In [133…
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import control as ct
import os

PATH = os.getcwd()
PATH
```

Out[133…
```
'/home/andre/Documents/Git/CONTROLE_DCA3701.X/U3'
```

Q1

1- Considere o sistema definido como

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -5 & -6 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \mathrm{u}$$

$$\mathrm{y} = [1 \quad 0 \quad 0]\mathbf{x}$$

Utilizando o controle por realimentação de estado u= $\mathbf{Kx}$, desejamos ter os pólos de malha fechada em $s = -1 \pm j4$, $s = -10$. Determine a matriz de ganho K de realimentação de estado.

Simule o sistema com o controlador por realimentação de estado, mostrando os gráficos da entrada u, saída y e dos 3 estados.

In [134…
```python
A = np.array([
    [0,1,0],
    [0,0,1],
    [-1,-5,-6]
])

B = np.array([
    [0],
    [1],
    [1]
])

C = np.array([
    [1,0,0]
])

Pc = np.array([-1 + 4j, -1 - 4j, -10], dtype=np.complex64)
```

- Polinômio característico:

1. $\Delta(s) = (s - (-1 + j4)) \cdot (s) = (s - (-1 - j4)) \cdot (s - (-10))$

2. $\Delta(s) = (s + 1 - j4) \cdot (s + 1 + j4) \cdot (s + 10)$
3. $\Delta(s) = (s^2 + 2s + 17) \cdot (s + 10)$
4. $\Delta(s) = s^3 + 12s^2 + 37s + 170$

- Usando o metódo de Ackermann

In [135…

```
qc = np.linalg.matrix_power(A, 3) + 12*np.linalg.matrix_power(A, 2) + 37*A +
print(qc)
```

```
[[169.  32.   6.]
 [ -6. 139.  -4.]
 [  4.  14. 163.]]
```

In [136…

```
U = np.concatenate((B, A@B, np.linalg.matrix_power(A, 2)@B), axis=1)
print(U)

U_inversa = np.linalg.inv(U)
print(U_inversa)
```

```
[[  0   1   1]
 [  1   1 -11]
 [  1 -11  60]]
[[ 0.73493976  0.85542169  0.14457831]
 [ 0.85542169  0.01204819 -0.01204819]
 [ 0.14457831 -0.01204819  0.01204819]]
```

In [137…

```
K =  -np.array([0,0,1]) @ U_inversa @ qc
print(K)

K = -ct.acker(A,B,Pc)
print(K)
```

```
[-24.55421687  -3.12048193  -2.87951807]
[[-24.55421687  -3.12048193  -2.87951807]]
```

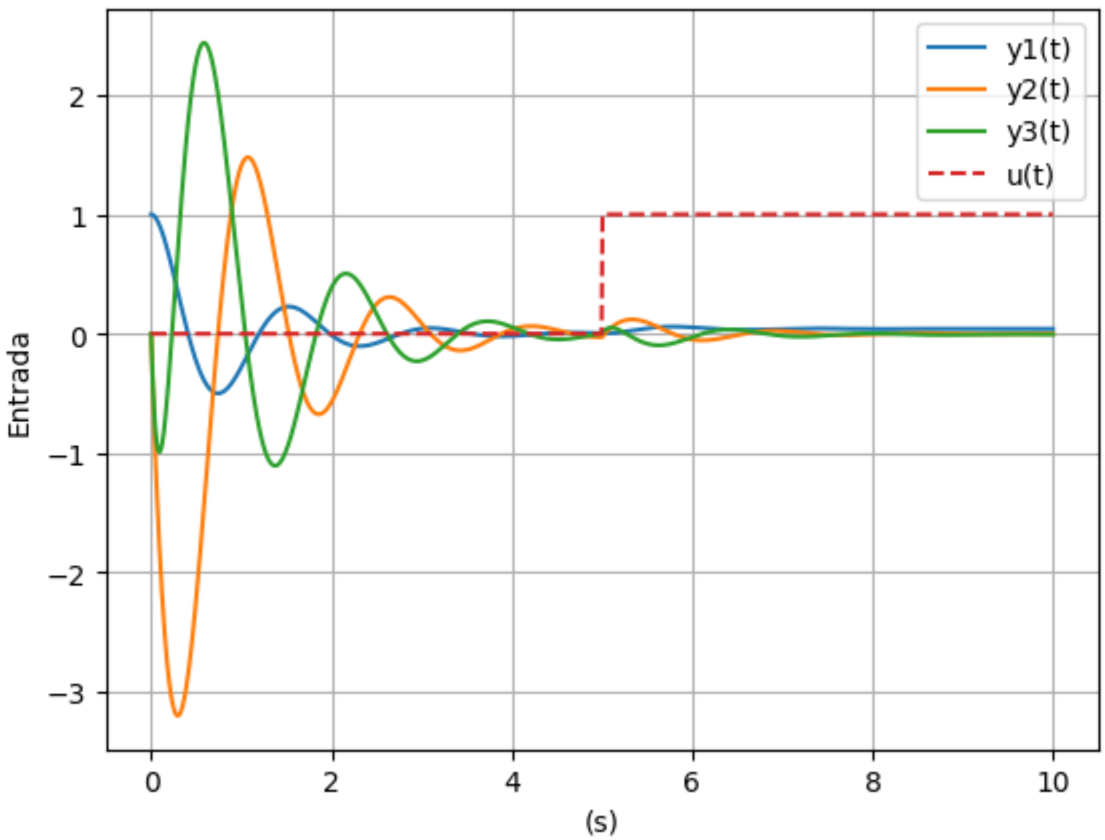
- Realimentação de estado

$$\begin{cases} \dot{\mathbf{x}} = (\mathbf{A} + \mathbf{BK})\mathbf{x} + \mathbf{B}r \\ y = \mathbf{Cx} \end{cases}$$

In [138…

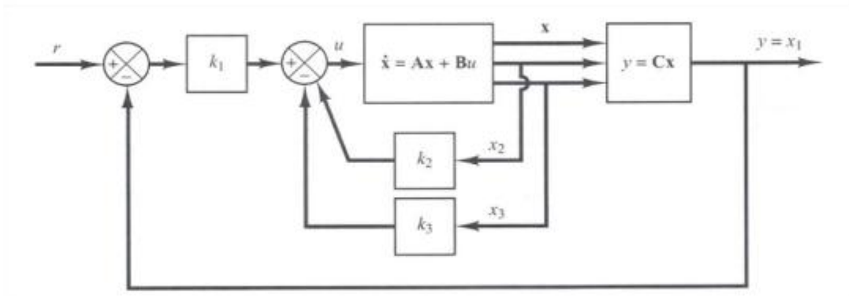
```
Aa = np.array([
    A + B@K
]).reshape(3,3)

