Modelação de Sistemas Físicos - Aula Prática nº11

Realização e resolução de problemas sobre:

- Cap. 7: Osciladores amortecidos e forçados

Problemas do Capítulo 7: Oscilador harmónico amortecido

Um corpo de massa $m = 0.25\ \mathrm{kg}$ preso a uma mola com constante elástica $k = 1\ \mathrm{N/m}$, está sujeito a uma força com coeficiente de amortecimento $b = 0.1\ \mathrm{kg/s}$, correspondendo a uma força resultante,

$$F = -kx - bv,$$

onde v é a velocidade ao longo do movimento.

a) Calcule a lei do movimento, assumindo que a massa está em repouso em $x = 0.4\ \mathrm{m}$ quando $t = 0\ \mathrm{s}$. Faça o gráfico da posição em função do tempo de $t = 0\ \mathrm{s}$ até $t = 20\ \mathrm{s}$.

Solução

Podemos usar o método de Euler-Cromer (adequado para movimentos oscilatórios) para descrever a dinâmica do movimento da massa.

Sendo força resultante dada por,

$$F = -kx - bv,$$

ou seja, a aceleração instantânea é

$$a = -\frac{1}{m}(kx + bv),$$

em que consideramos

- $x(t=0) = 0.4\ \mathrm{m}$
- $v(t=0) = 0\ \mathrm{m/s}$
- $m = 0.25\ \mathrm{kg}$
- $k = 1\ \mathrm{N/m}$
- $b = 0.1\ \mathrm{kg/s}$

