Lista 2

Luiz Georg 15/0041390

September 23, 2021

Setup

[1]:
```
import numpy as np
import control


def show_svg(path):
    from IPython.display import SVG, display

    display(SVG(filename=path))
    return


np.set_printoptions(precision=4)
```

Condições do problema

[2]:
```
# Modelo da Planta
# x_ = A @ x + B @ d

A = np.array([[-4.1172, 0.7781, 0.0], [-33.8836, -3.5729, 0.0], [0.0, 1.0, 0.0]])

B = np.array([[0.5435, -39.0847, 0.0]]).T

x0 = np.array([[0.5, 0, -0.1]]).T

print("A:")
print(A)
print()

print("B:")
print(B)
print()

print("x0:")
print(x0)
```

```
A:
[[ -4.1172   0.7781   0.    ]
 [-33.8836  -3.5729   0.    ]
 [  0.       1.       0.    ]]

B:
[[  0.5435]
 [-39.0847]
 [  0.    ]]
```

```
x0:
[[ 0.5]
 [ 0. ]
 [-0.1]]
```

1 Quais são os autovalores de A? O sistema é estável, instável ou marginalmente estável? Subamortecido ou superamortecido?

[3]:
```
# Autovalores de A
eigvals = np.linalg.eigvals(A)
print("polos:")
print(np.reshape(eigvals, (3, 1)))
```

```
polos:
[[ 0.     +0.j     ]
 [-3.8451+5.1275j]
 [-3.8451-5.1275j]]
```

Sistema marginalmente estável (2 polos no semiplano negativo e um polo na origem);
Sistema subamortecido (par de polos complexos).

2 O sistema é controlável ou não? Justifique.

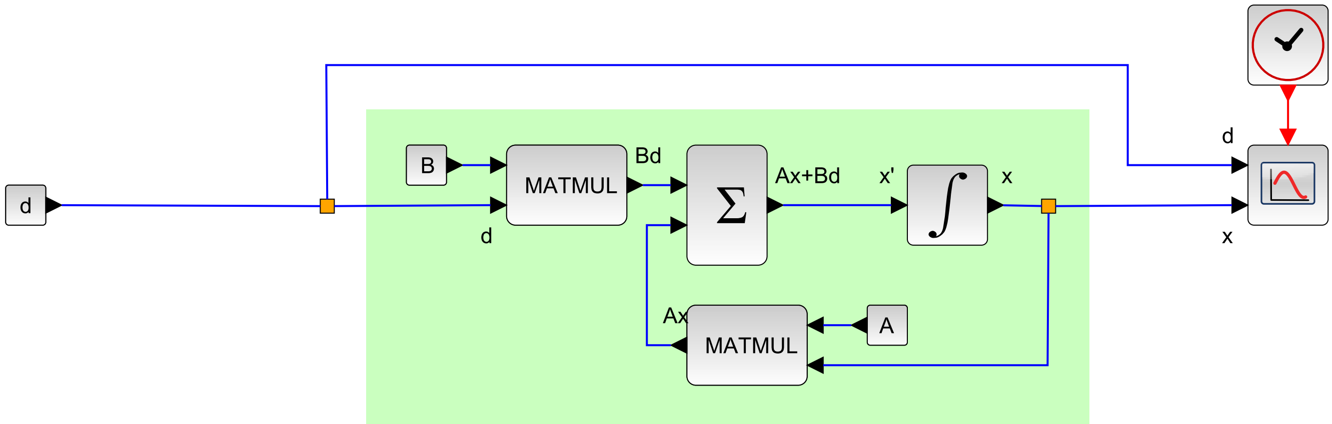
[4]:
```
# Matriz de controlabilidade
R = np.concatenate([B, A @ B, A @ A @ B], axis=1).T
print(np.linalg.matrix_rank(R) == A.shape[1])
```

```
True
```

Sistema completamente controlável, pois o posto da matriz de controlabilidade é igual ao número de estados.

3 Modele o sistema dinâmico no Simulink (MATLAB), Xcos (Scilab), ou similares

Diagrama do sistema (em verde) e sistema de simulação implementado no software xcos (scilab). Em todos os modelos desse documento, os valores do projeto foram calculados utilizando o código desse documento e copiados para o modelo no software xcos conforme necessário.

