Modelação de Sistemas Físicos - Aula Prática nº5

Realização e resolução de problemas vetoriais e a sua representação gráfica

Lista de operações, funções e métodos úteis (numpy)

Descrição	Exemplo	Equiv. Matemático
Criação	`a = np.array([1, 2, 3])`	$\mathbf{a} = (1, 2, 3)$
Operações elemento-a-elemento	`b = np.ones(3) + 1`	$b_i = 1_i + 1 = (2, 2, 2)$
	`c = a * b`	$c_i = a_i b_i = (2, 4, 6)$
	`c = a ** b`	$c_i = a_i^{b_i} = (1, 4, 9)$
Produto interno (escalar)	`c = np.inner(a, b)`	$\mathbf{c} = \mathbf{a} \cdot \mathbf{b} = 2 + 4 + 6 = 12$
Produto externo (vetorial)	`c = np.cross(a, b)`	$\mathbf{c} = \mathbf{a} \times \mathbf{b} = (-2, 4, -2)$
Soma dos elementos	`c = np.sum(a)`	$c = \sum_i a_i = 6$
Módulo vetorial	`c = np.linalg.norm(a)`	$c = \lvert\mathbf{a}\rvert = \sqrt{\sum_i a_i^2}$

Exercício 1: Vetores de um robô

Um simples robô pode deslocar-se no chão executando dois tipos de instruções. Pode rodar no sentido horário por um determinado ângulo, e pode avançar em linha reta uma determinada distância.

As instruções são dados ao robô na forma de *tuples* `(ang, dist)` , que significa que o robô deve rodar por um ângulo `ang` (em graus) e depois avançar uma distância `dist` (metros).

O robô começa na origem, orientado ao longo do eixo x. É-lhe dada a seguinte sequência de instruções:

(45,3), (90,2), (45,3), (45,2), (90,3)

a) Calcule a posição do robô após cada passo. Faça um gráfico da trajetória do robô.

Use a função `arrow()` do `matplotlib.pyplot` para representar graficamente o vetor para cada passo no movimento.

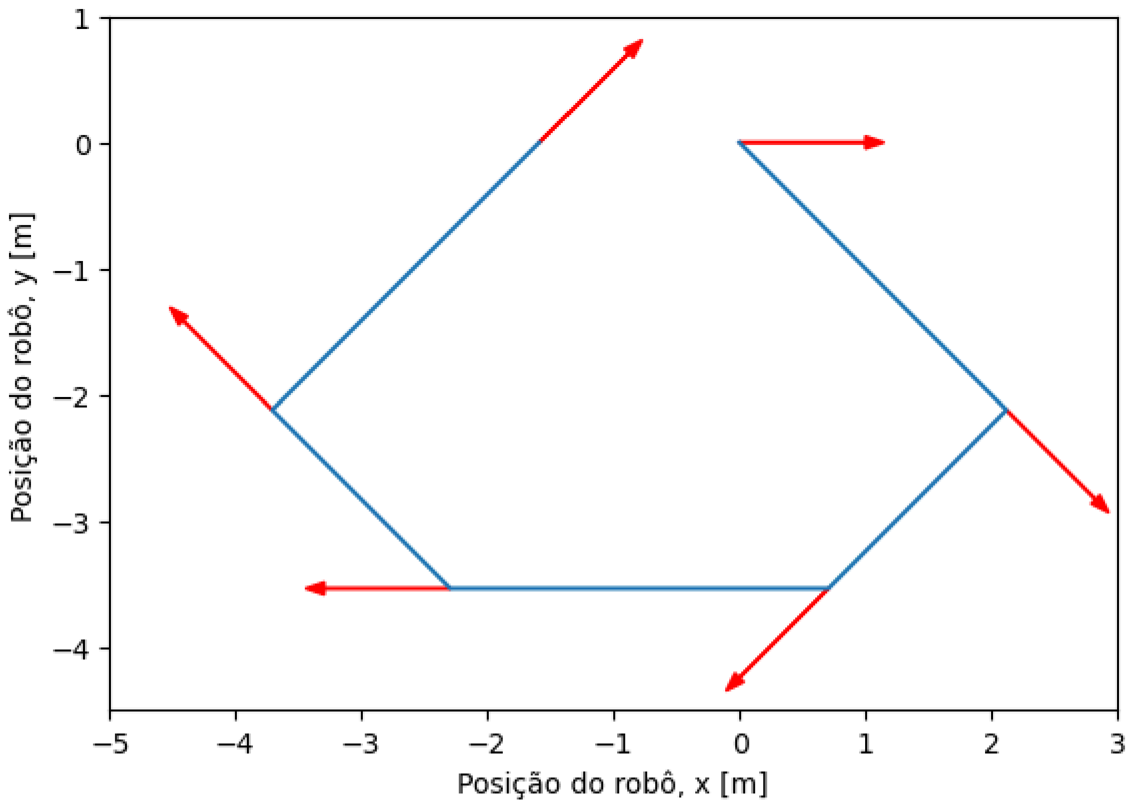
Exemplo do uso de `arrow()` :

```
import matplotlib.pyplot as plt
plt.arrow(x0,y0,dx,dy,color='r',width=0.1,length_includes_head=True)
```

