Sexto Relatório PCO119

João Vitor Yukio Bordin Yamashita

October 8, 2022

1 Questão 1

Todos os códigos usados nesse relatório podem ser encontrados no github da matéria[1].

1. Temos o seguinte sistema no Python:

```python
import control
Gz = control.TransferFunction([1,0.4],[1, -1.2, 0.35], 2)
Gz

```

Listing 1: Sistema em Z

$$G(z) = \frac{z + 0.4}{z^2 - 1.2z + 0.35} dt = 2$$

(a) Temos o seguinte lugar das raízes:

```python
control.rlocus(Gz);

```

Listing 2: Lugar das raízes

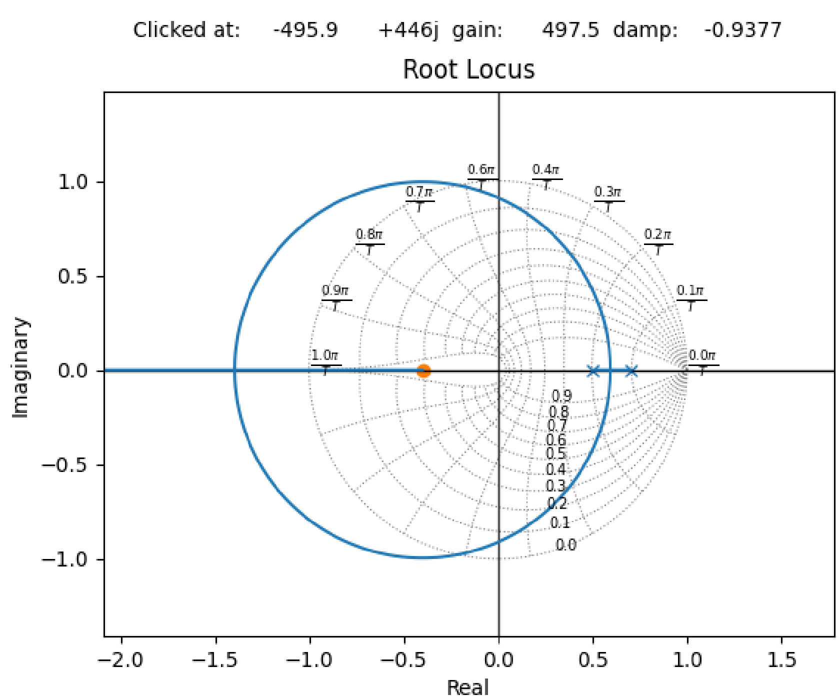

Figure 1: Lugar das raízes

[1] https://github.com/JoaoYukio/PCO119/tree/main/Atividade%206

Com isso podemos encontrar qual o valor de z onde estamos cruzando o círculo unitário, usando o zoom temos:

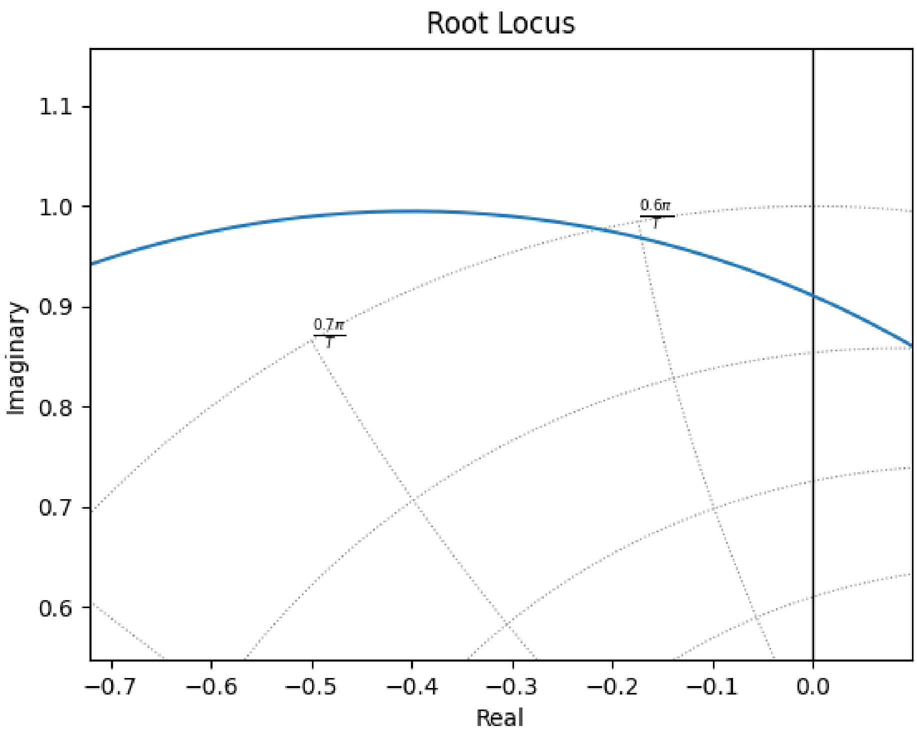

Figure 2: Cruzamento circulo

Temos um valor de aproximadamente z = -0.2124831 + 0.9771580j, com isso podemos calcular qual ganho K para atingir esse valor de z:

$$K = \frac{r_{P0.7} \times r_{P0.5}}{r_{z0.4}}$$

Usando o teorema de Pitágoras temos:

```
    zcorte = -0.2124831 + 0.9771580j
    rp0_7 = math.sqrt((zcorte.imag**2)+(0.7-zcorte.real)**2)
    rp0_5 = math.sqrt((zcorte.imag**2)+(0.5-zcorte.real)**2)
    rz0_4 = math.sqrt((zcorte.imag**2)+(0.4+zcorte.real)**2)
    K = (rp0_5*rp0_7)/rz0_4
    print(K)

    T1, yout1 = control.step_response(control.feedback((K)*Gz))
    plt.plot(T1,yout1)

    Output:
        1.6249662652993846

```

Listing 3: Calculo de K

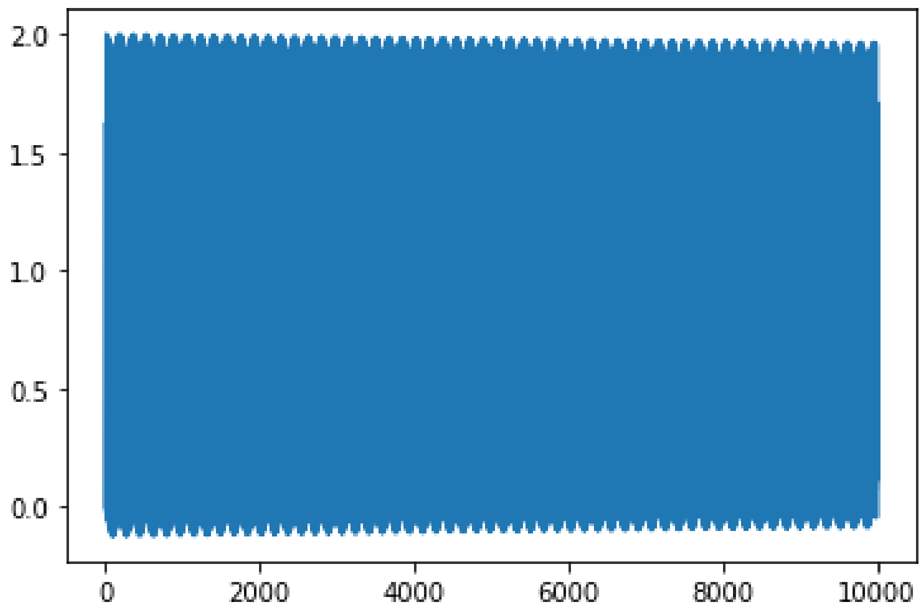

Figure 3: Ganho máximo

Podemos ver que o sistema se comporta aproximadamente como um sistema marginalmente estável. Ao aumentarmos ligeiramente o ganho K´= k + 0.1, podemos ver que o sistema se torna instável:

```
    T1, yout1 = control.step_response(control.feedback((K+0.1)*Gz), T = np.
    arange(0,1000,2))
    plt.plot(T1,yout1)