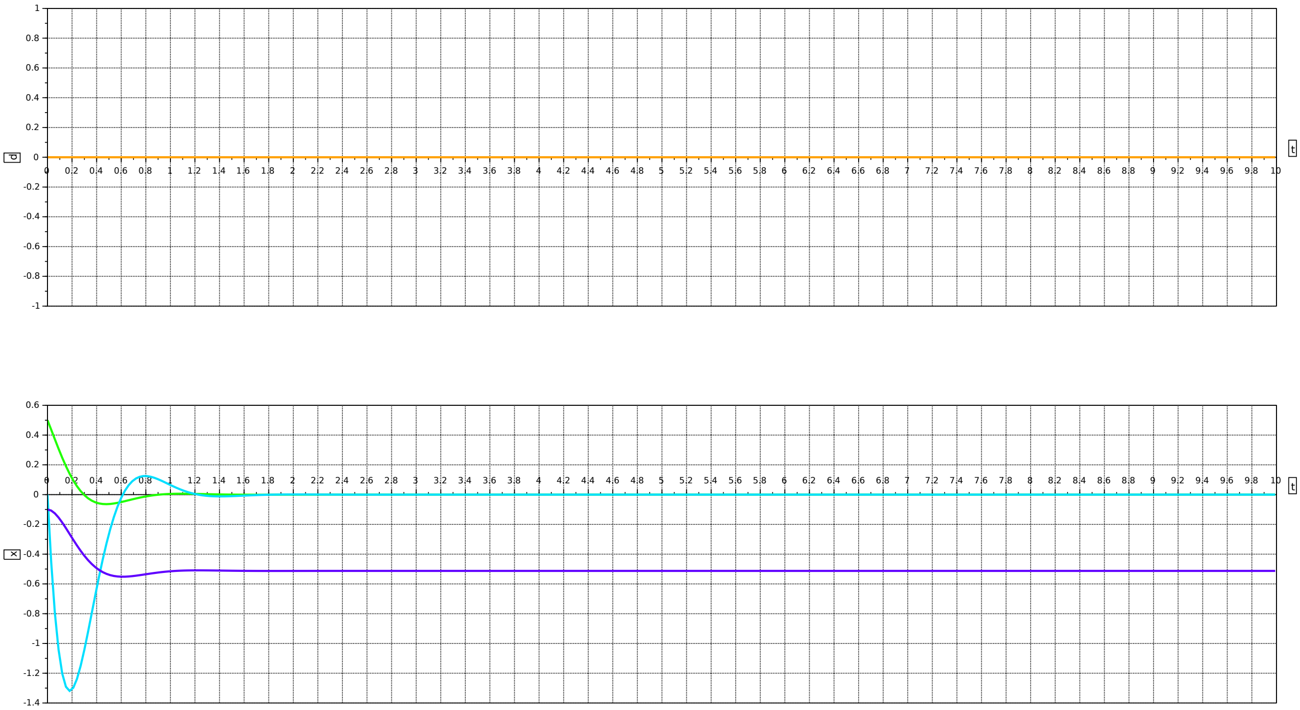


4 Simule o sistema nos seguintes casos

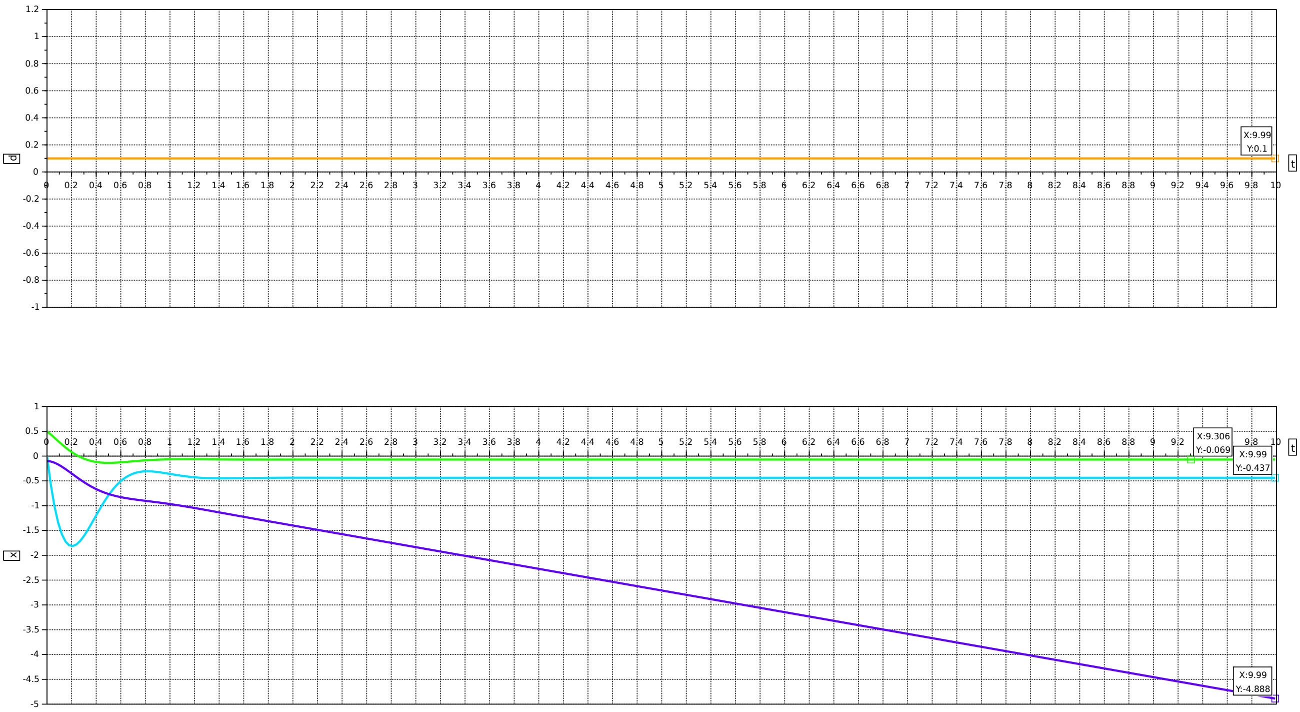
4.1 Entrada Nula

A imagem abaixo apresenta a simulação da resposta no tempo do sistema para entrada nula.

Nesse e em todos os próximos gráficos de resposta no tempo nesse documento, a linha laranja no gráfico superior representa a entrada na planta δ (chamada de d no gráfico por limitações do software), enquanto as linhas verde, azul e ciano, respectivamente, representam os estados α, q, e θ no gráfico de baixo. O eixo horizontal em todos os gráficos vai de 0 a 10 segundos, enquanto o eixo vertical foi ajustado automaticamente para destacar os detalhes do gráfico. Comparações de magnitude dever ser feitas com observação cuidadosa da escala de cada gráfico



4.2 Entrada degrau de 0.1 rad



5 Projete um regulador via alocação de polos (ou seja, calcule o vetor K) que forneça um sistema com as seguintes características de malha fechada

- Sobressinal de 20%
- Tempo de pico de 1 segundo

[5]:
```python
# Escolha de polos
tp = 1
PO = 0.20

wd = np.pi * tp
sigma = -abs(wd * np.log(0.2) / np.pi)

print(f"wd: {wd}")
print(f"sigma: {sigma}")
```

```
wd: 3.141592653589793
sigma: -1.6094379124341
```

[6]:
```python
# Ganho do feedback completo
K = control.place(A, B, (sigma + wd * 1j, sigma - wd * 1j, 10 * sigma))
print("K:")
print(K)
```

