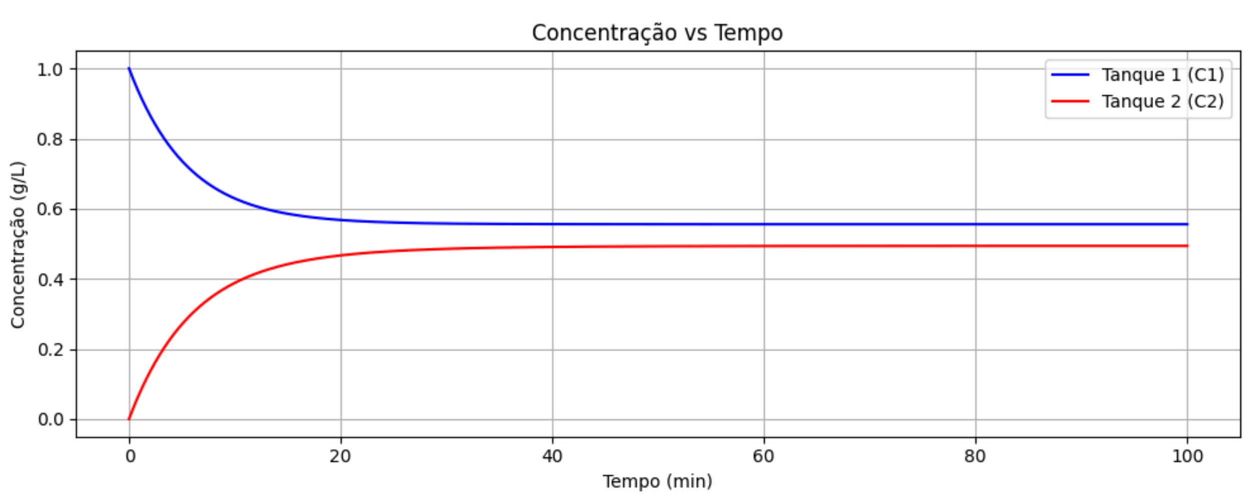
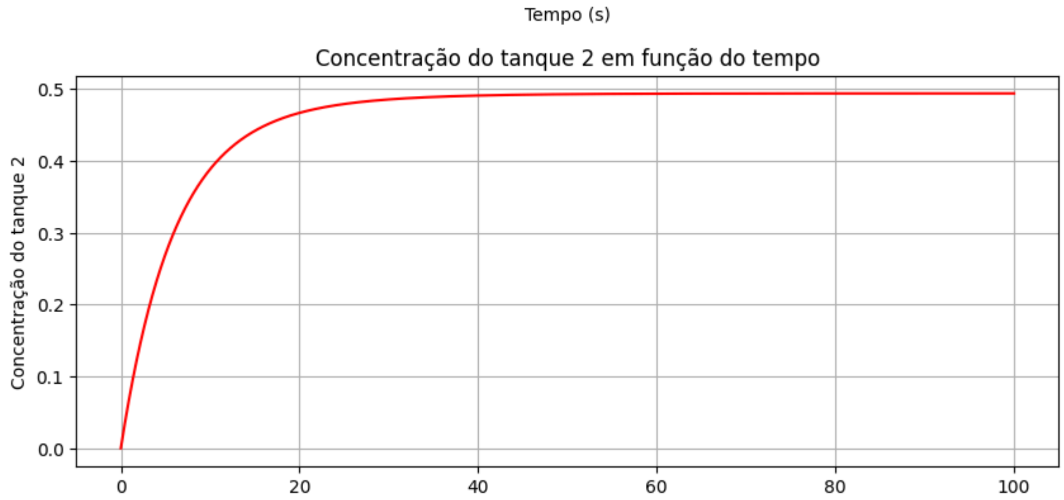
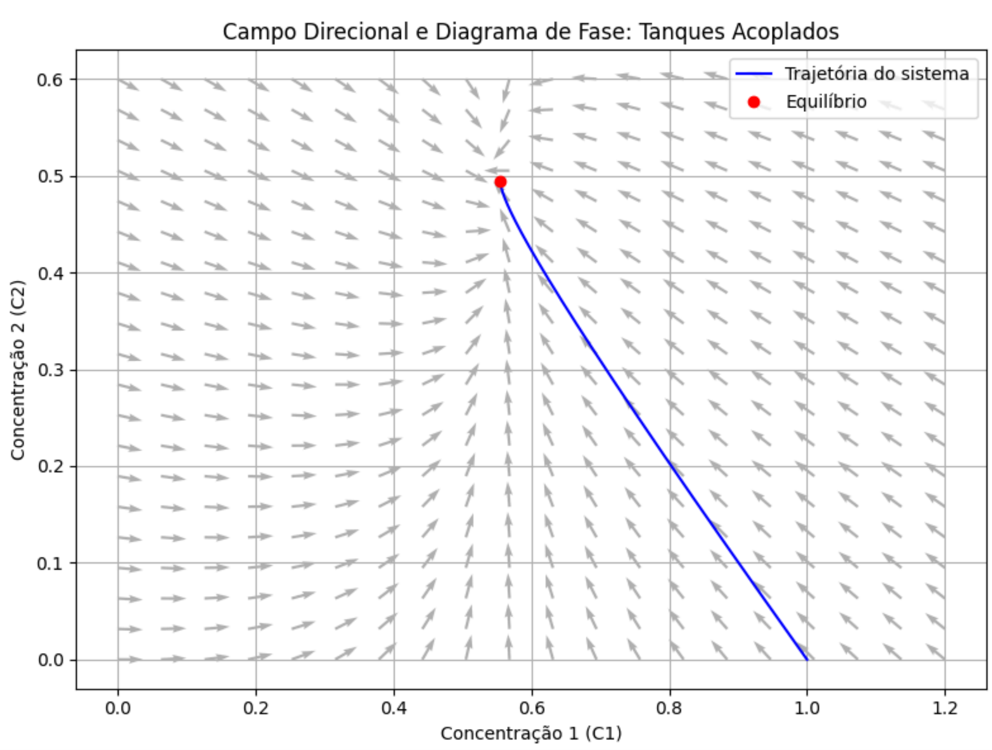
```
import numpy as np
import matplotlib.pyplot as plt

# Parametros
V1 = V2 = 100  # Volume (L)
Q_in = 10      # Fluxo de entrada (L/min)
Q_12 = 8       # Fluxo entre os tanques (L/min)
Q_out = 1      # Fluxo de saida (L/min)
C_in = 1       # Concentração de entrada (g/L)

# Tempo
dt = 0.001      # Variação de tempo (min)
T = 100    # Tempo total (min)
num_passos = int(T/dt)

# Inicializar
t = np.linspace(0, T, num_passos)
C1 = np.zeros(num_passos)
C2 = np.zeros(num_passos)
C1[0] = C_in  # Condição inicial
C2[0] = 0

# Método de Euler
for i in range(num_passos - 1):
    # Derivadas
    dC1_dt = (Q_in/V1) * (C_in - C1[i]) - (Q_12/V1) * C1[i]
    dC2_dt = (Q_12/V2) * (C1[i] - C2[i]) - (Q_out/V2) * C2[i]

    # Método de Euler com iterações
    C1[i+1] = C1[i] + dC1_dt * dt
    C2[i+1] = C2[i] + dC2_dt * dt

# Gráfico
plt.figure(figsize=(10, 4))
plt.plot(t, C1, 'b-', label='Tanque 1 (C1)')
plt.plot(t, C2, 'r-', label='Tanque 2 (C2)')
plt.xlabel('Tempo (min)')
plt.ylabel('Concentração (g/L)')
plt.title('Concentração vs Tempo')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()

# Resultados obtidos
plt.figure(figsize=(10, 4))

plt.plot(t, C1, 'b')
plt.title('Concentração do tanque 1 em função do tempo')
plt.xlabel('Tempo (s)')
plt.ylabel('Concentração do tanque 1')
plt.grid()

plt.show()

plt.figure(figsize=(10, 4))
plt.plot(t, C2, 'r')
plt.title('Concentração do tanque 2 em função do tempo')
plt.xlabel('Tempo (s)')
plt.ylabel('Concentração do tanque 2')
plt.grid()
plt.show()
```





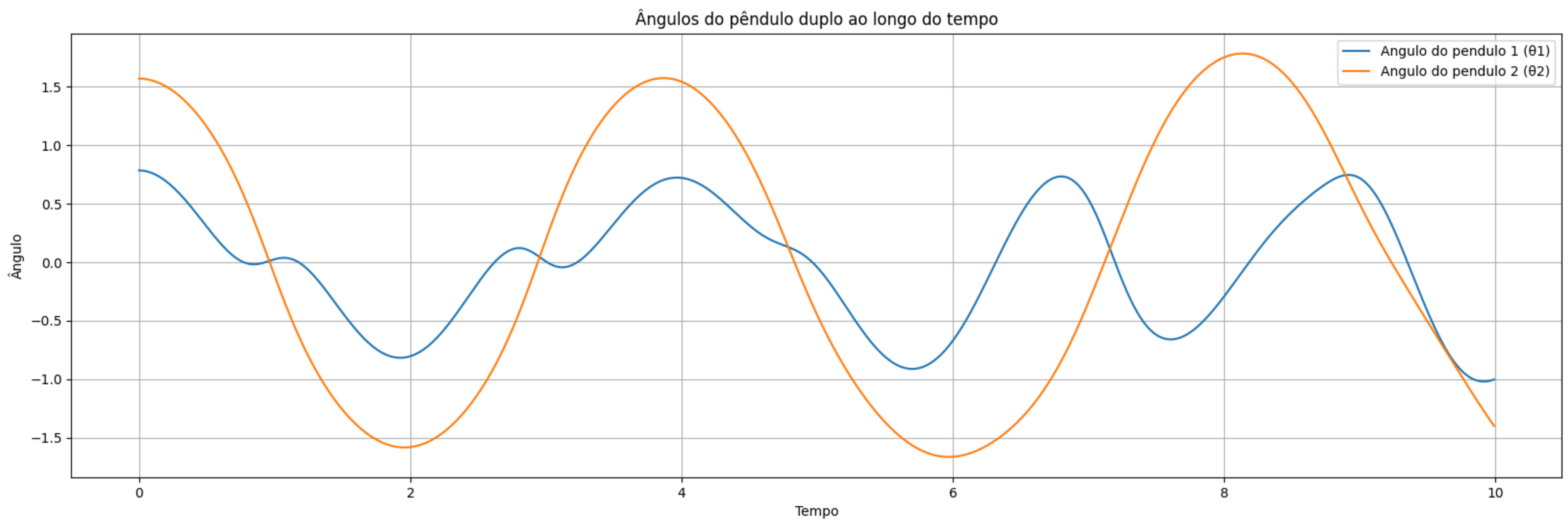

```
import numpy as np