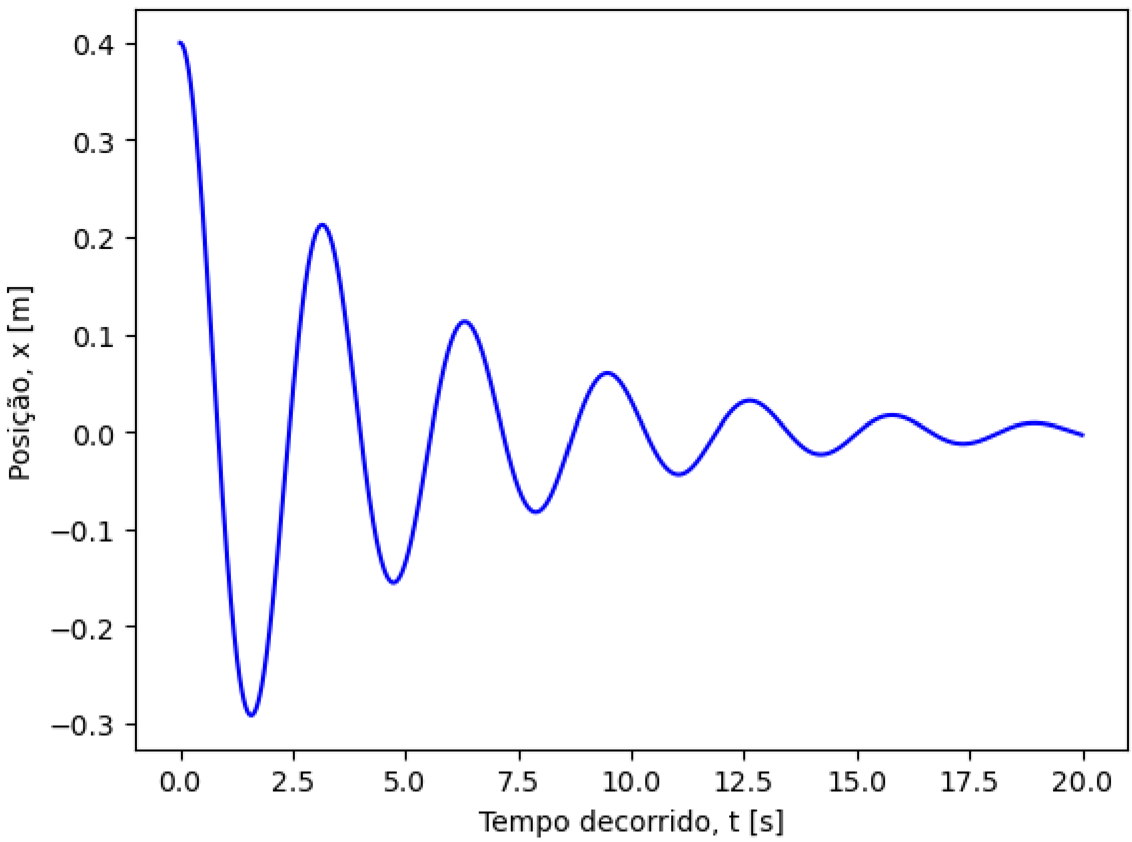
Assim, usando o método de Euler-Cromer,

$$a_i = -(k\,x_i + b\,v_i)/m$$

$$v_{i+1} = v_i + a_i\,\delta\tau$$

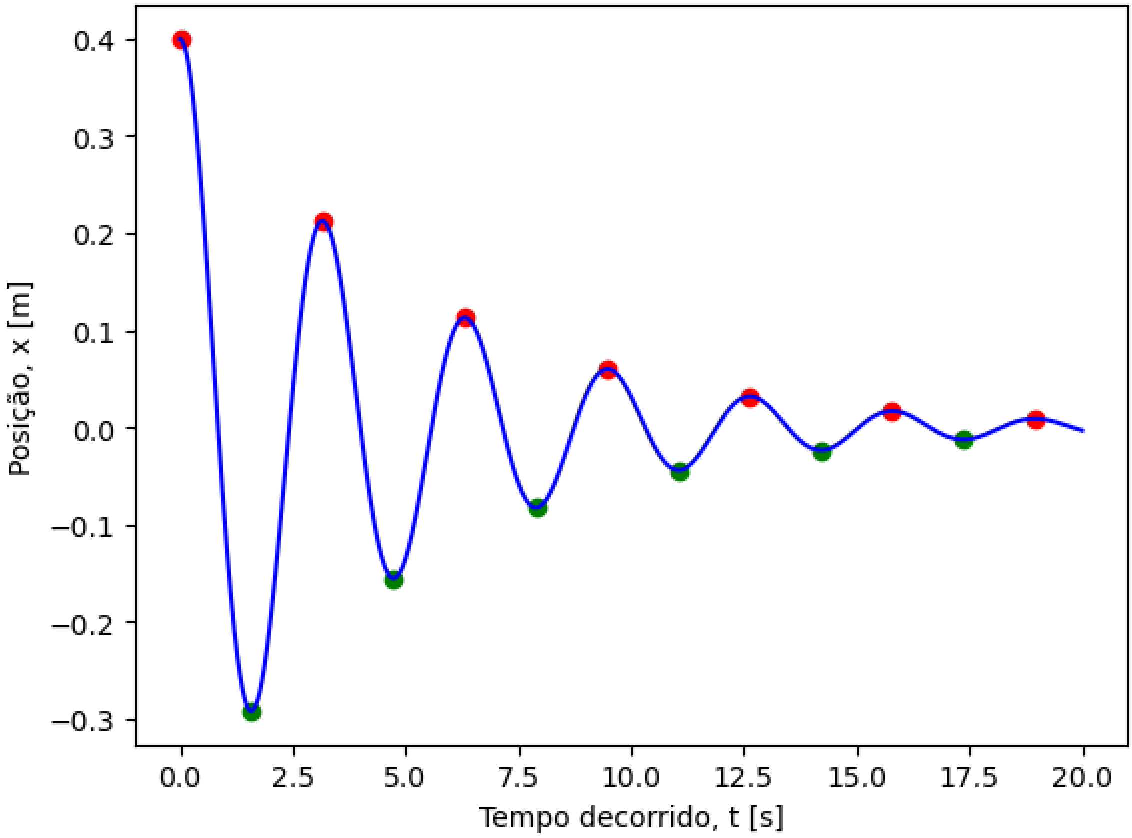
$$x_{i+1} = x_i + v_{i+1}\,\delta\tau$$

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

t0 = 0.0                          # condição inicial, tempo [s]
tf = 20.0                         # limite do domínio, tempo final [s]
dt = 0.001                        # passo [s]

x0 = 0.4                          # condição inicial, posição inicial [m]
v0 = 0.0                          # condição inicial, velocidade inicial [m/s]

m = 0.25                          # massa [kg]
k = 1.0                           # constante da mola [N/m]
b = 0.1                           # constante de amortecimento [kg/s]