In [1]:

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

plt.figure(figsize=(6.5,4.5))       # Preparar a representação gráfica
plt.axis([-5, 3, -4.5, 1.0])

x = 0.0; y = 0.0; theta = 0.0
pos = np.array([[x, y, theta]])     # posicionamento. Robô inicia na orígem com ang=0
plt.arrow(x, y, np.cos(-theta * np.pi / 180), np.sin(-theta * np.pi / 180), color='r'
          width=0.01, head_width=0.1)  # representar direção do robô

ang = 45; dist = 3                  # instrução 1
theta += ang
x += dist * np.cos(-theta * np.pi / 180)
y += dist * np.sin(-theta * np.pi / 180)
pos = np.append(pos, [[x, y, theta]], 0)
plt.arrow(x, y, np.cos(-theta * np.pi / 180), np.sin(-theta * np.pi / 180), color='r'
          width=0.01, head_width=0.1)  # representar direção do robô

ang = 90; dist = 2                  # instrução 2
theta += ang
x += dist * np.cos(-theta * np.pi / 180)
y += dist * np.sin(-theta * np.pi / 180)
pos = np.append(pos, [[x, y, theta]], 0)
plt.arrow(x, y, np.cos(-theta * np.pi / 180), np.sin(-theta * np.pi / 180), color='r'
          width=0.01, head_width=0.1)  # representar direção do robô

ang = 45; dist = 3                  # instrução 3
theta += ang
x += dist * np.cos(-theta * np.pi / 180)
y += dist * np.sin(-theta * np.pi / 180)
pos = np.append(pos, [[x, y, theta]], 0)
plt.arrow(x, y, np.cos(-theta * np.pi / 180), np.sin(-theta * np.pi / 180), color='r'
          width=0.01, head_width=0.1)  # representar direção do robô

ang = 45; dist = 2                  # instrução 4
theta += ang
x += dist * np.cos(-theta * np.pi / 180)
y += dist * np.sin(-theta * np.pi / 180)
pos = np.append(pos, [[x, y, theta]], 0)
plt.arrow(x, y, np.cos(-theta * np.pi / 180), np.sin(-theta * np.pi / 180), color='r'
          width=0.01, head_width=0.1)  # representar direção do robô

ang = 90; dist = 3                  # instrução 5
theta += ang
x += dist * np.cos(-theta * np.pi / 180)
y += dist * np.sin(-theta * np.pi / 180)
pos = np.append(pos, [[x, y, theta]], 0)
plt.arrow(x, y, np.cos(-theta * np.pi / 180), np.sin(-theta * np.pi / 180), color='r'
          width=0.01, head_width=0.1)  # representar direção do robô

plt.plot(pos[:,0], pos[:,1])
plt.xlabel("Posição do robô, x [m]")
```

```
plt.ylabel("Posição do robô, y [m]")
plt.show()
```



b) Quais são as coordenadas finais do robô?

In [2]:
```
x_f = x; y_f = y; theta_f = theta
print("As coordenadas finais do robô são:")
print("    r = ({0:.2f}, ".format(x_f), "{0:.2f})".format(y_f))
print("    ang = {0:.2f}".format(theta_f))
```