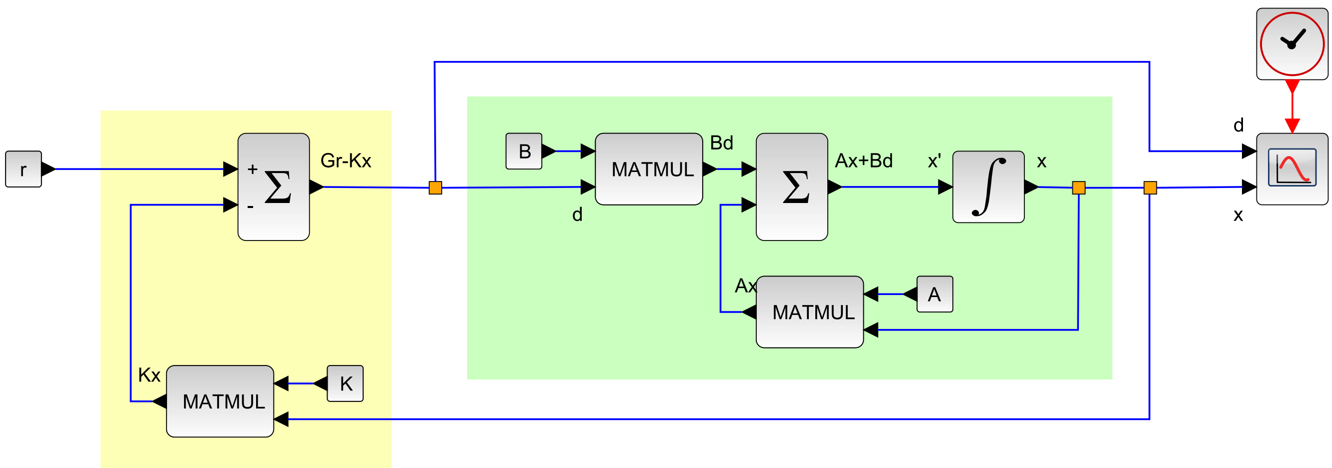
```
K:
[[ 2.3849 -0.2642 -1.1182]]
```

[7]:
```python
# Autovalores resultantes (conferir alocação)
eigvals = np.linalg.eigvals(A - B @ K)
print("polos:")
print(np.reshape(eigvals, (3, 1)))
```

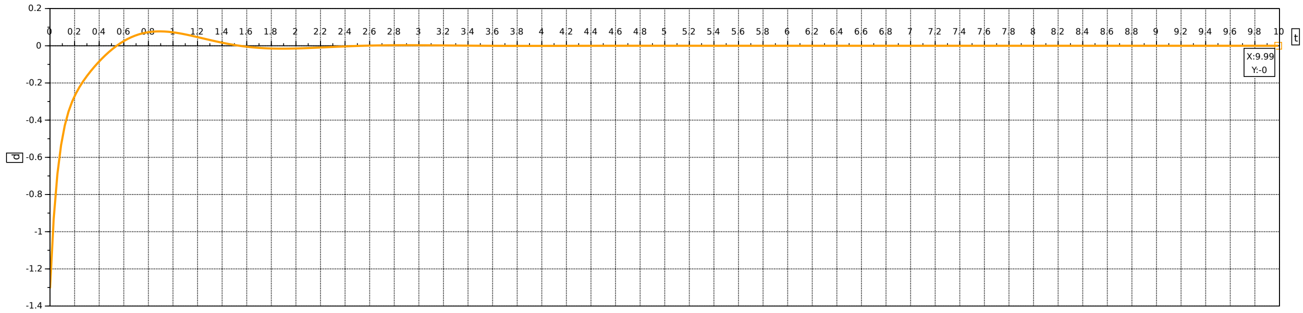
```
polos:
[[-16.0944+0.j     ]
 [ -1.6094+3.1416j]
 [ -1.6094-3.1416j]]
```

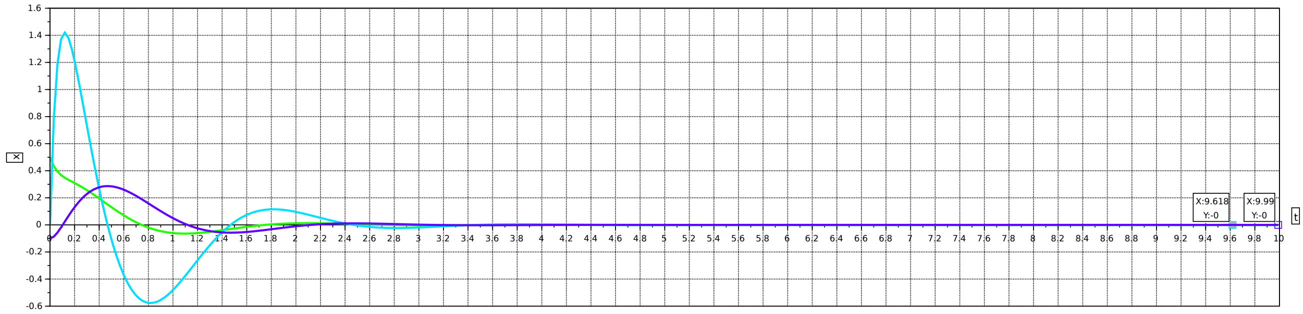
6 Implemente a alocação de polos no Simulink (MATLAB), Xcos (Scilab), ou similares. Assuma inicialmente que o controlador tem acesso ao vetor de estados completo. Inclua imagem(ns) do controlador construído.