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

# inicializar solução
a = np.zeros(np.size(t))          # aceleração [m/s2]
v = np.zeros(np.size(t))          # velocidade [m/s]
x = np.zeros(np.size(t))          # posição [m]
x[0] = x0
v[0] = v0

# método de Euler-Cromer
for i in range(np.size(t) - 1):
    a[i] = - (k * x[i] + b * v[i]) / m
    v[i + 1] = v[i] + a[i] * dt
    x[i + 1] = x[i] + v[i + 1] * dt

plt.plot(t, x, 'b-')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("Posição, x [m]")
plt.show()
```



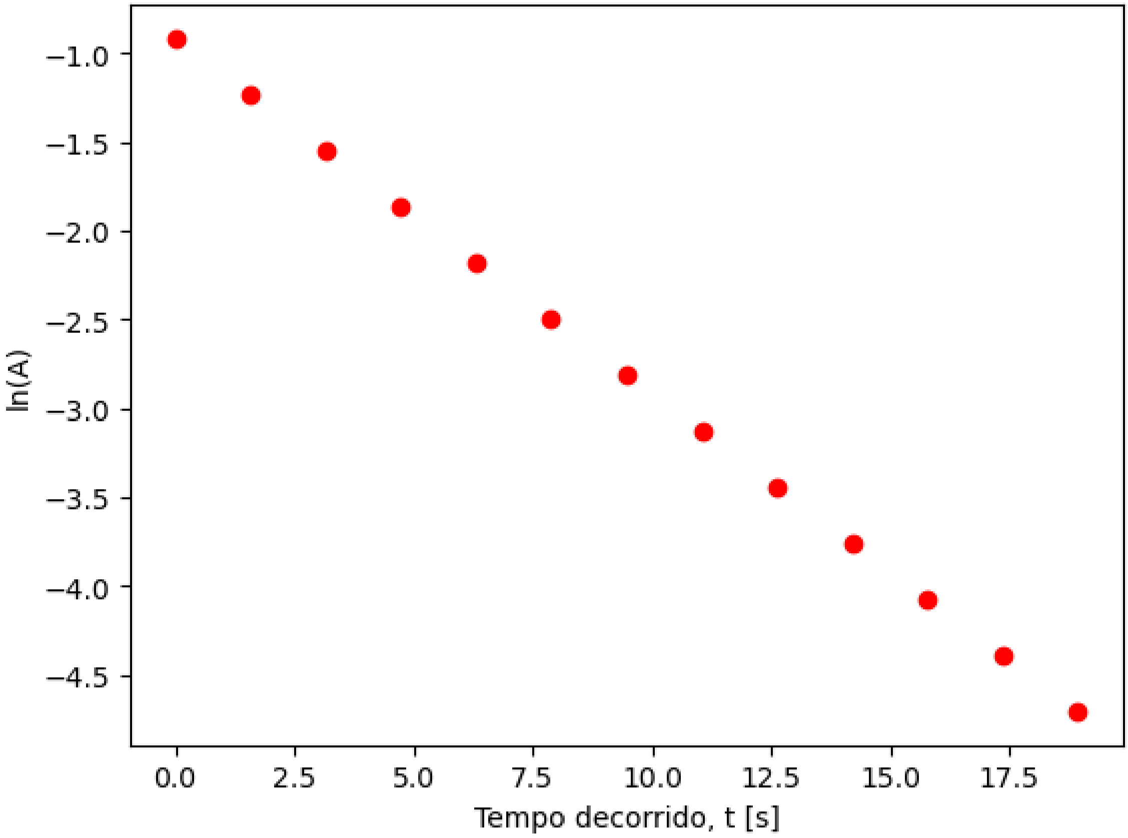
b) Encontre os máximos e mínimos (locais) da posição usando a interpolação pelo polinómio de Lagrange.

In [2]:
```python
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, ymax
    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
    ymax = a * xtb * xtc + b * xta * xtc + c * xta * xtb
    return xm, ymax


# arrays com valores máximos, mínimos e respetivos instantes de tempo
t_max = np.array([t0])
x_max = np.array([x0])
t_min = np.array([])
x_min = np.array([])

# Pesquisar pelo máximos e mínimos de x.
# Aqui definimos uma "janela corrida" no tempo em passos de 2, i.e, analisamos
# os máximos/mínimos 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:
    # x[i], x[i+1], x[i+2] e respetivos instantes de tempo para i = 0, ..., N-3
    tm, xm = maxminv(t[i], t[i+1], t[i+2], x[i], x[i+1], x[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]:
        # verificar se é máximo
        if xm > np.maximum(x[i], x[i+2]):
            t_max = np.append(t_max, tm)
            x_max = np.append(x_max, xm)
        else:
            t_min = np.append(t_min, tm)
            x_min = np.append(x_min, xm)

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



c) Faça um plot do logaritmo das amplitudes obtidas em b), em função do tempo, e encontre um ajuste linear. Qual é a dependência entre a amplitude o tempo? Qual é o declive e o seu erro? Concorda com a teoria?

