Modelação de Sistemas Físicos - Aula Prática nº10

Realização e resolução de problemas sobre:

- Osciladores, interpolação

Exercício 1: Movimento de um pêndulo simples

Uma massa suspensa do teto por um fio de comprimento $L = 1\,\mathrm{m}$ oscila à volta da sua posição de equilíbrio, expressa por $\theta = 0\,\mathrm{rad}$, de acordo com a equação diferencial,

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -\frac{g}{L}\sin\theta$$

onde $g = 9.8\,\mathrm{m/s}^2$ é a aceleração gravítica.

Pendulo simples

a) Simule o movimento do pêndulo usando o método de Euler-Cromer durante 10 s. A massa inicia com ângulo $\theta = 0.1\,\mathrm{rad}$ e com velocidade nula.

Este problema é resolvido em coordenadas angulares, i.e., usando:

- a aceleração angular, $\gamma = \mathrm{d}^2\theta/\mathrm{d}t^2 = \mathrm{d}\omega/\mathrm{d}t$;
- a velocidade angular, $\omega = \mathrm{d}\theta/\mathrm{d}t$;
- o ângulo (posição angular), θ

Sabendo que (por definição) a aceleração instantânea é dada por

$$\gamma = -\frac{g}{L}\sin\theta.$$

O método de Euler-Cromer em coordenadas angulares é dado por,

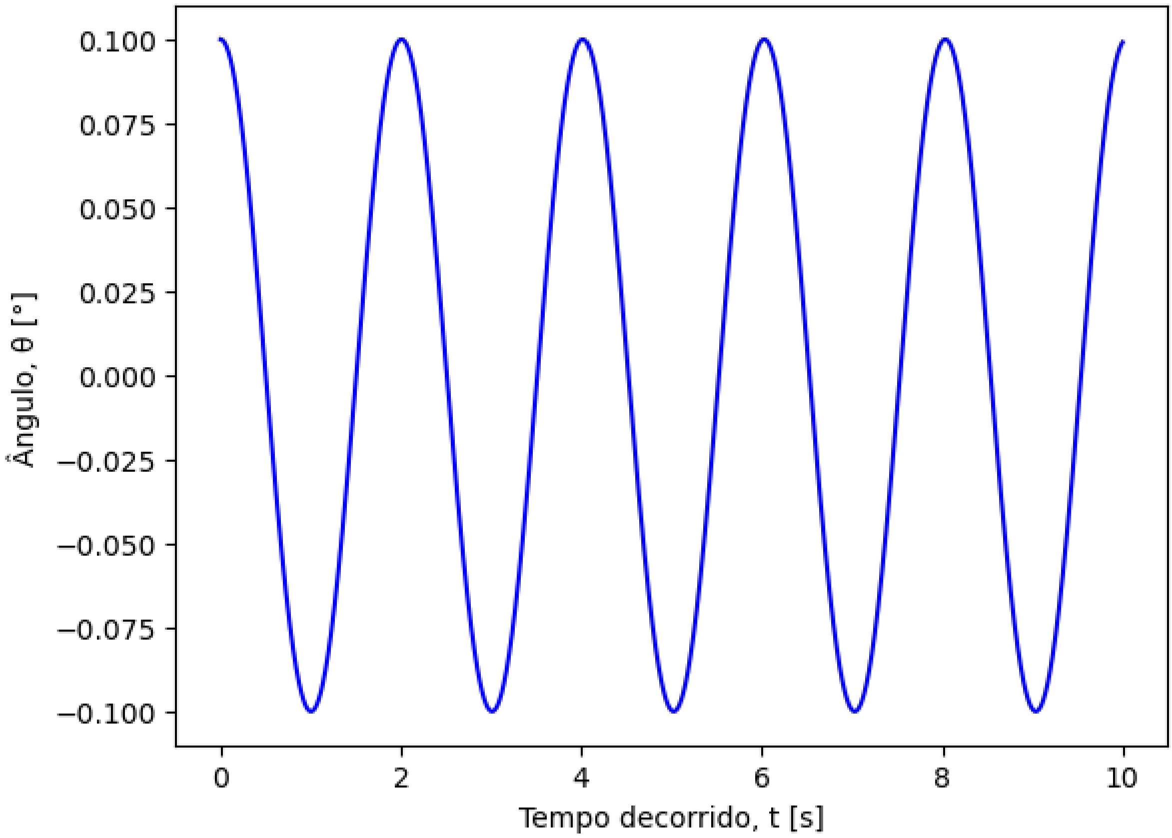
$$\gamma_i = -g\,\sin\theta_i/L$$

$$\omega_{i+1} = \omega_i + \gamma_i\,\delta\tau$$

$$\theta_{i+1} = \theta_i + \omega_{i+1}\,\delta\tau$$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

t0 = 0.0                          # condição inicial, tempo [s]
tf = 10.0                         # limite do domínio, tempo final [s]
dt = 0.001                        # passo [s]