Diagrama do controlador (em amarelo) conectado à planta (em verde)

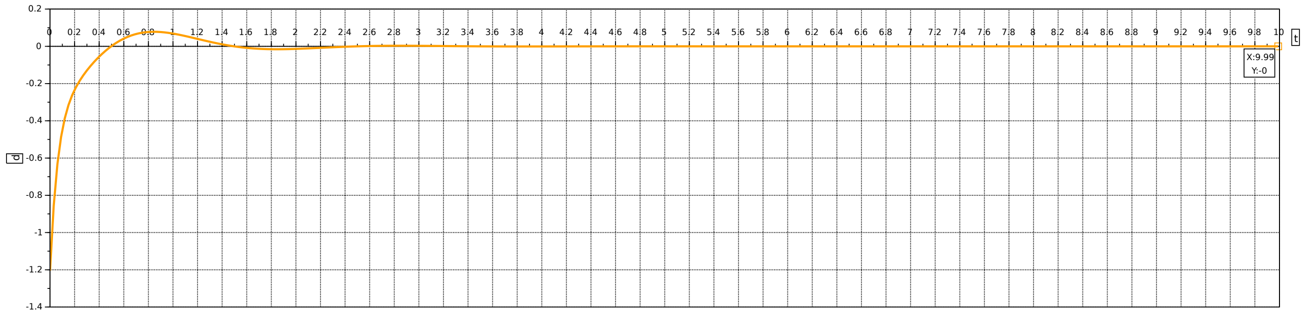


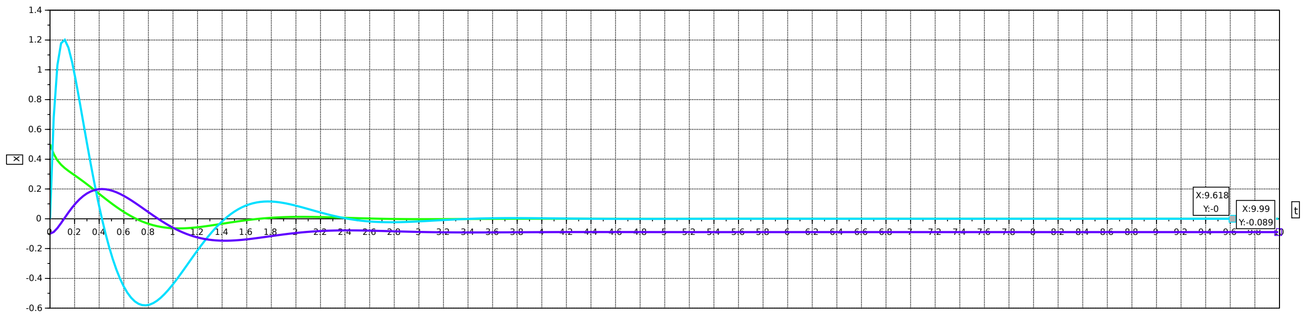
6.1 Simule o sistema com entrada nula





6.2 Simule o sistema com entrada degrau de 0.1 rad

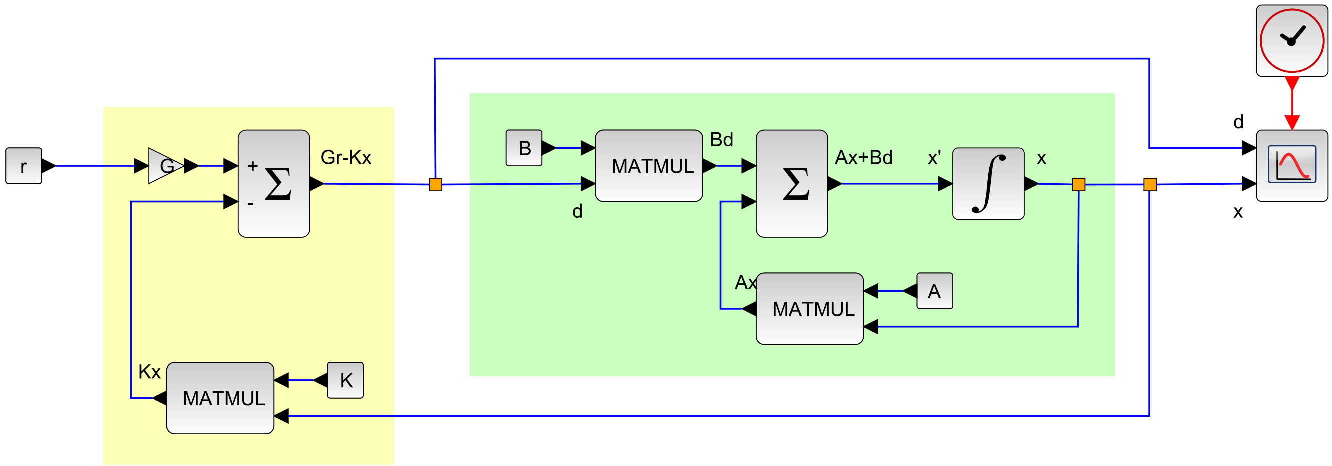




O valor de sobressinal e o tempo de pico desejados não podem ser observados nos gráficos, pois as condições iniciais dominam o sistema durante o início da resposta transiente (os valores iniciais são muito maiores que a entrada). O sistema também manteve a estabilidade de α e q, mas agora com θ estável, (anteriormente, marginalmente estável). Uma entrada degrau agora desloca θ para um degrau, e q e α para 0;