In [3]:
```
# agregamos os instantes de tempo correspondentes aos "extremos" = max + min
t_ext = np.append(t_min, t_max)

# consideramos as amplitudes como valores positivos dos extremos
A_ext = np.abs(np.append(x_min, x_max))

plt.scatter(t_ext, np.log(A_ext), color='red')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("ln(A)")
plt.show()
```



Vamos agora usar a função do `polyfit` do numpy para realizarmos um ajuste de uma reta aos dados pelo método dos mínimos quadrados. Aproveitamos também para obter a matrix das covariâncias de forma a obtermos o erro associado aos parâmetros ajustados. O erro associado ao coeficiente p_i do polinómio $p(x) = \sum_i p_i x^i$ é dado por,

$$\delta p_i = \sqrt{C_{ii}}$$

onde C_{ii} é o elemento diagonal matrix da convariância associado ao coeficiente p_i. Para mais detalhes, ver a demonstração em https://ipnpr.jpl.nasa.gov/progress_report/42-122/122E.pdf

Verificamos que o oscilador é "fracamente amortecido" porque $b < 2mk$. As amplitudes sucessivas são dadas por,

$$A(t) = A_0 \exp\left(-\frac{b}{2m}t\right)$$

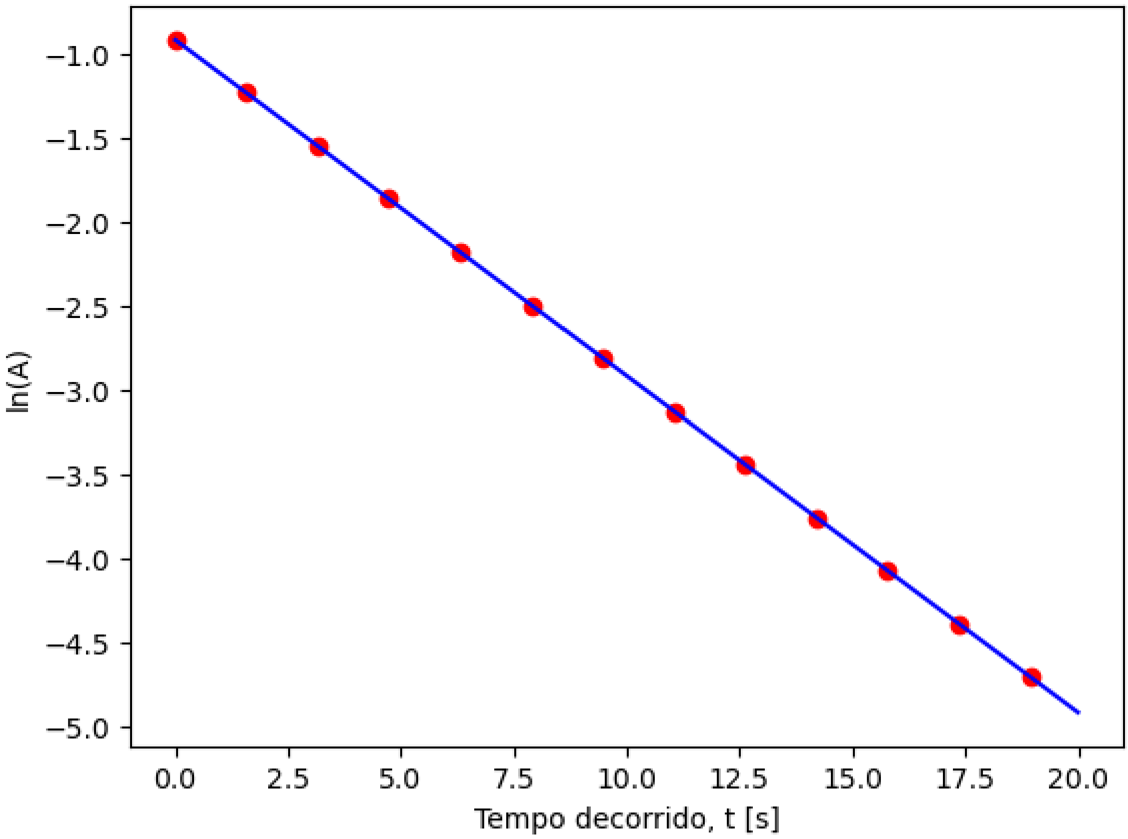
ou seja,

$$\ln(A) = -\frac{b}{2m}t + \ln(A_0)$$

de onde podemos concluir que se ajustarmos uma reta aos pontos, o declive da reta será $-b/2m$.

In [4]:
```
# obter parâmetros 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
p, C = np.polyfit(t_ext, np.log(A_ext), 1, cov='unscaled')

plt.plot(t, p[0] * t + p[1], 'b-')
plt.scatter(t_ext, np.log(A_ext), color='red')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("ln(A)")
plt.show()

# p[0] é o declive, p[1] é a ordenada na orígem
```

```
print("-b/2m = ", -b/(2*m))
print("p[0] = {0:.2f} ± {1:.2f}".format(p[0] , np.sqrt(C[0,0])))
```




```
-b/2m =  -0.2
p[0] = -0.20 ± 0.05
```

Sim. O resultado teórico corresponde ao resultado numérico.

Problemas do Capítulo 7: Oscilador harmónico forçado

Um corpo de massa $m = 1\ \mathrm{kg}$ move-se num oscilador harmónico forçado. Se a posição de equilíbrio for a origem do eixo, $x_{\mathrm{eq}} = 0\ \mathrm{m}$, o oscilador harmónico tem uma energia potencial $E_{\mathrm{p}} = \frac{1}{2}kx^2$, correspondente a uma força que exerce no corpo,

$$F_{\mathrm{mola}} = -kx.$$

O oscilador é amortecido pela força $-bv$ e sujeito a uma força externa $F_{\mathrm{ext}} = F_0 \cos(\omega_{\mathrm{f}}\, t)$, onde v é a velocidade instantânea e,

- $k = 1\ \mathrm{N/m}$
- $b = 0.05\ \mathrm{kg/s}$
- $F_0 = 7.5\ \mathrm{N}$
- $\omega_{\mathrm{f}} = 0.5\ \mathrm{rad/s}$

a) Calcule numericamente a lei do movimento, assumindo a velocidade inicial nula e a posição inicial 4 m

Podemos usar o método de Euler-Cromer (adequado para movimentos oscilatórios) para descrever a dinâmica do movimento da massa.