```
As coordenadas finais do robô são:
    r = (-1.59,  -0.00)
    ang = 315.00
```

c) Qual é a instrução necessária para fazer o robô retornar ao ponto inicial?

Considerando que $\mathbf{p}_\mathrm{f} = (\mathbf{r}_\mathrm{f},\, \theta_\mathrm{f})$ são as coordenadas atuais do robô, onde $\mathbf{r}_\mathrm{f} = (x_\mathrm{f},\, y_\mathrm{f})$.

Então, a intrução para o retorno ao início consite rodar na direção da orígem das coordenadas, e de seguida avançar ao longo de um vetor $-\mathbf{r}_\mathrm{f}$, ou seja

1. Orientar o robô ao longo da direção $\theta = \arcsin(-y_\mathrm{f}/r_\mathrm{f})$
2. Percorrer uma distância $d = r_\mathrm{f}$.

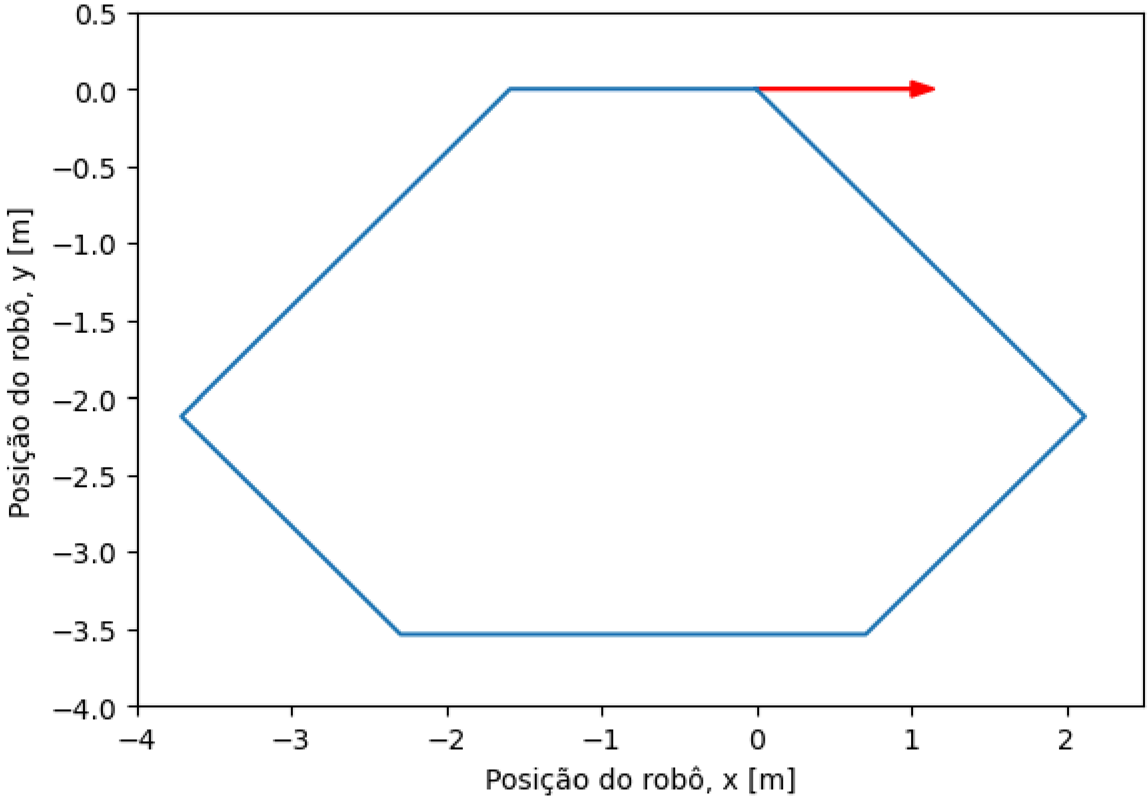
Considerando que o último ângulo (orientação do robô) era θ_f, o robô terá primeiro que rodar $\delta\theta = \arcsin(-y_\mathrm{f}/r_\mathrm{f}) - \theta_\mathrm{f}$ e depois avançar a distância d.

Note-se que $\delta\theta \geq 0$ (robô só roda no sentido dos ponteiros do relógio), e portanto $\delta\theta$ é o menor positivo tal que $\delta\theta = \arcsin(-y_\mathrm{f}/r_\mathrm{f}) - \theta_\mathrm{f} + 360n$, em que n é um número inteiro. Esta última operação pode ser efetuada com a função `numpy.remainder()`

In [3]:
```
d = np.sqrt(x_f**2 + y_f**2)
```