7 Altere controlador para que se transforme em um rastreador com erro ao degrau nulo em regime permanente. O sinal de entrada define o ângulo de arfagem *theta* desejado.

O modelo de rastreador é uma modificação simples do modelo anterior, adicionando apenas um ganho proporcional G à entrada de referência r. O diagrama completo pode ser visto na imagem abaixo, enquanto o cálculo do ganho é realizado em código python.

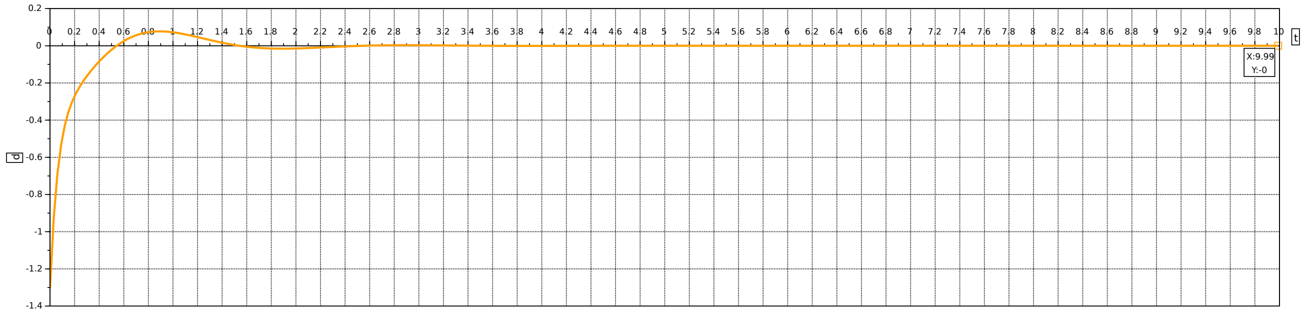


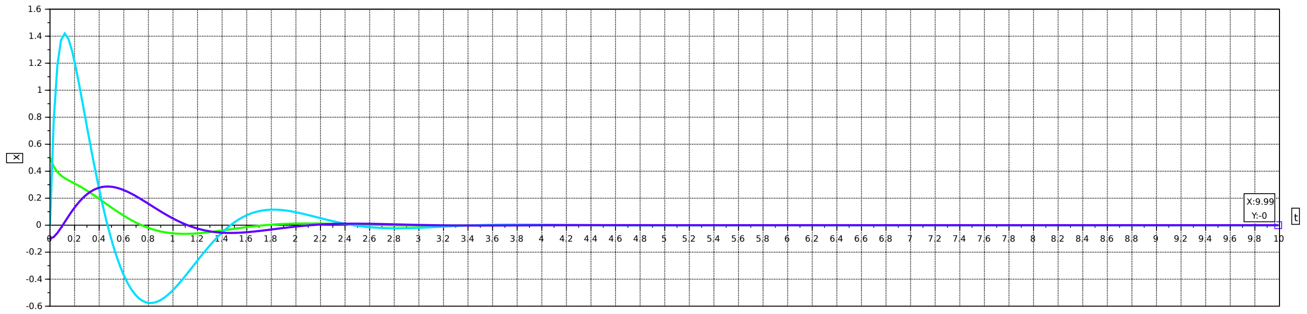
[9]:
```python
# Função para calcular o ganho proporcional
    def gain_scale(k, a, b, c):
    """
        Calcula o ganho proporcional da referência para um rastreador com feedback completo

        Modelo:
        dx = Ax + Bu
        y  = Cx
        u = Gr - Kx

        Método:
        https://www.control.utoronto.ca/people/profs/kwong/ece410/2008/notes/chap5.pdf
        G = [K I] * [[A B], [C 0]]^(-1) * [0 I]'
        """
    o = np.zeros_like(c).T
    i = np.eye(c.shape[0])

    m1 = np.block([k, i])
    m2 = np.linalg.inv(np.block([[a, b], [c, 0 * i]]))
    m3 = np.block([[o], [i]])

    g = m1 @ m2 @ m3
    return g
```