import matplotlib.pyplot as plt

# Parâmetros do sistema
V1 = V2 = 100
Q_in = 10
Q_12 = 8
```

```
Q_out = 1
C_in = 1

# Criar uma grade (grid) de pontos no espaço de fases
C1_vals = np.linspace(0, 1.2, 20)
C2_vals = np.linspace(0, 0.6, 20)
C1_grid, C2_grid = np.meshgrid(C1_vals, C2_vals)

# Calcular as derivadas em cada ponto do grid
dC1_dt = (Q_in/V1) * (C_in - C1_grid) - (Q_12/V1) * C1_grid
dC2_dt = (Q_12/V2) * (C1_grid - C2_grid) - (Q_out/V2) * C2_grid

# Normalizar os vetores para melhor visualização
magnitude = np.sqrt(dC1_dt**2 + dC2_dt**2)
dC1_dt_norm = dC1_dt / magnitude
dC2_dt_norm = dC2_dt / magnitude

# Plotar o campo vetorial
plt.figure(figsize=(8,6))
plt.quiver(C1_grid, C2_grid, dC1_dt_norm, dC2_dt_norm, color='gray', alpha=0.6)

# Simulação da trajetória (igual à sua)
dt = 0.0001
T = 100
num_passos = int(T/dt)
t = np.linspace(0, T, num_passos)
C1 = np.zeros(num_passos)
C2 = np.zeros(num_passos)
C1[0] = C_in
C2[0] = 0

for i in range(num_passos - 1):
    dC1_dt_i = (Q_in/V1)*(C_in - C1[i]) - (Q_12/V1)*C1[i]
    dC2_dt_i = (Q_12/V2)*(C1[i] - C2[i]) - (Q_out/V2)*C2[i]
    C1[i+1] = C1[i] + dC1_dt_i * dt
    C2[i+1] = C2[i] + dC2_dt_i * dt

# Adicionar trajetória real
plt.plot(C1, C2, color='blue', label='Trajetória do sistema')
plt.plot(C1[-1], C2[-1], 'ro', label='Equilíbrio')

# Ajustes finais
plt.xlabel('Concentração 1 (C1)')
plt.ylabel('Concentração 2 (C2)')
plt.title('Campo Direcional e Diagrama de Fase: Tanques Acoplados')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
```

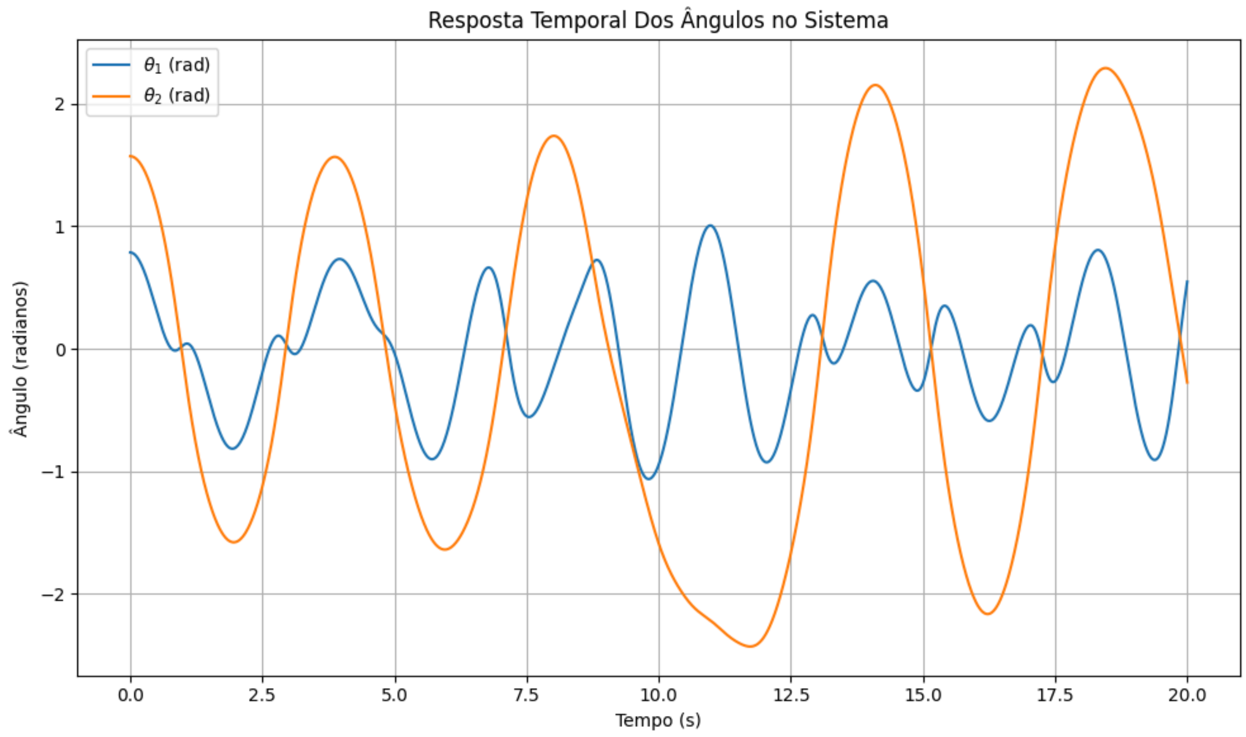


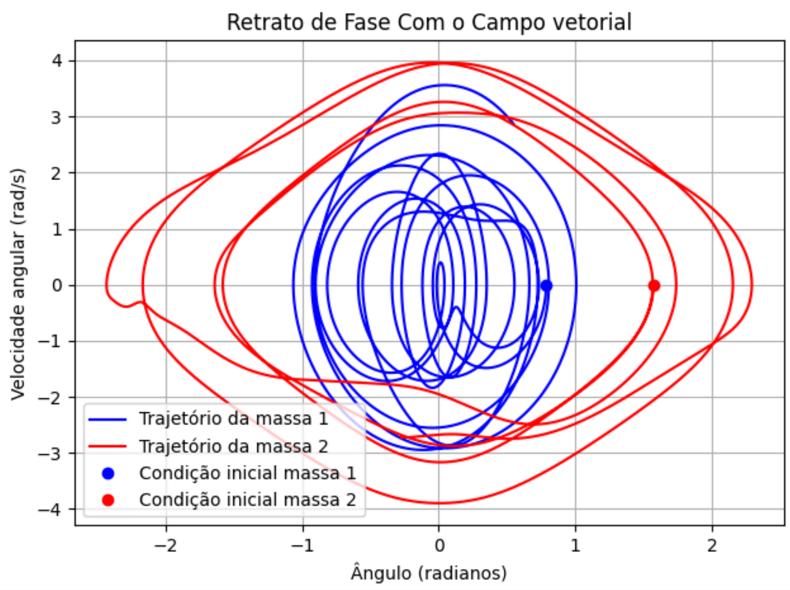

```
#Questão 2
import numpy as np
import matplotlib.pyplot as plt

# Parametros
m1 = 1.0  # Massa do primeiro pendulo