θ0 = 0.1                          # condição inicial, ângulo inicial [rad]
ω0 = 0.0                          # condição inicial, velocidade angular inicial [rad

L = 1.0                           # comprimento do fio [m]
g = 9.8                           # aceleração gravítica [m/s^2]
```

```python
# inicializar domínio temporal [s]
t = np.arange(t0, tf, dt)

# inicializar solução
γ = np.zeros(np.size(t))          # aceleração angular [rad/s2]
ω = np.zeros(np.size(t))          # velocidade angular [rad/s]
θ = np.zeros(np.size(t))          # ângulo [rad]
θ[0] = θ0
ω[0] = ω0

# método de Euler
for i in range(np.size(t) - 1):
    γ[i] = - g / L * np.sin(θ[i])
    ω[i + 1] = ω[i] + γ[i] * dt
    θ[i + 1] = θ[i] + ω[i + 1] * dt

plt.plot(t, θ, 'b-')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("Ângulo, θ [°]")
plt.show()
```



b) Para ângulos de oscilação pequenos, a equação diferencial pode ser approximada a

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -\frac{g}{L}\theta,$$

cuja solução analítica é

$$\theta(t) = A\cos\left(\sqrt{\frac{g}{L}}t + \phi\right).$$

Compare esta solução com a solução numérica obtida na alínea a), escolhendo valores adequados para A e ϕ.

Solução:

Comparemos então a solução analítica, válida apenas para amplitudes de oscilação pequenos,

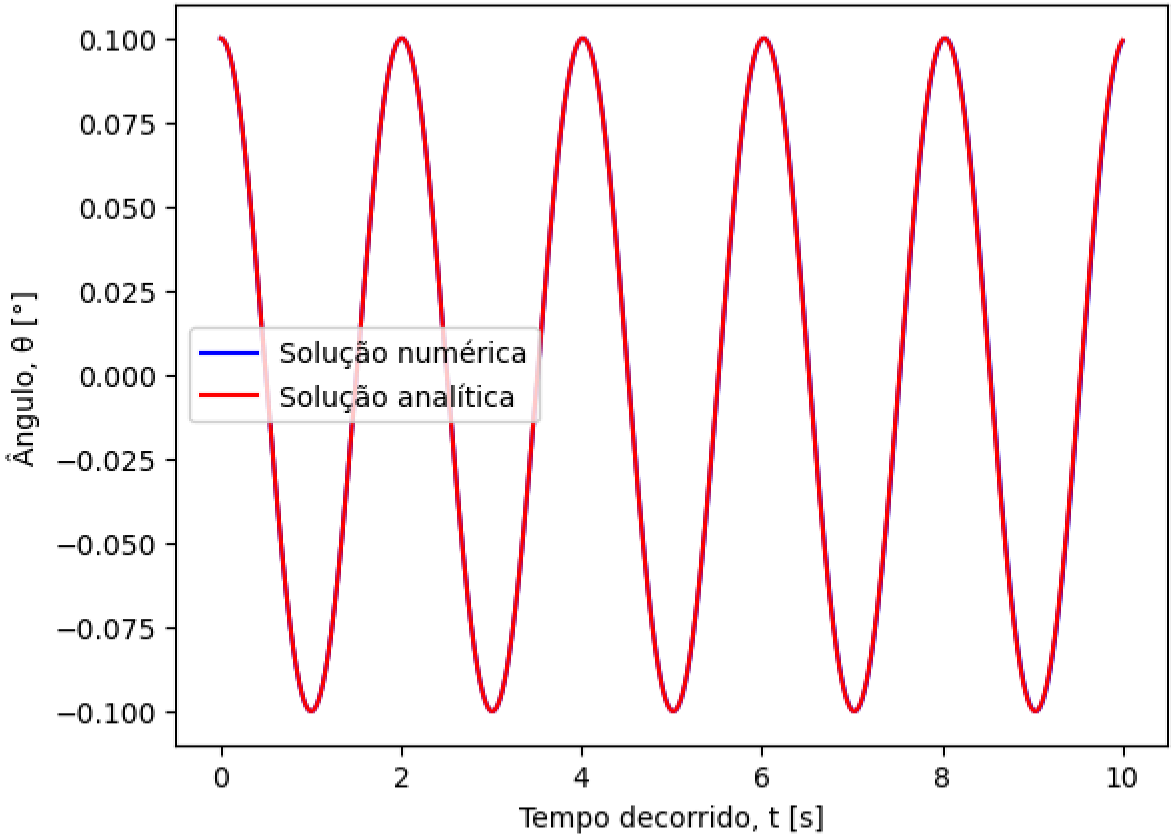
$$\theta_{\mathrm{a}}(t) = A\cos\left(\sqrt{\frac{g}{L}}t + \phi\right),$$

com a solução numérica $\theta(t)$ obtida na alínea anterior.

Notemos que:

- A constante de fase $\phi = 0$. Só assim, o pendulo inicia o movimento com amplitude máxima e velocidade nula.
- A amplitude $A = 0.1\,\mathrm{rad}$.

In [2]:
```
# Solução analítica
A = 0.1                                        # amplitude [rad]
φ = 0.0                                        # constante de fase [rad]
θ_a = A * np.cos(np.sqrt(g / L) * t + φ)  # equação do movimento [rad]