```

Listing 4: Sistema instavel

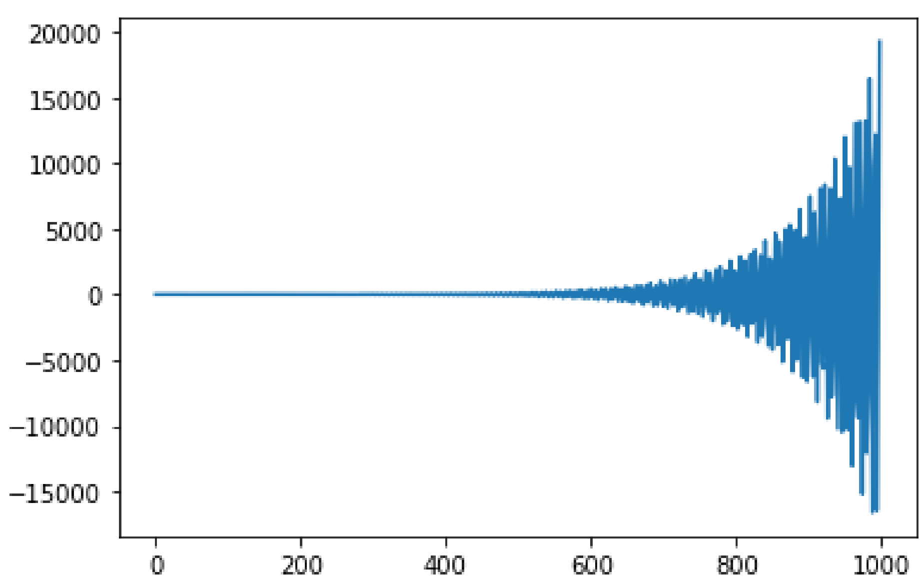

Figure 4: Sistema instavel com ganho k = k + 0.1

(b) Para verificarmos se z = 0.36 +/- j0.64 pertence ao lugar das raízes fazemos:

$$\sum_{i=1}^{n}\theta_{p_i} - \sum_{j=1}^{m}\theta_{z_j} = \pi[rad]$$

Temos então:

$$\theta_{p_{0.7}} = \pi - \arctan(\frac{0.64}{0.34}) = 2.059130277851302[Rad]$$

$$\theta_{p_{0.5}} = \pi - \arctan(\frac{0.64}{0.5-0.36}) = 1.7861540264926346[Rad]$$

$$\theta_{z_{0.4}} = \arctan(\frac{0.64}{0.4+0.36}) = 0.6998928697192437[Rad]$$

Temos então que,

$$\sum_{i=1}^{n}\theta_{p_i} - \sum_{j=1}^{m}\theta_{z_j} = 2.059130277851302 + 1.7861540264926346 - 0.6998928697192437 =$$

$$3.1453914346246927 \approx \pi[rad]$$

Em Python temos:

```
tetap1 = math.pi - math.atan((0.64/0.34))
tetap2 = math.pi - math.atan((0.64/(0.5-0.36)))
tetaz1 = math.atan((0.64/(0.4+0.36)))
tetap1 + tetap2 -tetaz1

Output:
    3.1453914346246927

```

Listing 5: z no LR

Portanto, z = 0.36 +/- j0.64 pertence ao lugar das raízes.