Ba = np.array(B)
Ca = np.array(C)

x0 = np.array(Ca)

lent = 1000
t = np.linspace(0,10, lent)
u = np.ones(lent)
```

```python
u[0:int(lent/2)] = 0

system = signal.StateSpace(Aa, Ba, Ca, 0)
_, X,Y = signal.lsim(system, u, t, x0)

y1 = np.zeros(lent)
y2 = np.zeros(lent)
y3 = np.zeros(lent)

for i in range(lent):
    y1[i] = Y[i][0]
    y2[i] = Y[i][1]
    y3[i] = Y[i][2]

plt.plot(t, y1, label='y1(t)')
plt.plot(t, y2, label='y2(t)')
plt.plot(t, y3, label='y3(t)')
plt.plot(t, u, label='u(t)', linestyle='--')

plt.legend()
plt.grid()

plt.xlabel('(s)')
plt.ylabel('Entrada')

plt.savefig("grafico_q1.png")

plt.show()
```



Q2

2- Considere o servossistema da figura abaixo



$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -5 & -6 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

$$y = [1 \quad 0 \quad 0]\mathbf{x}$$

Determine as constantes de ganho de realimentação k_1, k_2 e k_3, de modo que os pólos de malha fechada estejam localizados em $s = -2 \pm j4$, $s = -10$

Simule o sistema com o controlador por realimentação de estado, mostrando os gráficos da entrada u, saída y e dos 3 estados.

In [139…

```
A = np.array([
    [0,1,0],
    [0,0,1],
    [0,-5,-6]
])

B = np.array([
    [0],
    [0],
    [1]
])

C = np.array([
    [1,0,0]
])

Pc = np.array([-2 + 4j, -2 - 4j, -10], dtype=np.complex64)
```

- Polinômio Característico

1. $\Delta(s) = (s - (-2 + 4j))(s - (-2 - 4j))(s - (-10))$
2. $\Delta(s) = (s + 2 - 4j)(s + 2 + 4j)(s + 10)$
3. $\Delta(s) = (s^2 + 4s + 20)(s + 10)$

4. $\Delta(s) = s^3 + 14s^2 + 60s + 200$

- Metódo de Ackermann para achar os coeficientes K's

In [140…
```
qc = np.linalg.matrix_power(A, 3) + 14*np.linalg.matrix_power(A, 2) + 60 * A
print(qc)
```

```
[[200.  55.   8.]
 [  0. 160.   7.]
 [  0. -35. 118.]]
```

In [141…
```
U = np.concatenate((B, A@B, np.linalg.matrix_power(A, 2)@B), axis=1)
print(U)

U_inversa = np.linalg.inv(U)
print(U_inversa)
```

```
[[ 0  0  1]
 [ 0  1 -6]
 [ 1 -6 31]]
[[5. 6. 1.]
 [6. 1. 0.]
 [1. 0. 0.]]
```

In [142…
```
K = -np.array([0,0,1]) @ U_inversa @ qc
print(K)

K = -ct.acker(A,B,Pc)
print(K)
```

```
[-200.  -55.   -8.]
[[-200.  -55.   -8.]]
```

- Simulação

In [143…
```
Aa = np.array([
    A + B@K
]).reshape(3,3)

Ba = np.array(B)
Ca = np.array(C)

x0 = np.array(Ca)

lent = 1000
t = np.linspace(0,10, lent)
u = np.ones(lent)

u[0:int(lent/2)] = 0

system = signal.StateSpace(Aa, Ba, Ca, 0)
_, X,Y = signal.lsim(system, u, t, x0)

y1 = np.zeros(lent)
```

```
y2 = np.zeros(lent)
y3 = np.zeros(lent)

for i in range(lent):
    y1[i] = Y[i][0]
    y2[i] = Y[i][1]
    y3[i] = Y[i][2]

plt.plot(t, y1, label='y1(t)')
plt.plot(t, y2, label='y2(t)')
plt.plot(t, y3, label='y3(t)')
plt.plot(t, u, label='u(t)', linestyle='--')

plt.legend()
plt.grid()

plt.xlabel('(s)')
plt.ylabel('Entrada')

plt.savefig("grafico_q1.png")

plt.show()
```