```python
plt.figure(figsize=(6.5,4.5))
plt.axis([-4, 2.5, -4, 0.5])

#ang = np.arcsin(-y_f / d) - theta_f                   # 1. Rotação <= 360 graus
ang = np.remainder(np.arcsin(-y_f / d) - theta_f, 360)  # 1. Rotação ângulo arbitrár
dist = d                                                # 2. Avanço
theta += ang
x += dist * np.cos(-theta * np.pi / 180)
y += dist * np.sin(-theta * np.pi / 180)
pos = np.append(pos, [[x, y, theta]], 0)
print("Instrução de retorno:")
print("    dist = ({0:.2f}, ".format(dist))
print("    ang = {0:.2f}".format(ang))
plt.arrow(x, y, np.cos(-theta * np.pi / 180), np.sin(-theta * np.pi / 180), color='r'

plt.plot(pos[:,0], pos[:,1])
plt.xlabel("Posição do robô, x [m]")
plt.ylabel("Posição do robô, y [m]")
plt.show()
```

```
Instrução de retorno:
    dist = (1.59,
    ang = 45.00
```



Pergunta 1:

Qual é a soma dos ângulos que o robô rodou para regressar à origem? Em geral, que rotação seria necessária para que o robô voltasse a ter a sua orientação original?

Exercício 2: Bola de futebol com rotação (movimento a 3D)

Uma bola de futebol é pontapeada de modo a rodar sobre si própria, o que adiciona a força de Magnus às outras forças existêntes. A força de Magnus resulta do escoamento do ar ser diferente nos lados diametralmente opostos da bola, e é dada por:

$$\mathbf{F}_{\mathrm{mag}} = \frac{1}{2} A\, \rho_{\mathrm{ar}}\, r\, \boldsymbol{\omega} \times \mathbf{v},$$

onde $A = \pi r^2$ é a área de secção de corte da bola com raio r, ρ_{ar} a massa volúmica do ar, $\boldsymbol{\omega}$ a velocidade angular da bola, e $\mathbf{v}$ a sua velocidade instantânia. As duas últimas grandezas são vetoriais.

No caso de uma bola de futebol pontapeada, à força de Magnus, adiciona-se a força gravítica e a força da resistência do ar para se obter a força resultante.

A força gravítica é dada por $\mathbf{F}_{\mathrm{grav}} = -mg\mathbf{j}$.

A força de atrito é dada por $\mathbf{F}_{\mathrm{attr}} = -mD\mathbf{v}|\mathbf{v}| = -mD\mathbf{v}v$.

Determinar se é golo ou não, após a bola ser chutada do canto com rotação. Assuma que a origem das coordenadas é a meio da linha de golo, *i.e.* exatamente a meio da linha que une os dois postes da baliza. Implementar o movimento da bola com rotação, usando o método de Euler.

Os dados do problema são:

- $\mathbf{r}_0 = (x_0, y_0, z_0) = (0, 0, 23.8)\ \mathrm{m}$
- $\mathbf{v}_0 = (v_0, v_0, v_0) = (25, 5, -50)\ \mathrm{m/s}$
- $\boldsymbol{\omega} = (\omega_0, \omega_0, \omega_0) = (0, 390, 0)\ \mathrm{rad/s}$
- $t_0 = 0\ \mathrm{m}$
- Massa da bola, $m = 0.45\ \mathrm{kg}$
- Raio da bola, $r = 11\ \mathrm{cm}$
- Secção de corte da bola, $A = \pi r^2$
- Densidade do ar, $\rho_{\mathrm{ar}} = 1.225\ \mathrm{kg/m^3}$
- Velocidade terminal, $v_{\mathrm{T}} = 100\ \mathrm{km/h}$ (não aparece no enunciado - é assumido)