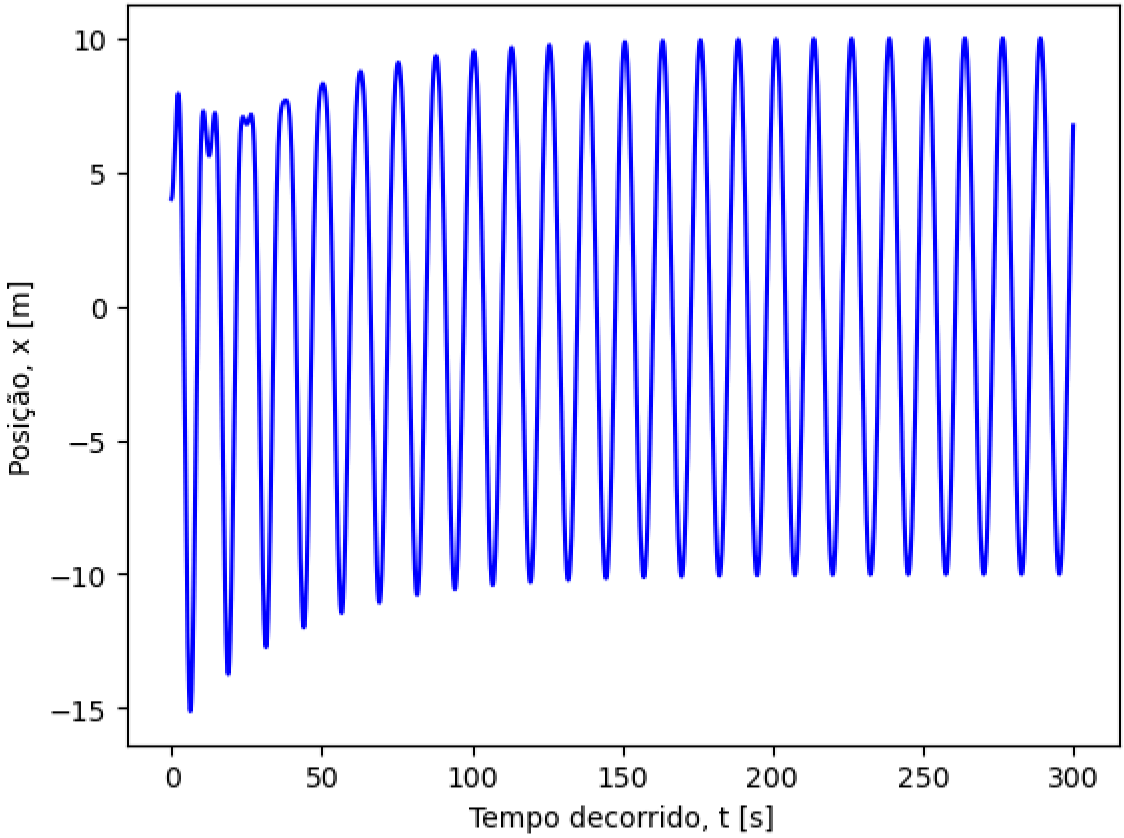
Sendo força resultante dada por,

$$F = -kx - bv + F_0 \cos(\omega_{\mathrm{f}}\, t),$$

ou seja, a aceleração instantânea é

$$a = -\left[kx + bv - F_0 \cos(\omega_{\mathrm{f}}\, t)\right]/m,$$

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt

t0 = 0.0                          # condição inicial, tempo [s]
tf = 300.0                         # limite do domínio, tempo final [s]
dt = 0.001                        # passo [s]

x0 = 4.0                          # condição inicial, posição inicial [m]
v0 = 0.0                          # condição inicial, velocidade inicial [m/s]

m = 1.0                           # massa [kg]
k = 1.0                           # constante da mola [N/m]
b = 0.05                          # constante de amortecimento [kg/s]
F_0 = 7.5                         # amplitude da força externa [N]
ω_f = 0.5                         # frequência angular da força externa [rad/s]

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

# inicializar solução
a = np.zeros(np.size(t))          # aceleração [m/s2]
v = np.zeros(np.size(t))          # velocidade [m/s]
x = np.zeros(np.size(t))          # posição [m]
x[0] = x0
v[0] = v0

# método de Euler-Cromer
for i in range(np.size(t) - 1):
    a[i] = - (k * x[i] + b * v[i] - F_0 * np.cos(ω_f * t[i])) / m
    v[i + 1] = v[i] + a[i] * dt
    x[i + 1] = x[i] + v[i+1] * dt