m2 = 1.0  # Massa do segundo pendulo
l1 = 1.0  # Comprimento do primeiro pendulo
l2 = 1.0  # Comprimento do segundo pendulo
g = 9.81  # Aceleração da gravidade

# Tempo
dt = 0.01  # Variação de tempo
t_end = 10  # Tempo total
time = np.arange(0, t_end, dt)

# Initial conditions
theta1 = np.pi/4  # Angulo inicial do pendulo 1
theta2 = np.pi/2 # Angulo inicial do pendulo 2
omega1 = 0 # Velocidade angular inicial do pendulo 1
omega2 = 0  # Velocidade angular inicial do pendulo 2

# Vetores para armazenar o angulo e a velocidade
theta1_arr = np.zeros(len(time))
theta2_arr = np.zeros(len(time))
omega1_arr = np.zeros(len(time))
omega2_arr = np.zeros(len(time))

# Metodo de Euler
for i in range(len(time)):
    theta1_arr[i] = theta1
    theta2_arr[i] = theta2
    omega1_arr[i] = omega1
    omega2_arr[i] = omega2

    # Equação de movimento
    delta_theta = theta2 - theta1
    denom1 = (m1 + m2) * l1 - m2 * l1 * np.cos(delta_theta) * np.cos(delta_theta)
    denom2 = (l2/l1) * denom1

    alpha1 = (-m2 * l1 * omega1**2 * np.sin(delta_theta) * np.cos(delta_theta) +
              m2 * g * np.sin(theta2) * np.cos(delta_theta) +
              m2 * l2 * omega2**2 * np.sin(delta_theta) -
              (m1 + m2) * g * np.sin(theta1)) / denom1

    alpha2 = (l1 * omega1**2 * np.sin(delta_theta) - g * np.sin(theta2) +
              g * np.sin(theta1) * np.cos(delta_theta)) / denom2

    # Atualizar o angulo e a velocidade
    omega1 += alpha1 * dt
    omega2 += alpha2 * dt
    theta1 += omega1 * dt
```

```
    theta2 += omega2 * dt

# Gráfico
plt.figure(figsize=(20, 6))
plt.plot(time, theta1_arr, label='Angulo do pendulo 1 (θ1)')
plt.plot(time, theta2_arr, label='Angulo do pendulo 2 (θ2)')