(c) Temos

```
control.rlocus(Gz);

```

Listing 6: Lugar das raízes

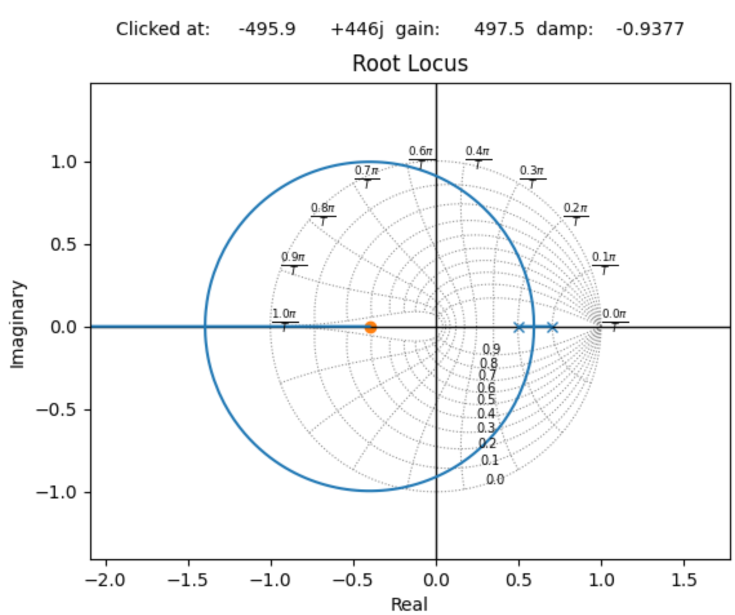

Figure 5: Lugar das raízes

(d) Como o ganho é menor do que o ganho máximo a resposta esperada é um sistema estável e superamortecido, considerando que para K = 1 os polos são apenas valores reais. Como podemos ver abaixo:

```
    yout2, T2 = control.step_response(Gz)
    plt.plot(yout2, T2)

```

Listing 7: Resposta ao degrau

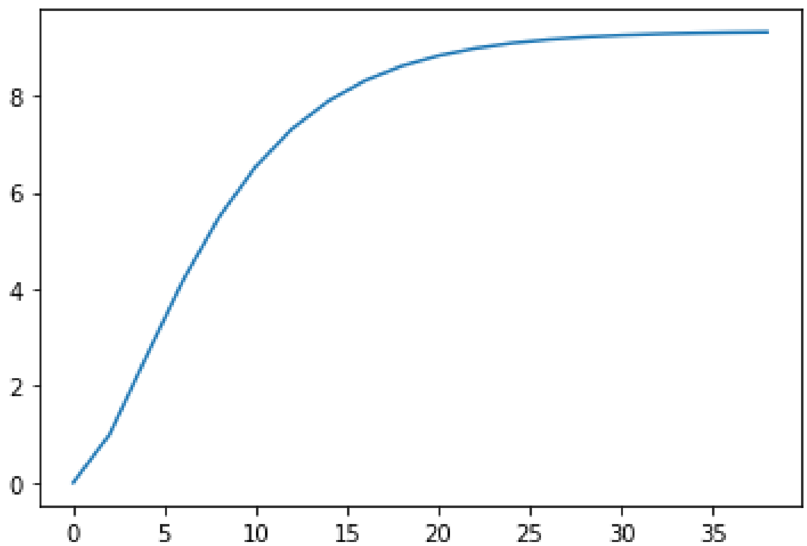

Figure 6: Resposta ao degrau

2. (a) Temos a seguinte discretização do sistema:

```
            Gs = control.TransferFunction([1], [2,1])
            Ts = 0.1
            Gz = control.sample_system(Gs, Ts, 'zoh')
            Gz

```

Listing 8: Discretizacao

$$G(z) = \frac{0.04877}{z - 0.9512} dt = 0.1$$

(b) Para o sistema em W temos:

```
            import harold

            Gs1 = harold.Transfer([1], [2,1])
            Gz1 = harold.discretize(Gs1, Ts, method = 'zoh')
            Gw1 = harold.undiscretize(Gz1, method = 'tustin')
            Gw = control.TransferFunction(Gw1.num[0], Gw1.den[0])
            Gw