plt.plot(t, x, 'b-')
plt.xlabel("Tempo decorrido, t [s]")
plt.ylabel("Posição, x [m]")
plt.show()
```



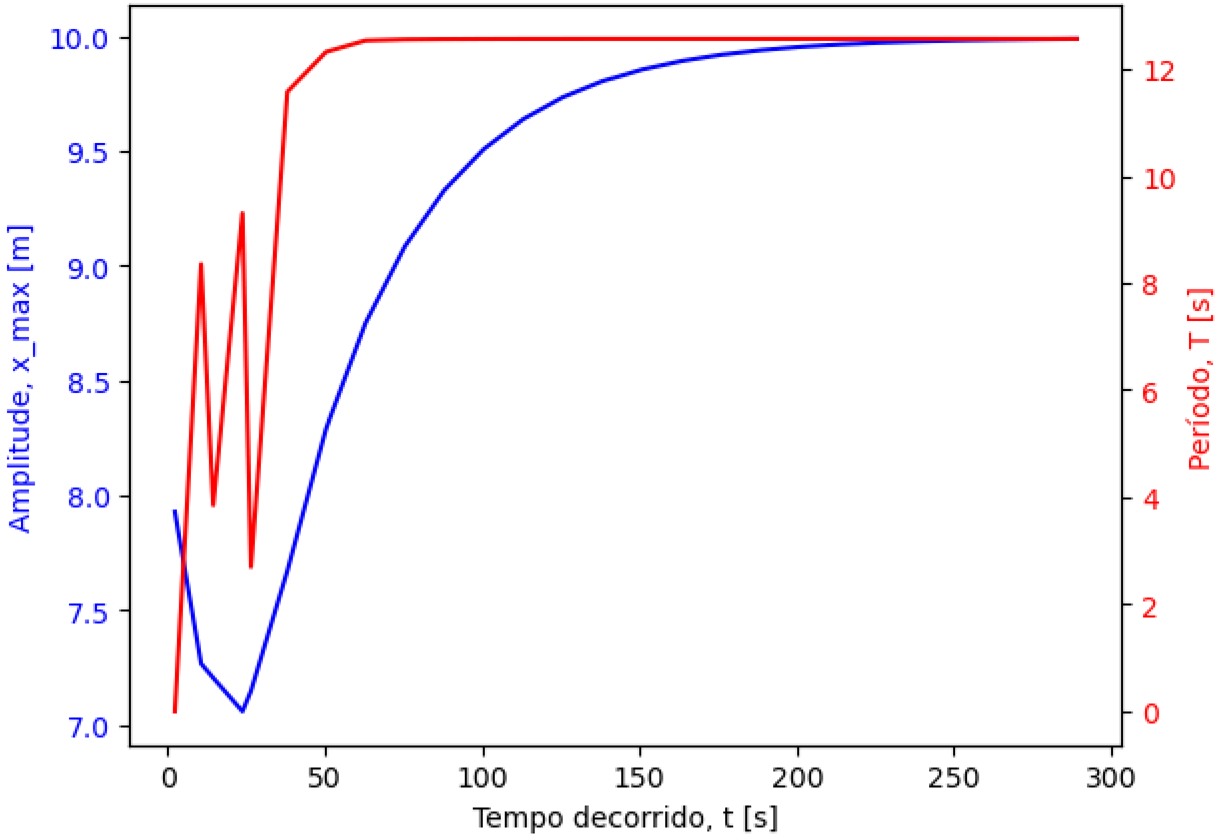
b) Calcule a amplitude do movimento e o seu período no regime estacionário, usando os resultados numéricos.

Aqui vamos novamente utilizar a função `maxminv` definita acima para calcular os vários máximos consecutivos. Daí poderemos obter a amplitude em regime estacionário e o respetivo período.

Aproveitamos para representar graficamente a posição e o período de forma a verificar quando atingimos o estado estacionário.

In [6]:
```
# arrays com valores máximos, respetivos instantes de tempo, e o período
t_max = np.array([])   # instante de tempo nos máximos
x_max = np.array([])   # máximos de amplitude
T = np.array([])       # período entre máximos

# Pesquisar pelo máximos de x.
# Aqui definimos uma "janela corrida" no tempo em passos de 2, i.e, analisamos
# os máximos 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:
    # x[i], x[i+1], x[i+2] e respetivos instantes de tempo para i = 0, ..., N-3
    tm, xm = maxminv(t[i], t[i+1], t[i+2], x[i], x[i+1], x[i+2])

    # verificar se extremo está dentro da "janela corrida" (t[i] <-> t[i+2])
    if t[i] < tm and tm < t[i+2]:
        # verificar se é máximo e adicionar a lista se esse for o caso
        if xm > np.maximum(x[i], x[i+2]):
            t_max = np.append(t_max, tm)
            x_max = np.append(x_max, xm)
            # calcular diferenca entre os dois ultimos instantes de tempo
            # e adicionar ao array dos periodos
            T = np.append(T, t_max[np.size(t_max) - 1] - t_max[np.size(t_max) - 2])


fig, ax1 = plt.subplots()
ax1.set_xlabel('Tempo decorrido, t [s]')
ax1.set_ylabel('Amplitude, x_max [m]', color='blue')
```