plt.title('Ângulos do pêndulo duplo ao longo do tempo')
plt.xlabel('Tempo')
plt.ylabel('Ângulo')
plt.legend()
plt.grid()
plt.show()
```

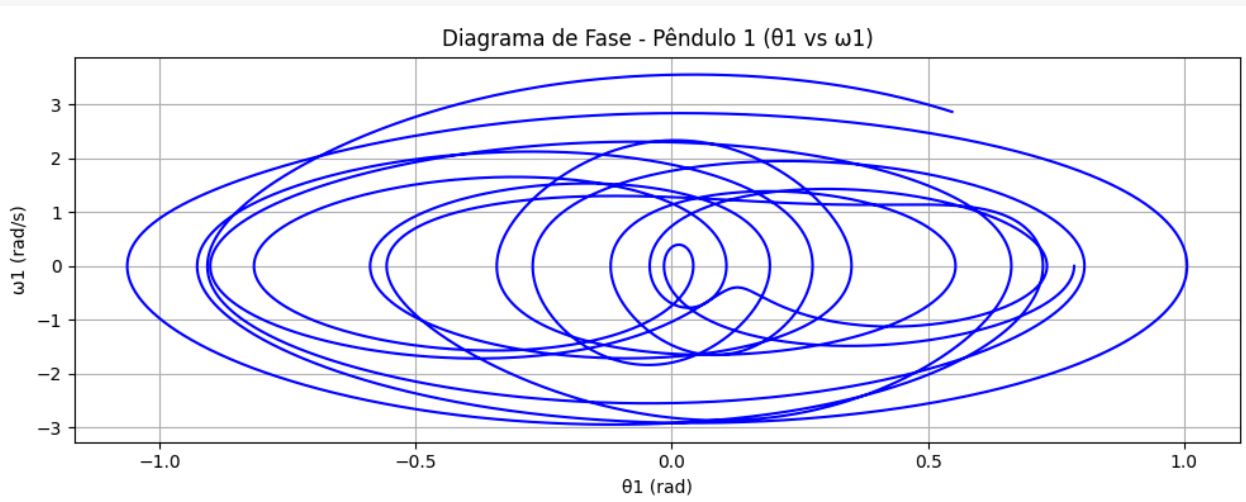


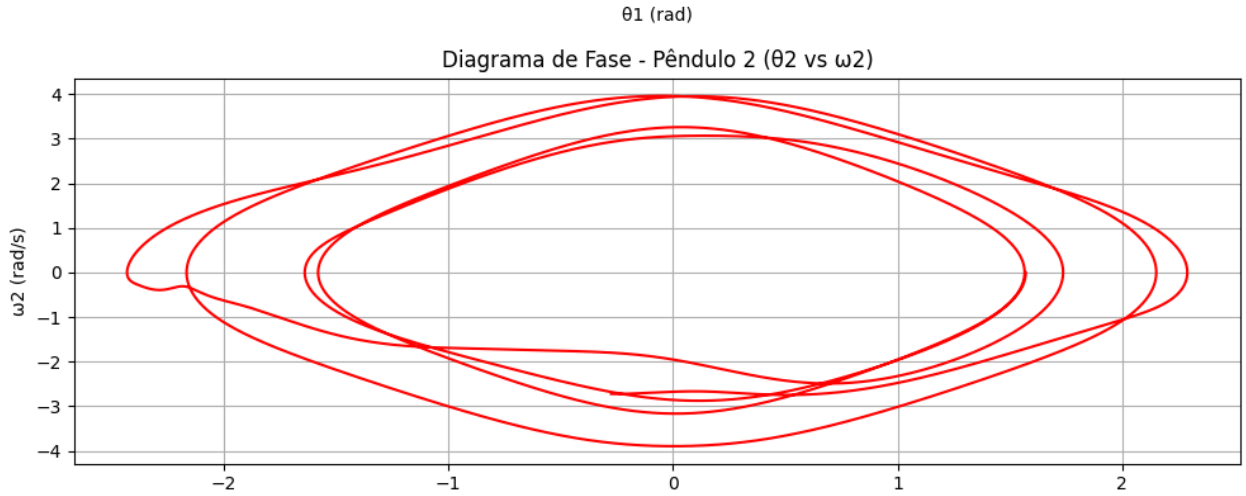

```
import numpy as np
import matplotlib.pyplot as plt

# Parâmetros
m1 = 1.0
m2 = 1.0
l1 = 1.0
l2 = 1.0
g = 9.81

# Tempo
dt = 0.001
t_end = 20
time = np.arange(0, t_end, dt)

# Condições iniciais ajustadas
theta1 = np.pi/4      # Ângulo inicial pêndulo 1
theta2 = np.pi/2     # Ângulo inicial pêndulo 2
omega1 = 0.0      # Velocidade inicial pêndulo 1
omega2 = 0.0      # Velocidade inicial pêndulo 2

# Vetores para armazenar
theta1_arr = np.zeros(len(time))
theta2_arr = np.zeros(len(time))
omega1_arr = np.zeros(len(time))
omega2_arr = np.zeros(len(time))

# Integração por Euler
for i in range(len(time)):
    theta1_arr[i] = theta1
    theta2_arr[i] = theta2
    omega1_arr[i] = omega1
    omega2_arr[i] = omega2

    delta_theta = theta2 - theta1
    denom1 = (m1 + m2) * l1 - m2 * l1 * np.cos(delta_theta)**2
    denom2 = (l2 / l1) * denom1

    alpha1 = (-m2 * l1 * omega1**2 * np.sin(delta_theta) * np.cos(delta_theta) +
              m2 * g * np.sin(theta2) * np.cos(delta_theta) +
              m2 * l2 * omega2**2 * np.sin(delta_theta) -
              (m1 + m2) * g * np.sin(theta1)) / denom1

    alpha2 = (l1 * omega1**2 * np.sin(delta_theta) -
              g * np.sin(theta2) +
              g * np.sin(theta1) * np.cos(delta_theta)) / denom2

    # Atualizar ângulos e velocidades
    omega1 += alpha1 * dt
    omega2 += alpha2 * dt
    theta1 += omega1 * dt
    theta2 += omega2 * dt

# Gráfico
plt.figure(figsize=(10, 6))
plt.plot(time, theta1_arr, label=r'$\theta_1$ (rad)')