```

Listing 9: Tranformada para w

$$G(w) = \frac{-0.02499s + 0.4999}{s + 0.4999}$$

(c) Temos o seguinte diagrama de Bode para os sistemas:

```
            control.bode_plot(Gs, label = "Modelo S");
            control.bode_plot(Gz, label = "Modelo Z");
            control.bode_plot(Gw, label = "Modelo W");
            plt.legend()
```

Listing 10: Diagrama de Bode

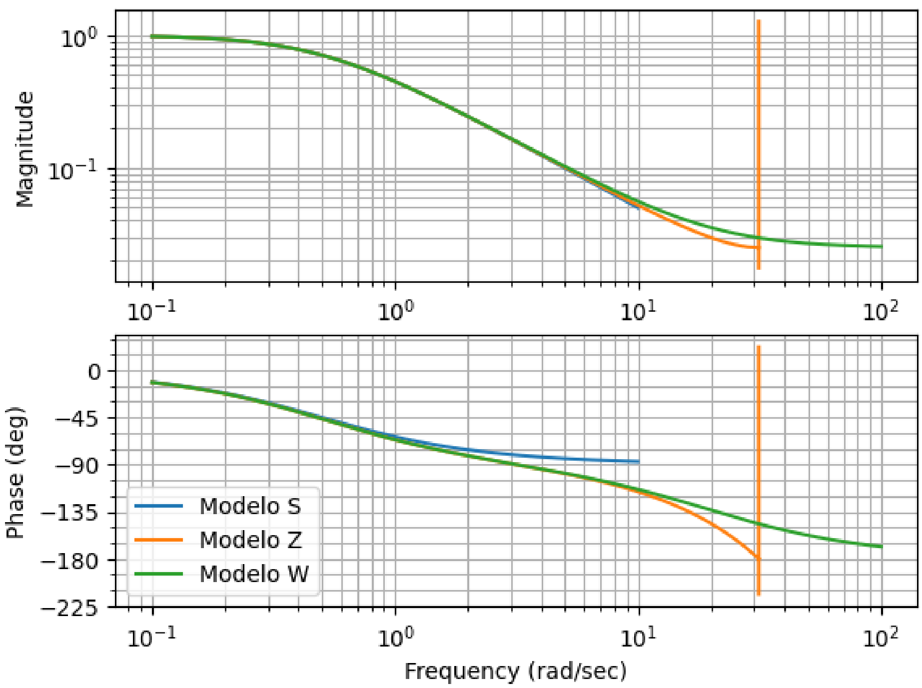

Figure 7: Diagrama de Bode dos sistemas

```
            gm, pm, wcg, wcp = control.margin(Gw)
            print('Margem de ganho: ', gm)
            print('Margem de fase: ', str(pm) + ' graus')
            print('Frequencia associada com a margem de ganho: ', str(wcg) + ' Rad/seg')
            print('Frequencia associada com a margem de fase: ', str(wcp) + ' Rad/seg')

            Output:
                Margem de ganho:  inf
                Margem de fase:  inf graus
                Frequencia associada com a margem de ganho:  nan Rad/seg
                Frequencia associada com a margem de fase:  nan Rad/seg
```

Listing 11: Margens do sistema

Podemos ver que o sistema não possui margem de ganho nem de fase.