plt.plot(t, θ, 'b-', t, θ_a, 'r-')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("Ângulo, θ [°]")
plt.legend(['Solução numérica', 'Solução analítica'])
plt.show()
```



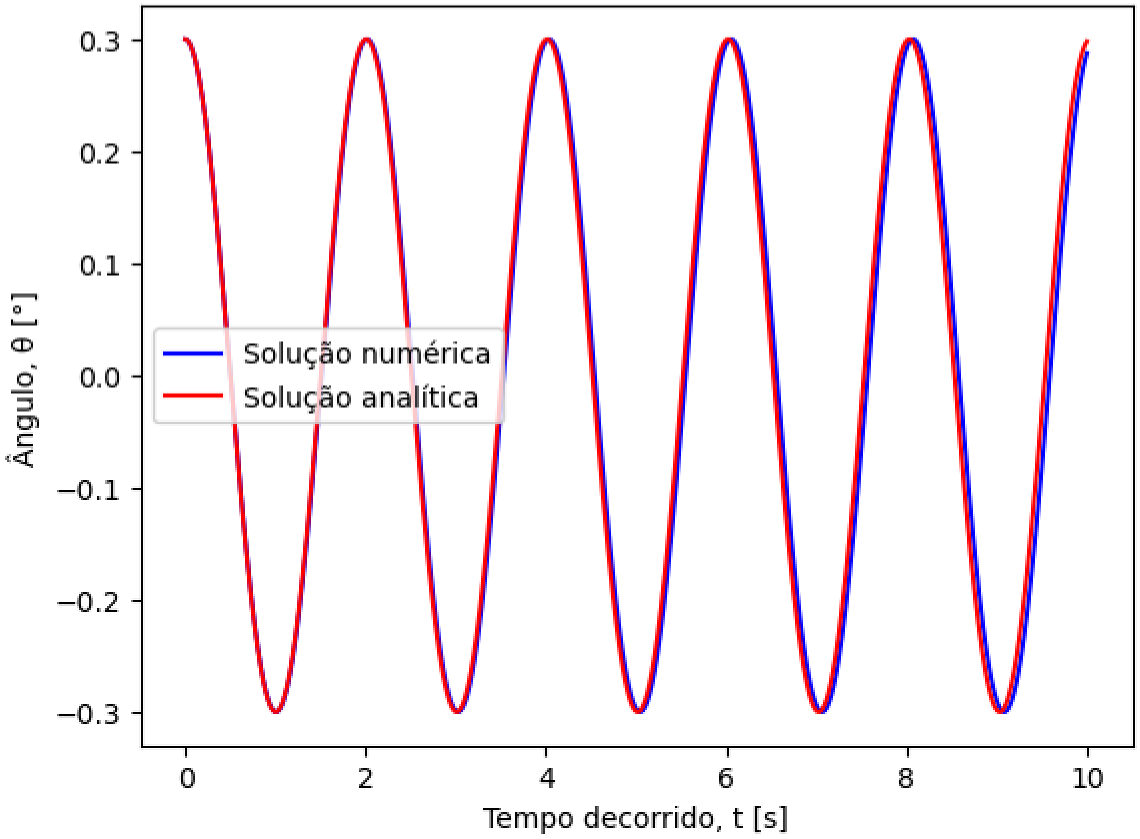
c) Repita a comparação com ângulos iniciais de 0.3 rad e 0.5 rad. As soluções numéricas e teóricas são diferentes?

Solução:

Usando $\theta_0 = A = 0.3\,\mathrm{rad}$ temos:

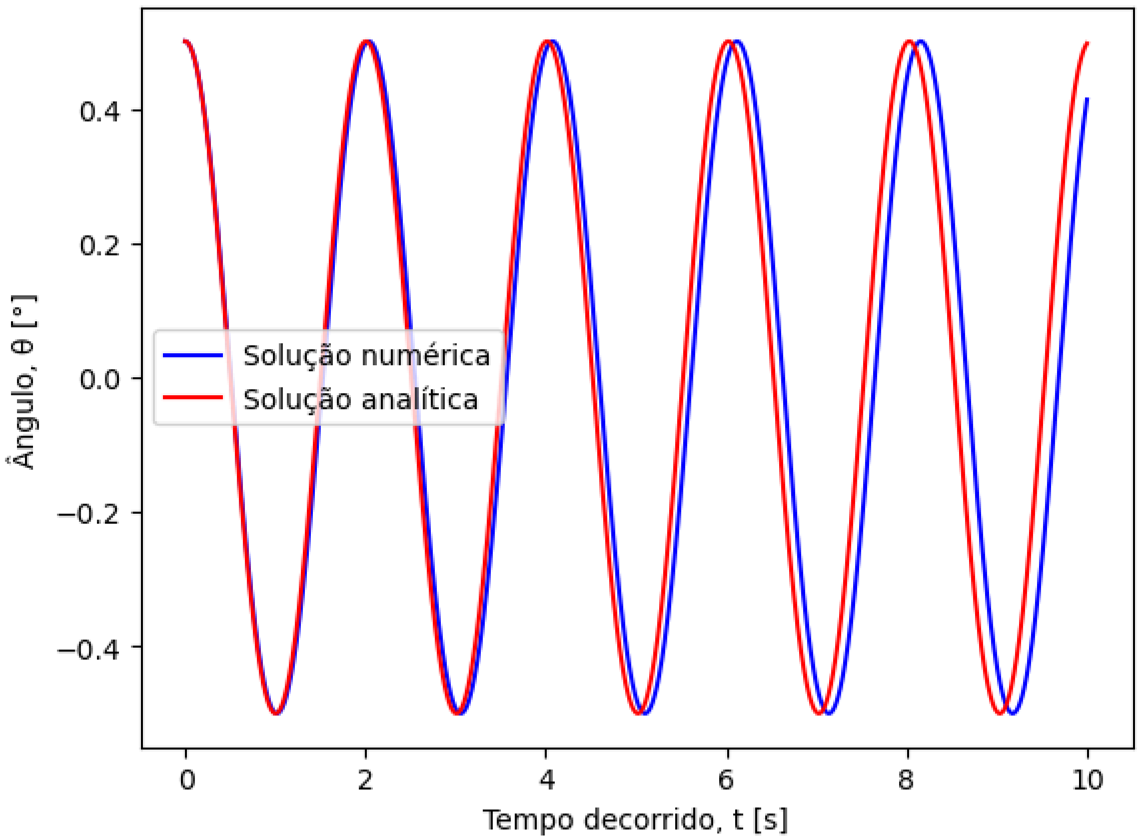
In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```python
t0 = 0.0                          # condição inicial, tempo [s]
tf = 10.0                         # limite do domínio, tempo final [s]
dt = 0.001                        # passo [s]

θ0 = 0.3                          # condição inicial, ângulo inicial [rad]
ω0 = 0.0                          # condição inicial, velocidade angular inicial [rad

L = 1.0                           # comprimento do fio [m]
g = 9.8                           # aceleração gravítica [m/s^2]

# inicializar domínio temporal [s]
t = np.arange(t0, tf, dt)

# inicializar solução
γ = np.zeros(np.size(t))          # aceleração angular [rad/s2]
ω = np.zeros(np.size(t))          # velocidade angular [rad/s]
θ = np.zeros(np.size(t))          # ângulo [rad]
θ[0] = θ0
ω[0] = ω0

# método de Euler
for i in range(np.size(t) - 1):
    γ[i] = - g / L * np.sin(θ[i])
    ω[i + 1] = ω[i] + γ[i] * dt
    θ[i + 1] = θ[i] + ω[i + 1] * dt

# Solução analítica
A = 0.3                                   # amplitude [rad]
φ = 0.0                                   # constante de fase [rad]
θ_a = A * np.cos(np.sqrt(g / L) * t + φ)  # equação do movimento [rad]

plt.plot(t, θ, 'b-', t, θ_a, 'r-')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("Ângulo, θ [°]")
plt.legend(['Solução numérica', 'Solução analítica'])
plt.show()
```