plt.plot(time, theta2_arr, label=r'$\theta_2$ (rad)')
plt.title('Resposta Temporal Dos Ângulos no Sistema')
plt.xlabel('Tempo (s)')
plt.ylabel('Ângulo (radianos)')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()


# Trajetórias
plt.plot(theta1_arr, omega1_arr, 'b', label='Trajetório da massa 1')
plt.plot(theta2_arr, omega2_arr, 'r', label='Trajetório da massa 2')

# Condições iniciais
plt.plot(theta1_arr[0], omega1_arr[0], 'bo', label='Condição inicial massa 1')
plt.plot(theta2_arr[0], omega2_arr[0], 'ro', label='Condição inicial massa 2')

# Estética
plt.title('Retrato de Fase Com o Campo vetorial')
plt.xlabel('Ângulo (radianos)')
plt.ylabel('Velocidade angular (rad/s)')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```






```
#Diagrama de fase da Q2

# Diagrama de fase do pêndulo 1
plt.figure(figsize=(10, 4))
plt.plot(theta1_arr, omega1_arr, 'b')
plt.title('Diagrama de Fase - Pêndulo 1 (θ1 vs ω1)')
plt.xlabel('θ1 (rad)')
plt.ylabel('ω1 (rad/s)')
plt.grid()
plt.tight_layout()
plt.show()

# Diagrama de fase do pêndulo 2
plt.figure(figsize=(10, 4))
plt.plot(theta2_arr, omega2_arr, 'r')
plt.title('Diagrama de Fase - Pêndulo 2 (θ2 vs ω2)')
plt.xlabel('θ2 (rad)')
plt.ylabel('ω2 (rad/s)')
plt.grid()
plt.tight_layout()
plt.show()
```






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

# Parâmetros do Pêndulo Duplo
m1 = 1.0  # Massa do primeiro pêndulo em kg
m2 = 1.0  # Massa do segundo pêndulo em kg
l1 = 1.0  # Comprimento do primeiro pêndulo em metros
l2 = 1.0  # Comprimento do segundo pêndulo em metros
g = 9.81  # Aceleração da gravidade em m/s²

# Configurações de tempo
```

```
dt = 0.001     # Passo de tempo em segundos (quanto menor, maior a precisão)