(d) Primeiramente vamos fazer a compensação da frequência devido a reconstrução do sinal:

$$w_w = \frac{2}{T}\tan(\frac{wT}{2})$$

```
            ww = 2/Ts * math.tan((10*Ts)/2)
            ww

            Output:
                10.926049796875809
```

Listing 12: Ajuste da frequência

Para o calculo dos parâmetros do controlador vamos usar:

$$\varphi = -180 + \phi_m - \angle G(j\omega_u) \tag{1}$$

$$K_P = \frac{\cos(\varphi)}{|G(j\omega_u)|} \tag{2}$$

$$K_D\omega_u - \frac{K_I}{\omega_u} = \frac{\sin(\varphi)}{|G(j\omega_u)|} \tag{3}$$

De 1 temos:

```
magJW, phase, omega = control.bode(Gw, dB =False,omega = ww, plot
=False)
phi = -180 + 45 - (phase[0] * (180 / 3.14159265))
phi

Output:
    -18.97171766320092

```

Listing 13: Calculo phi

Com o valor de φ podemos calcular 2:

```
Kp = math.cos(phi * (math.pi)/180) / (magJW[0])
Kp

Output:
    18.158063203608588

```

Listing 14: Calculo Kp

Também podemos calcular 3:

```
Ki = - ww * (math.sin(phi * (math.pi)/180)/(magJW[0]))
Ki

Output:
    68.20366314639502

```

Listing 15: Calculo Ki

Com isso podemos modelar nosso controlador PI:

```
P = control.TransferFunction([Kp], 1)
I  = control.TransferFunction(Ki, [1, 0])
PI = P + I
PI

```

Listing 16: Controlador PI

Temos o seguinte controlador PI:

$$G_{PI}(w) = \frac{18.16s + 68.2}{s}$$

(e) Temos a seguinte resposta em frequência:

```
control.bode(control.series(Gw, PI), label = 'Sistema controlado'
);
plt.legend()