Usando $\theta_0 = A = 0.5\ \mathrm{rad}$ temos:

In [4]:
```python
import numpy as np
import matplotlib.pyplot as plt

t0 = 0.0                          # condição inicial, tempo [s]
tf = 10.0                         # limite do domínio, tempo final [s]
dt = 0.001                        # passo [s]

θ0 = 0.5                          # condição inicial, ângulo inicial [rad]
ω0 = 0.0                          # condição inicial, velocidade angular inicial [rad

L = 1.0                           # comprimento do fio [m]
g = 9.8                           # aceleração gravítica [m/s^2]

# inicializar domínio temporal [s]
t = np.arange(t0, tf, dt)

# inicializar solução
γ = np.zeros(np.size(t))          # aceleração angular [rad/s2]
ω = np.zeros(np.size(t))          # velocidade angular [rad/s]
θ = np.zeros(np.size(t))          # ângulo [rad]
θ[0] = θ0
ω[0] = ω0

# método de Euler
for i in range(np.size(t) - 1):
    γ[i] = - g / L * np.sin(θ[i])
    ω[i + 1] = ω[i] + γ[i] * dt
    θ[i + 1] = θ[i] + ω[i + 1] * dt

# Solução analítica
A = 0.5                                   # amplitude [rad]
φ = 0.0                                   # constante de fase [rad]
θ_a = A * np.cos(np.sqrt(g / L) * t + φ)  # equação do movimento [rad]

plt.plot(t, θ, 'b-', t, θ_a, 'r-')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("Ângulo, θ [°]")
plt.legend(['Solução numérica', 'Solução analítica'])
plt.show()
```



Pergunta 1:

Para qual intervalo de ângulos iniciais podemos confiar na aproximação teórica?