- Coeficiente de atrito, $D = g/v_{\mathrm{T}}^2$.

David Beckham taking a corner.

Notando que $\boldsymbol{\omega}$ só tem componente ao longo da direção Oy, podemos simplificar a expressão da força de Magnus. O produto externo $\boldsymbol{\omega} \times \mathbf{v}$ é dado por

$$\boldsymbol{\omega} \times \mathbf{v} = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 0 & \omega_y & 0 \\ v_x & v_y & v_z \end{bmatrix} = \omega_y v_z \mathbf{i} - \omega_y v_x \mathbf{k}.$$

Portanto:

$$\mathbf{F} = -mg\,\hat{\mathbf{j}} - mD\mathbf{v}v + \frac{1}{2} A\, \rho_{\mathrm{ar}}\, r\, (\omega_y v_z \mathbf{i} - \omega_y v_x \mathbf{k})$$

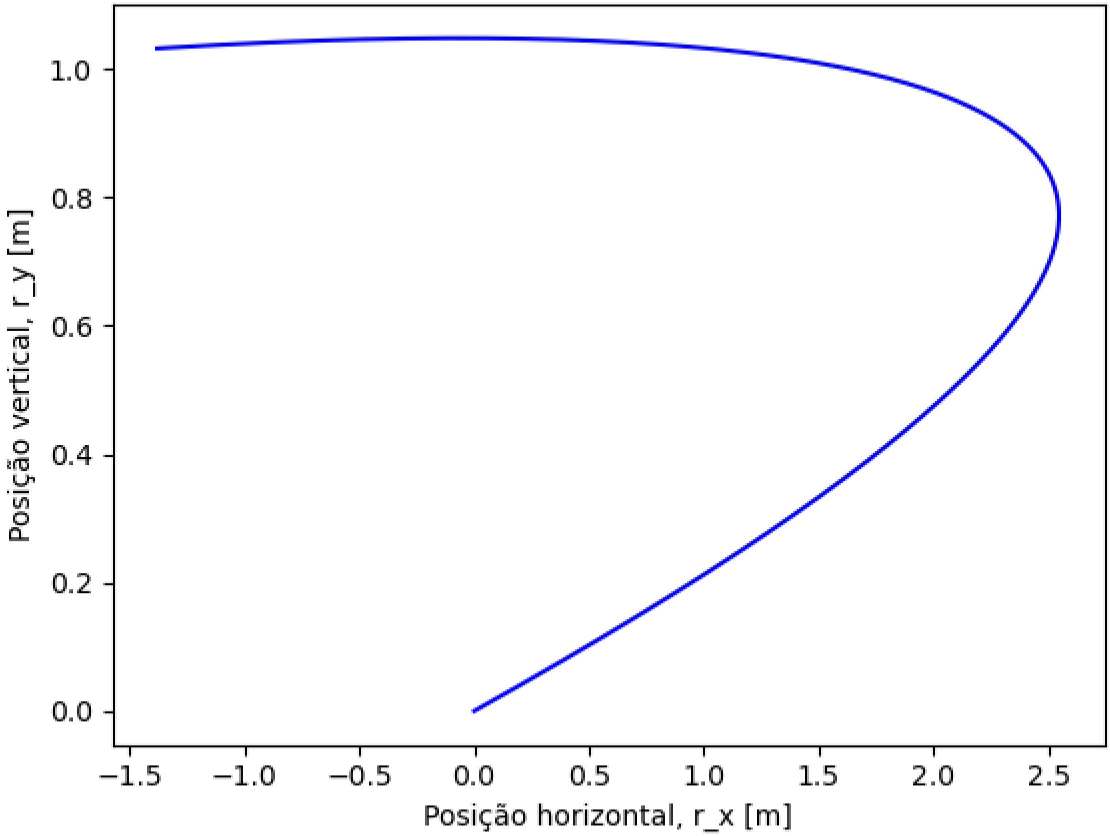
ou seja, as componentes da força resultante são dadas por,

$$\begin{cases} F_x &=& -mDv_x v + A\, \rho_{\mathrm{ar}}\, r\, \omega_y\, v_z/2 \\ F_y &=& -mg - mDv_y v \\ F_z &=& -mDv_z v - A\, \rho_{\mathrm{ar}}\, r\, \omega_y\, v_x/2 \end{cases}$$

Assim, a aceleração é dada por,

$$\begin{cases} a_x &= -Dv_x v + A\,\rho_{\text{ar}}\,r\,\omega_y\,v_z/2m \\ a_y &= -g - Dv_y v \\ a_z &= -Dv_z v - A\,\rho_{\text{ar}}\,r\,\omega_y\,v_x/2m \end{cases}$$

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

t0 = 0.0                          # condição inicial, tempo [s]
tf = 0.5                          # limite do domínio, tempo final [s]
dt = 0.001                        # passo [s]

r0 = np.array([0.0, 0.0, 23.8])   # condição inicial, velocidade inicial [m/s]
v0 = np.array([25.0, 5.0, -50.0]) # condição inicial, velocidade inicial [m/s]
w = 390.0                         # condição inicial, velocidade angular CONSTANTE [r

g = 9.8                     # aceleração gravítica [m/s^2]
R = 0.11                    # raio da bola [m]
A = np.pi * R ** 2          # área da secção da bola
m = 0.45                    # massa da bola [kg]
rho = 1.225                 # densidade do ar [kg/m^3]
v_T = 100 * 1000 / 3600     # velocidade terminal [m/s]
D = g / v_T ** 2

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

# inicializar solução, aceleração [m/s^2]
a = np.zeros([3, np.size(t)])

# inicializar solução, velocidade [m/s]
v = np.zeros([3, np.size(t)])
v[:,0] = v0

# inicializar solução, posição [m]
r = np.zeros([3, np.size(t)])
r[:,0] = r0


for i in range(np.size(t) - 1):
    # Caclular norma da velocidade, |v|
    v_norm = np.linalg.norm(v[:, i])
    a[0, i] = - D * v[0, i] * v_norm + A * rho * R * w * v[2,i] / (2 * m)
    a[1, i] = - g - D * v[1, i] * v_norm
    a[2, i] = - D * v[2, i] * v_norm - A * rho * R * w * v[0,i] / (2 * m)
    v[:, i + 1] = v[:, i] + a[:, i] * dt
    r[:, i + 1] = r[:, i] + v[:, i] * dt

plt.plot(r[0,:], r[1,:], 'b-')
plt.xlabel("Posição horizontal, r_x [m]")
plt.ylabel("Posição vertical, r_y [m]")
plt.show()
```