[10]:
```python
# Encontra o ganho proporcional
C = np.array([[0, 0, 1]])   # Rastreia theta
G = gain_scale(K, A, B, np.array([[0, 0, 1]]))
print("G:")
print(G)
```

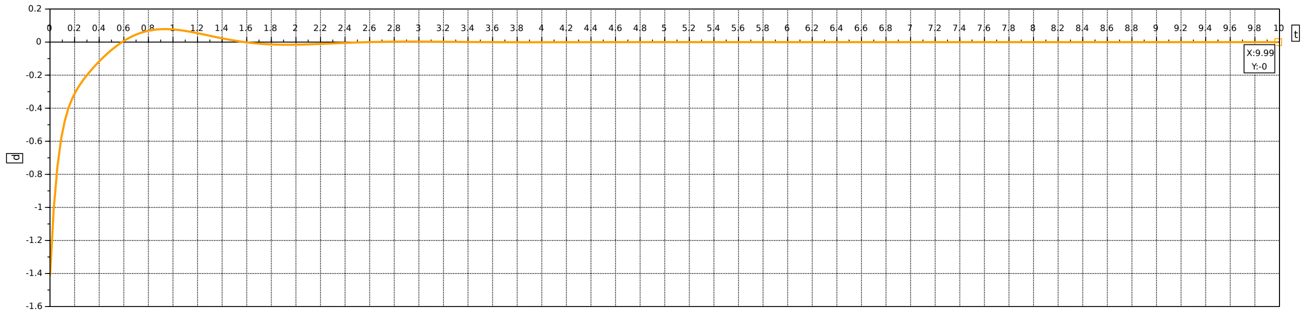
```
G:
[[-1.1182]]
```

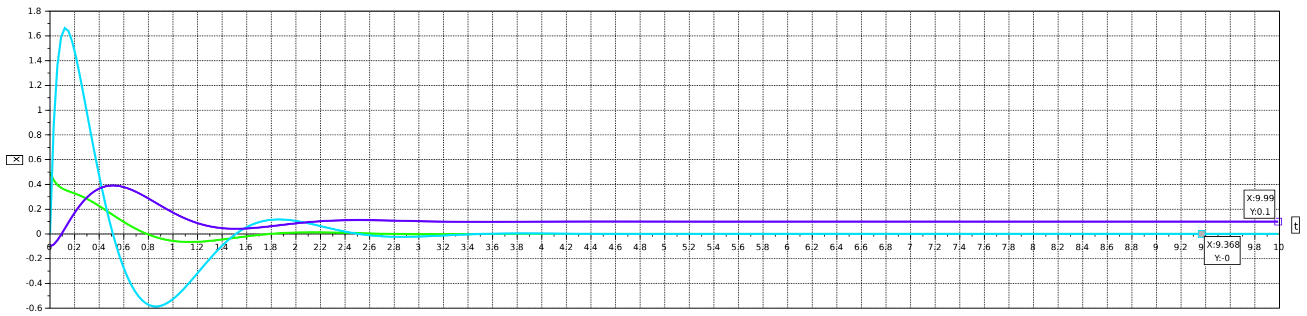
7.1 Simule o sistema com entrada nula





7.2 Simule o sistema com degrau de 0.1 rad





Conforme desejado, o estado θ agora rastreia a entrada r com erro nulo ao degrau

8 Projetar e simular dois reguladores diferentes projetados via LQR, listados abaixo.

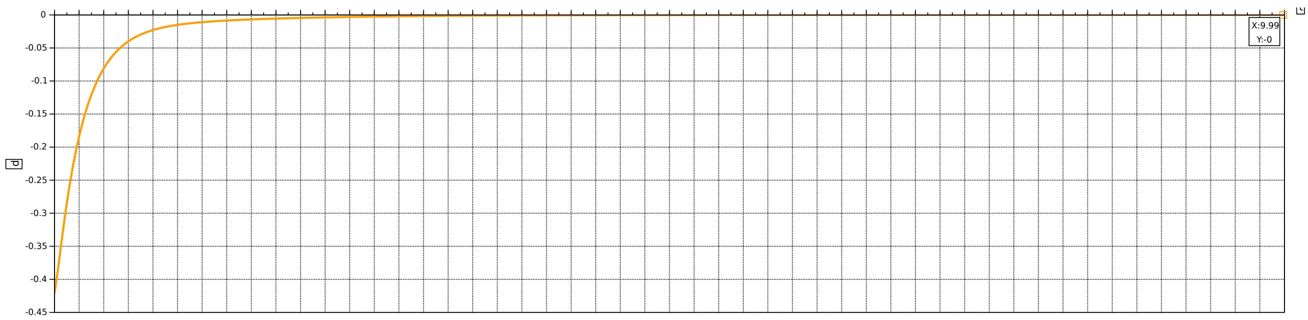
O modelo do controlador LQR é o mesmo do controlador por alocação de polos desenvolvido acima mudando apenas o método de escolha do ganho/polos.