Exercício 2: Medição do período de oscilação

Use interpolação com polinómios de Lagrange para medir o período das oscilações nas simulações de Exercício 1. Compare os resultados com o valor teórico

$$T = 2\pi\sqrt{\frac{L}{g}}$$

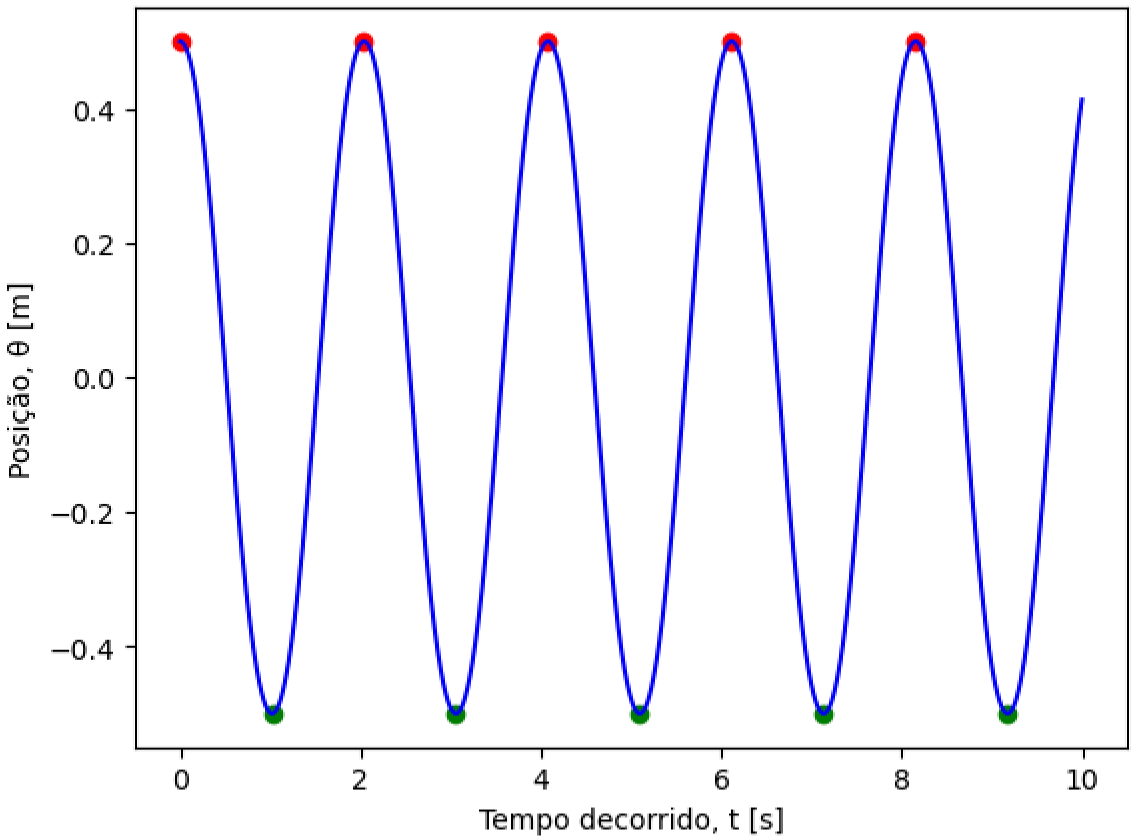
Para medir o período, será necessário considerar dois máximos consecutivos, e calcular a diferênca temporal entre os mesmos.

Para identificar programaticamente cada máximo, encontre nos dados os três pontos mais próximos do máximo, e usar a função `maxminv()` (disponível no e-learning no ficheiro `function_lagrange.py` na pasta Programas Python) para encontrar o máximo da parábola que interpola entre esses três pontos.