```
ax1.tick_params(axis='y', labelcolor='blue')
ax1.plot(t_max, x_max, 'b-')

ax2 = ax1.twinx()  # criar segundo sistema de eixos com o mesmo eixo Ox
ax2.set_ylabel('Período, T [s]', color='red')
ax2.plot(t_max, T, 'r-')
ax2.tick_params(axis='y', labelcolor='red')

plt.show()
```

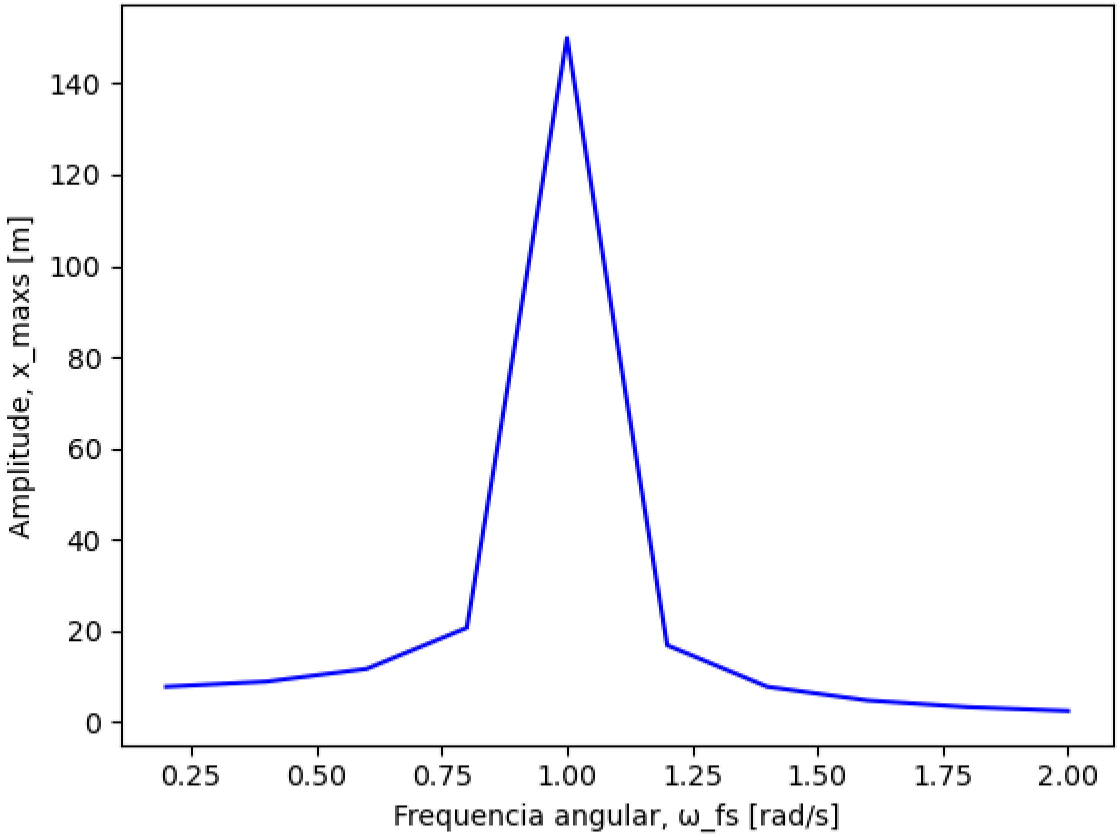


In [7]:
```
print("O período em regime estacionário é T = {0:.2f} s".format(T[-1]))
print("A amplitude em regime estacionário é x_max = {0:.2f} m".format(x_max[-1]))
```

```
O período em regime estacionário é T = 12.57 s
A amplitude em regime estacionário é x_max = 9.99 m
```

c) Repita para valores de ω_f entre 0.2 e 2 rad/s. Faça um gráfico da amplitude em regime estacionário em função de ω_f. Qual a frequência angular ω_f que corresponde à maior amplitude?

In [8]:
```
ω_fs  = np.array([0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0])
x_maxs = np.array([7.81, 8.93, 11.71, 20.71, 149.91, 16.9, 7.79, 4.80, 3.34, 2.50])

plt.plot(ω_fs, x_maxs, 'b-')
plt.xlabel("Frequencia angular, ω_fs [rad/s]")
plt.ylabel("Amplitude, x_maxs [m]")
plt.show()
```



A amplitude máxima em regime estacionário corresponde à frequência angular $\omega_\mathrm{f} = 1\ \mathrm{rad/s}$. Este valor corresponde à frequência angular natural da mola,
$\omega_0 = \sqrt{k/m} = \sqrt{(1\ \mathrm{N/m})/(1\ \mathrm{kg})} = 1\ \mathrm{rad/s}$. A este fenómono chamamos "resonância".