, tempo + h/2)

      k4x = f1(x + h * k3x, y + h * k3y)
      k4y = f2(x + h * k3x, y + h * k3y, tempo + h)

      x = x + (h/6)*(k1x + 2*k2x + 2*k3x + k4x)
      y = y + (h/6)*(k1y + 2*k2y + 2*k3y + k4y)


      tempo = tempo + h

    return tempos, valores_x, valores_y

  h = 0.01

  tempos, posicoes, velocidades = rk4(h)


  print("Passo de integração =", h)
  print("posição final (x) =", posicoes[-1], "m")
  print("Velocidade final (y) =", velocidades[-1], "m/s")

  print('\n')


  plt.figure(figsize=(6,4))
  plt.plot(tempos, posicoes, label="Posição x(t)")
  plt.xlabel("Tempo (s)")
  plt.ylabel("Posição (m)")
  plt.title("Posição no tempo - Questão 6")
  plt.grid(True)
  plt.legend()
  plt.show()
```

```
Passo de integração = 0.01
posição final (x) = 0.022969762531938123 m
Velocidade final (y) = 0.8684761257806122 m/s
```