In [5]:
```
# indice e tempo para o qual a bola atinge a "linha de fundo", ie, instante de
# tempo a partir do qual a coordenada x da bola se torna negativa.
ixzero = np.size(r[0, r[0,:]>=0])   # usar indexação condicional
txzero = t[ixzero]
print("Tempo correspondente ao cruzamento da linha de fundo, txzero =", txzero, "s")
print("Coordenadas da bola quando cruza a linha de fundo:")
print("   x = ", r[0,ixzero], "m")
print("   y = ", r[1,ixzero], "m")
print("   z = ", r[2,ixzero], "m")
```

```
Tempo correspondente ao cruzamento da linha de fundo, txzero = 0.439 s
Coordenadas da bola quando cruza a linha de fundo:
   x =  -0.008868894618749651 m
   y =  1.0476821348636254 m
   z =  3.2642579535939813 m
```

Conlcusões:

- A bola cruza a linha de fundo abaixo da trave da baliza ($y = 1.05$ m comparado com 2.44 m da altura de uma baliza regulamentar);
- A bola cruza a linha de fundo a $z = 3.3$ m do centro da baliza, *i.e.*, entra na baliza. Este valor é inferior a metade da largura da baliza (7.32 m).

Portanto, se o guarda redes *não se esforçar muito*, será golo!!!

Pergunta 2:

A velocidade de rotação da bola deve ser aumentada ou diminuída para que a bola se aproxime mais do centro da baliza?