t_end = 20     # Tempo total de simulação em segundos
time = np.arange(0, t_end, dt)

# Condições iniciais
theta1_init = np.pi / 4  # Ângulo inicial do pêndulo 1
theta2_init = np.pi / 2  # Ângulo inicial do pêndulo 2
omega1_init = 0.0        # Velocidade angular inicial do pêndulo 1
omega2_init = 0.0        # Velocidade angular inicial do pêndulo 2

# Arrays para armazenar os ângulos e velocidades angulares
theta1_arr = np.zeros(len(time))
theta2_arr = np.zeros(len(time))
omega1_arr = np.zeros(len(time))
omega2_arr = np.zeros(len(time))

# Definir condições iniciais para a simulação
theta1 = theta1_init
theta2 = theta2_init
omega1 = omega1_init
omega2 = omega2_init

# Simulação usando o método de Euler
for i in range(len(time)):
    theta1_arr[i] = theta1
    theta2_arr[i] = theta2
    omega1_arr[i] = omega1
    omega2_arr[i] = omega2

    # Equações de movimento do pêndulo duplo
    delta_theta = theta2 - theta1

    # Denominadores de alpha1 e alpha2
    denom1 = (m1 + m2) * l1 - m2 * l1 * np.cos(delta_theta)**2
    denom2 = (l2 / l1) * denom1

    # Acelerações angulares (alpha1 e alpha2)
    # Prevenir divisão por zero
    if denom1 == 0:
        alpha1 = 0
    else:
        alpha1 = (-m2 * l1 * omega1**2 * np.sin(delta_theta) * np.cos(delta_theta) +
                  m2 * g * np.sin(theta2) * np.cos(delta_theta) +
                  m2 * l2 * omega2**2 * np.sin(delta_theta) -
                  (m1 + m2) * g * np.sin(theta1)) / denom1

    if denom2 == 0:
        alpha2 = 0
    else:
        alpha2 = (l1 * omega1**2 * np.sin(delta_theta) - g * np.sin(theta2) +
                  g * np.sin(theta1) * np.cos(delta_theta)) / denom2

    # Atualiza as velocidades angulares e ângulos usando o método de Euler