Solução:

In [5]:
```
def maxminv(x0,x1,x2,y0,y1,y2):
    # Máximo ou mínimo usando o polinómio de Lagrange
    # Dados (input): (x0,y0), (x1,y1) e (x2,y2)
    # Resultados (output): xm, ym
    xab = x0 - x1
    xac = x0 - x2
    xbc = x1 - x2
```

```python
    a = y0 / (xab * xac)
    b = -y1 / (xab * xbc)
    c = y2 / (xac * xbc)
    xmla = (b + c) * x0 + (a + c) * x1 + (a + b) * x2
    xm = 0.5 * xmla / (a + b + c)
    xta = xm - x0
    xtb = xm - x1
    xtc = xm - x2
    ym = a * xtb * xtc + b * xta * xtc + c * xta * xtb
    return xm, ym

# arrays com valores máximos, mínimos e respetivos instantes de tempo
t_max = np.array([t0])
θ_max = np.array([θ0])
t_min = np.array([])
θ_min = np.array([])

# Pesquisar pelo máximos e mínimos de θ.
# Aqui definimos uma "janela corrida" no tempo em passos de 2, i.e, analisamos
# os máximos/mínimos de θ que ocorrem entre t[i] e t[i+2], com i = 0, 2, 4, 6, etc.
# de forma a evitar encontros duplicados
for i in range(0, np.size(t) - 3, 2):
    # Percorrer domínio temporal em sequências de três valores:
    # θ[i], θ[i+1], θ[i+2] e respetivos instantes de tempo para i = 0, ..., N-3
    tm, θm = maxminv(t[i], t[i+1], t[i+2], θ[i], θ[i+1], θ[i+2])

    # verificar se max/min esta dentro da "janela corrida" (t[i] <-> t[i+2])
    if t[i] < tm and tm < t[i+2]:
        # θm é máximo?
        if θm > np.maximum(θ[i], θ[i+2]):
            t_max = np.append(t_max, tm)
            θ_max = np.append(θ_max, θm)
        else: # caso contrário é mínimo
            t_min = np.append(t_min, tm)
            θ_min = np.append(θ_min, θm)

plt.plot(t, θ, 'b-')
plt.scatter(t_max, θ_max, color='red')
plt.scatter(t_min, θ_min, color='green')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("Posição, θ [m]")
plt.show()
```



A distância temporal entre os dois primeiros máximos é obtida por:

In [6]:
```
ΔT = t_max[1] - t_max[0]
print("ΔT = {0:.4f} s".format(ΔT))
```

```
ΔT = 2.0384 s
```

Que compara com $T = 2\pi\sqrt{L/g} = T = 2\pi\ \sqrt{1/9.8}$:

In [7]:
```
print("T = {0:.4f} s".format(2.0 * np.pi * np.sqrt(1.0/9.8)))
```

```
T = 2.0071 s
```

Exercício 3: Variação do período com comprimento do pêndulo

a) Meça o período das oscilações nas simulações para diferentes valores do comprimento do pêndulo, L.

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt

def maxminv(x0,x1,x2,y0,y1,y2):
    # Máximo ou mínimo usando o polinómio de Lagrange
    # Dados (input): (x0,y0), (x1,y1) e (x2,y2)
    # Resultados (output): xm, ym
    xab = x0 - x1
    xac = x0 - x2
    xbc = x1 - x2
    a = y0 / (xab * xac)
    b = -y1 / (xab * xbc)
    c = y2 / (xac * xbc)
    xmla = (b + c) * x0 + (a + c) * x1 + (a + b) * x2
    xm = 0.5 * xmla / (a + b + c)
    xta = xm - x0
    xtb = xm - x1
    xtc = xm - x2
```

```python
    ym = a * xtb * xtc + b * xta * xtc + c * xta * xtb
    return xm, ym

t0 = 0.0                            # condição inicial, tempo [s]
tf = 10.0                           # limite do domínio, tempo final [s]
dt = 0.001                          # passo [s]

θ0 = 0.1                            # condição inicial, ângulo inicial [rad]
ω0 = 0.0                            # condição inicial, velocidade angular inicial [rad

L = 1.0                             # comprimento do fio [m]
g = 9.8                             # aceleração gravítica [m/s^2]

# inicializar domínio temporal [s]
t = np.arange(t0, tf, dt)

# inicializar solução
γ = np.zeros(np.size(t))            # aceleração angular [rad/s2]
ω = np.zeros(np.size(t))            # velocidade angular [rad/s]
θ = np.zeros(np.size(t))            # ângulo [rad]
θ[0] = θ0
ω[0] = ω0

# arrays com valores máximos, mínimos e respetivos instantes de tempo
t_max = np.array([t0])
θ_max = np.array([θ0])
t_min = np.array([])
θ_min = np.array([])

# método de Euler
for i in range(np.size(t) - 1):
    γ[i] = - g / L * np.sin(θ[i])
    ω[i + 1] = ω[i] + γ[i] * dt
    θ[i + 1] = θ[i] + ω[i + 1] * dt

# Pesquisar pelo máximos e mínimos de θ.
# Aqui definimos uma "janela corrida" no tempo em passos de 2, i.e, analisamos
# os máximos/mínimos de θ que ocorrem entre t[i] e t[i+2], com i = 0, 2, 4, 6, etc.
# de forma a evitar encontros duplicados
for i in range(0, np.size(t) - 3, 2):
    # Percorrer domínio temporal em sequências de três valores:
    # θ[i], θ[i+1], θ[i+2] e respetivos instantes de tempo para i = 0, ..., N-3
    tm, θm = maxminv(t[i], t[i+1], t[i+2], θ[i], θ[i+1], θ[i+2])