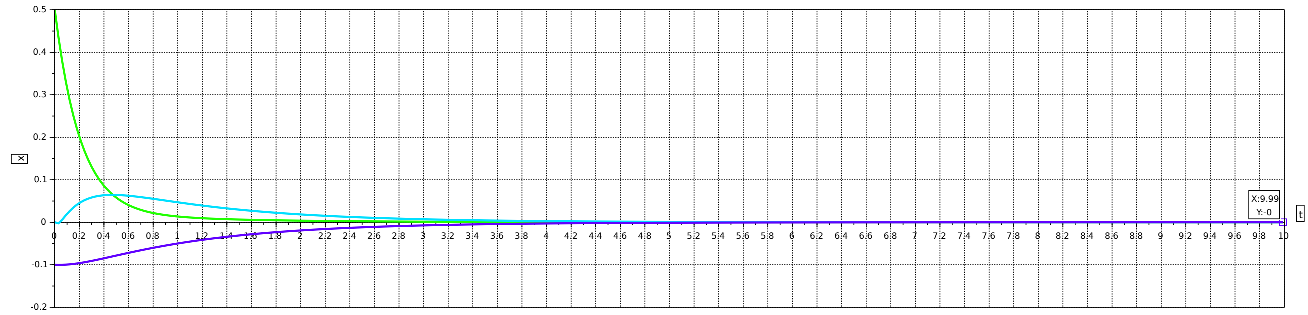
8.1 Estados e sinais de controle com pesos iguais

[11]:
```
# LQR com pesos iguais
Q1 = np.eye(3)
R1 = np.array([[1]])
K1, _, eig1 = control.lqr(A, B, Q1, R1)
# Calculado também o ganho proporcional, não necessário para a entrada nula
G1 = gain_scale(K1, A, B, C)

print("Q1:")
print(Q1)
print()

print("R1:")
print(R1)
print()

print("K1:")
print(K1)
print()

print("G1:")
print(G1)
print()

print("polos:")
print(np.reshape(eig1, (3, 1)))
```

```
Q1:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

R1:
[[1]]

K1:
[[ 0.6423 -0.9274 -1.    ]]

G1:
[[-1.]]

polos:
[[-38.4769+0.j]
 [ -4.8481+0.j]
 [ -0.9614+0.j]]
```

A resposta no tempo do sistema, mostrada abaixo, mostra que há uma resposta superamortecida. O pico de uso do atuador é de aproximadamente -0.46 rad.





8.2 Um ajuste que poupa (reduz) o uso do atuador (profundor)

[12]:
```python
# LQR com pesos variáveis, focando em baixo uso de atuador
# Optado por diminuir também a importância dos estados não rastreados
Q2 = np.array([[0.1, 0, 0], [0, 0.1, 0], [0, 0, 1]])
R2 = np.array([[10]])
K2, _, eig2 = control.lqr(A, B, Q2, R2)
G2 = gain_scale(K2, A, B, C)

print("Q2:")
print(Q2)
print()

print("R2:")
print(R2)
print()

print("K2:")
print(K2)
print()

print("G2:")
print(G2)
print()

print("polos:")
print(np.reshape(eig2, (3, 1)))
```

```
Q2:
[[0.1 0.  0. ]
 [0.  0.1 0. ]
 [0.  0.  1. ]]

R2:
[[10]]

K2:
[[ 0.2573 -0.0656 -0.3162]]
```

```
G2:
[[-0.3162]]

polos:
[[-4.5892+5.066j]
 [-4.5892-5.066j]
 [-1.2137+0.j   ]]
```

Conforme desejado, o segundo controlador apresenta menor uso do atuador (pico sai de -0.46 a -0.16). Entretanto, esse baixo uso do atuador tem como consequência adversa uma piora do comportamento transiente do sistema. θ e q inicialmente se distanciam do valor estável, devido a forças da dinâmica passiva da planta superiores à força do atuador (α alto gera momento alto).