    omega1 += alpha1 * dt
    omega2 += alpha2 * dt
    theta1 += omega1 * dt
    theta2 += omega2 * dt

# --- Preparar o gráfico de campo direcional (quiver) para o espaço de fases do Pêndulo 1 ---

# Criar uma grade para theta1 e omega1
theta1_grid_vals = np.linspace(-np.pi, np.pi, 20)
omega1_grid_vals = np.linspace(-5, 5, 20)  # Ajuste o intervalo conforme necessário
theta1_grid, omega1_grid = np.meshgrid(theta1_grid_vals, omega1_grid_vals)

# Calcular d(theta1)/dt e d(omega1)/dt para os pontos da grade
dtheta1_dt_grid = omega1_grid  # d(theta1)/dt = omega1

# Para obter d(omega1)/dt (alpha1), fixamos theta2 e omega2 em seus valores iniciais
domega1_dt_grid = np.zeros_like(theta1_grid)

for r_idx, row in enumerate(theta1_grid):
    for c_idx, t1_val in enumerate(row):
        o1_val = omega1_grid[r_idx, c_idx]

        # Recalcula alpha1 para cada ponto da grade, usando theta2 e omega2 iniciais
        delta_theta_grid = theta2_init - t1_val
        denom1_grid = (m1 + m2) * l1 - m2 * l1 * np.cos(delta_theta_grid)**2

        if denom1_grid == 0:
            alpha1_grid_val = 0
        else:
            alpha1_grid_val = (-m2 * l1 * o1_val**2 * np.sin(delta_theta_grid) * np.cos(delta_theta_grid) +
                               m2 * g * np.sin(theta2_init) * np.cos(delta_theta_grid) +
                               m2 * l2 * omega2_init**2 * np.sin(delta_theta_grid) -
                               (m1 + m2) * g * np.sin(t1_val)) / denom1_grid
        domega1_dt_grid[r_idx, c_idx] = alpha1_grid_val

# Normaliza os vetores para melhor visualização
magnitude_pendulum1 = np.sqrt(dtheta1_dt_grid**2 + domega1_dt_grid**2)
magnitude_pendulum1[magnitude_pendulum1 == 0] = 1e-9  # Evita divisão por zero