    # verificar se max/min esta dentro da "janela corrida" (t[i] <-> t[i+2])
    if t[i] < tm and tm < t[i+2]:
        # θm é máximo?
        if θm > np.maximum(θ[i], θ[i+2]):
            t_max = np.append(t_max, tm)
            θ_max = np.append(θ_max, θm)
        else: # caso contrário é mínimo
            t_min = np.append(t_min, tm)
            θ_min = np.append(θ_min, θm)

T = t_max[1] - t_max[0]
print("L = {0:.4f} m".format(L))
print("T = {0:.4f} s".format(T))
```

```
L = 1.0000 m
T = 2.0078 s
```

Registamos os valores medidos nos arrays `L_arr` e `ΔT_arr`:

In [9]:
```
L_arr = np.array([1.0000, 1.1000, 1.2000, 1.3000, 1.4000, 1.5000, 1.6000, 1.7000, 1.8
T_arr = np.array([2.0078, 2.1059, 2.1995, 2.2894, 2.3758, 2.4592, 2.5399, 2.6181, 2.6
```

b) Faça um gráfico do logaritmo do período T em função do logaritmo do comprimento L.

Solução:

No temos que segundo o modelo aproximando para amplitudes de oscilação pequenos:

$$T = 2\pi\sqrt{\frac{L}{g}}$$

ou

$$\log(T) = \frac{1}{2}\log(L) + \log\left(\frac{2\pi}{\sqrt{g}}\right)$$

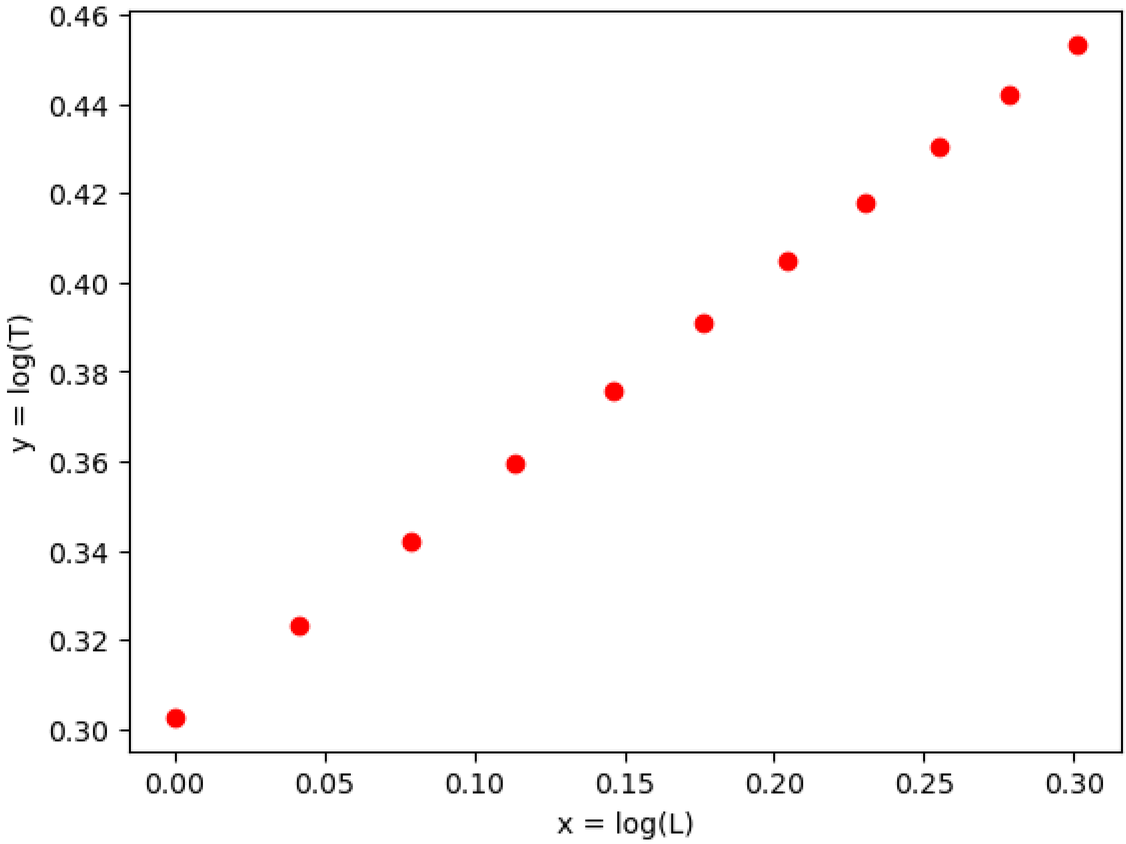
Realizamos uma substituição de variáveis de forma a obtermos uma equação de uma reta, $y = mx + b$, onde:

$$x \equiv \log(L)$$

$$y \equiv \log(T)$$

e ainda, $m = 1/2$ e $b = \log(2\pi/\sqrt{g})$.

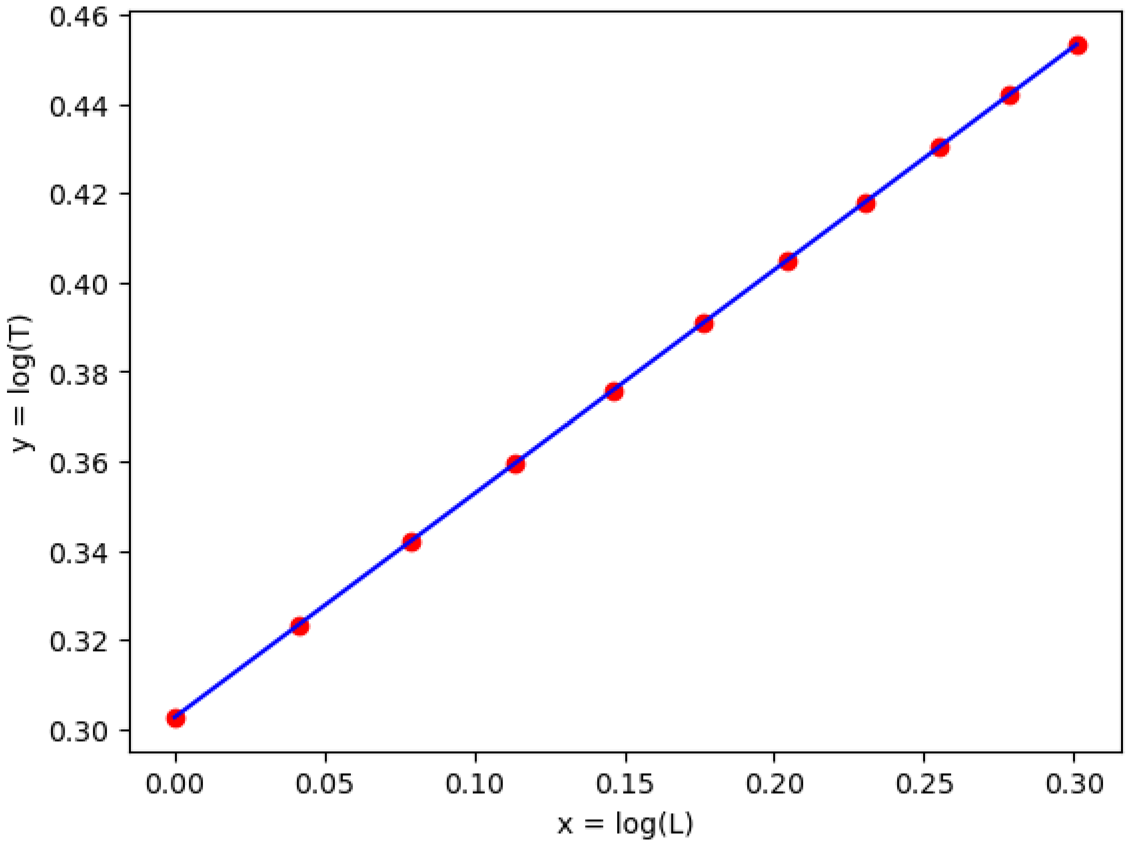
In [10]:
```
x = np.log10(L_arr)
y = np.log10(T_arr)
m = 0.5
b = np.log10(2 * np.pi / np.sqrt(g))

plt.scatter(x, y, color='red')
plt.xlabel("x = log(L)")
plt.ylabel("y = log(T)")
plt.show()
```



c) Faça um ajuste linear e adicione a reta do ajuste ao gráfico da alínea anterior. Encontre o declive e o seu erro.

Solução:

In [11]:
```
# obter parámetros ajustados e respetivos erros
# declive: p[0] +/- sqrt(C[0,0])
# ordenada na orígem: p[1] +/- sqrt(C[1,1])
# C[i,j] matriz da covariância
# Ordem do ajuste = 1, i.e., p(x) = p[0] * x + p[1]
p, C = np.polyfit(x, y, 1, cov = True)

# Reta ajustada
plt.plot(x, p[0] * x + p[1], 'b-')
# Pontos medidos
plt.scatter(x, y, color='red')
plt.xlabel("x = log(L)")
plt.ylabel("y = log(T)")
plt.show()
```



In [12]:
```
# O declive e respetivo erro obtidos no ajuste são:
print("m ± Δm = {} ± {}".format(p[0] , np.sqrt(C[0,0])))
```

```
m ± Δm = 0.5001160143355312 ± 1.9255441713394896e-05
```

Pergunta 2:

Qual é o declive do ajuste? Explique o significado deste resultado.