dtheta1_dt_norm = dtheta1_dt_grid / magnitude_pendulum1
domega1_dt_norm = domega1_dt_grid / magnitude_pendulum1

# --- Diagrama de Fase combinado com o campo direcional do Pêndulo 1 ---

plt.figure(figsize=(10, 8))

# Plota o campo vetorial (quiver plot) do espaço de fases do Pêndulo 1
plt.quiver(theta1_grid, omega1_grid, dtheta1_dt_norm, domega1_dt_norm,
           color='gray', alpha=0.6, scale=30, width=0.002,
           label='Campo Direcional (Pêndulo 1, com θ2 e ω2 fixos nos valores iniciais)')

# Plota a trajetória do Pêndulo 1
plt.plot(theta1_arr, omega1_arr, color='blue', linewidth=1.5, label='Trajetória do Pêndulo 1')
# Marca o ponto inicial do Pêndulo 1
plt.plot(theta1_arr[0], omega1_arr[0], 'o', color='darkblue', markersize=8, label='Início Pêndulo 1')

# Plota a trajetória do Pêndulo 2
plt.plot(theta2_arr, omega2_arr, color='red', linewidth=1.5, label='Trajetória do Pêndulo 2')
# Marca o ponto inicial do Pêndulo 2
plt.plot(theta2_arr[0], omega2_arr[0], 'o', color='darkred', markersize=8, label='Início Pêndulo 2')

plt.title('Diagrama de Fase: Trajetórias de ambos os Pêndulos com Campo Direcional do Pêndulo 1\n(Campo assume θ2 e ω2 fixos nos valores iniciais)')
plt.xlabel('Ângulo (θ) [rad]')
plt.ylabel('Velocidade Angular (ω) [rad/s]')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

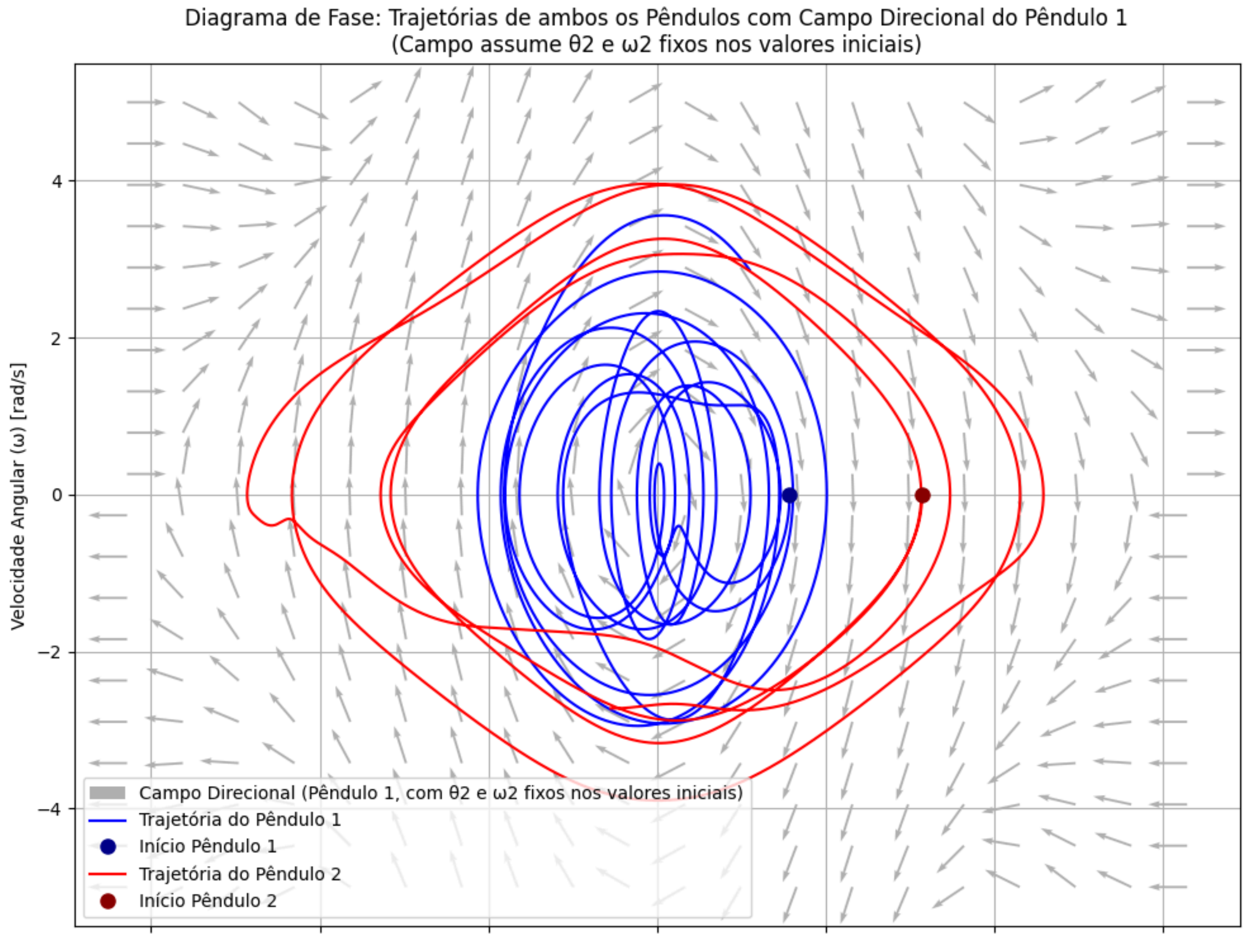


Diagrama de Fase: Trajetórias de ambos os Pêndulos com Campo Direcional do Pêndulo 1
(Campo assume θ2 e ω2 fixos nos valores iniciais)
Velocidade Angular (ω) [rad/s]
4
2
0
−2
−4
Campo Direcional (Pêndulo 1, com θ2 e ω2 fixos nos valores iniciais)
Trajetória do Pêndulo 1
Início Pêndulo 1
Trajetória do Pêndulo 2
Início Pêndulo 2