```

Listing 17: Bode em w

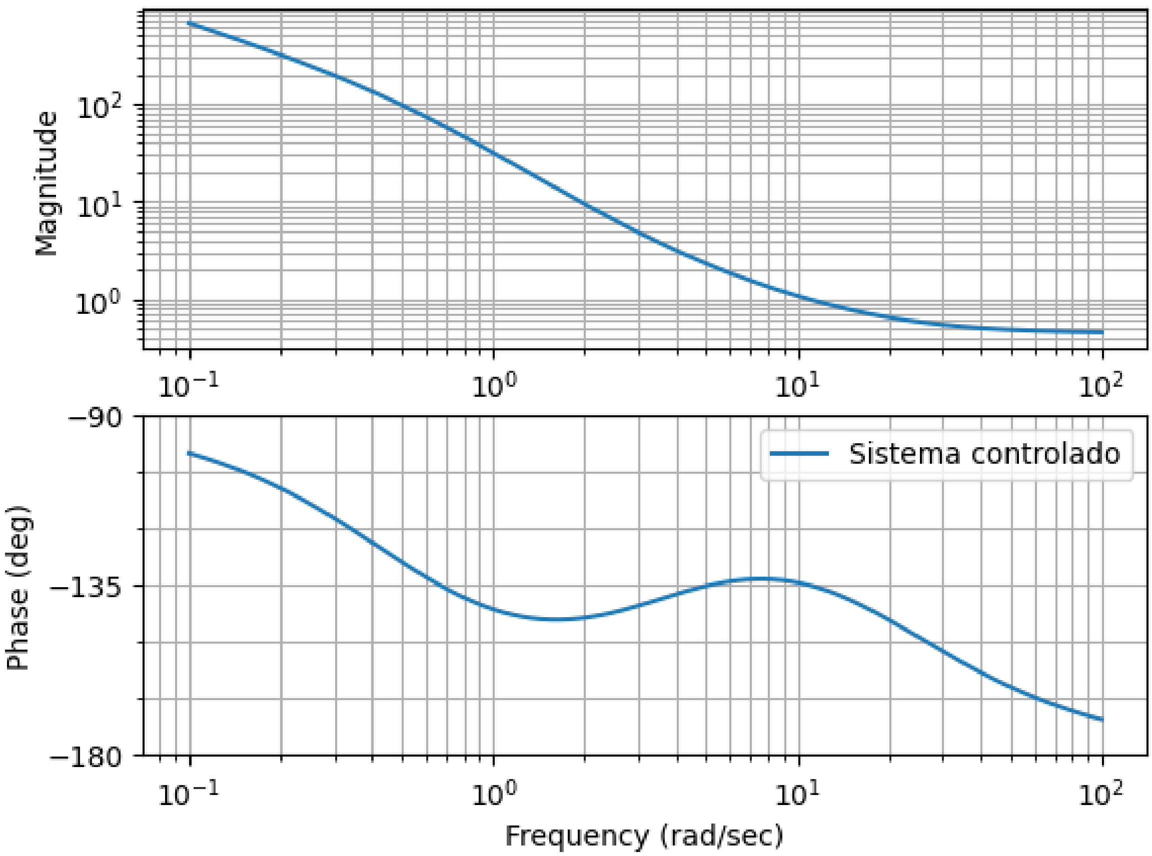

Figure 8: Diagrama de Bode do sistema controlado

Podemos confirmar que os requisitos do projeto foram atendidos fazendo:

```
        gm, pm, wcg, wcp = control.margin(control.series(Gw, PI))
        print('Margem de ganho: ', gm)
        print('Margem de fase: ', str(pm) + ' graus')
        print('Frequencia associada com a margem de ganho: ', str(wcg) + ' Rad/seg')
        print('Frequencia associada com a margem de fase: ', str(wcp) + ' Rad/seg')

        Output:
                Margem de ganho:  inf
                Margem de fase:  45.00000013258162 graus
                Frequencia associada com a margem de ganho:  nan Rad/seg
                Frequencia associada com a margem de fase:  10.926049796875812 Rad/seg

```

Listing 18: Margens do sistema controlado

Podemos ver que as margens do sistema são exatamente as especificadas do projeto. Usando o zoom podemos ver exatamente os pontos de interesse:

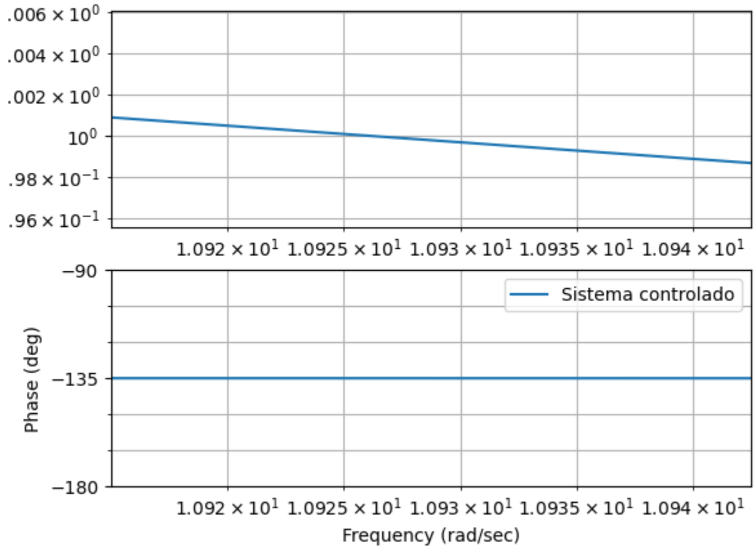

Figure 9: Frequência de cruzamento

Podemos perceber em 9 que o cruzamento de 1dB ocorre em 10.926 [Rad/s] e que temos uma fase de -135°, o que implica uma margem de fase de 45°, exatamente o que foi especificado no projeto (considerando o ajuste da frequência devido a conversão de z para w). Podemos ver que o controlador se trata de um avanço de fase.

(f) A resposta ao degrau do sistema passa a ser:

```python
y, T = control.step_response(control.feedback(control.series(Gw, PI)))
plt.plot(y,T)
plt.grid()
plt.title("Resposta do sistema controlado")

```

Listing 19: Resposta do sistema controlado

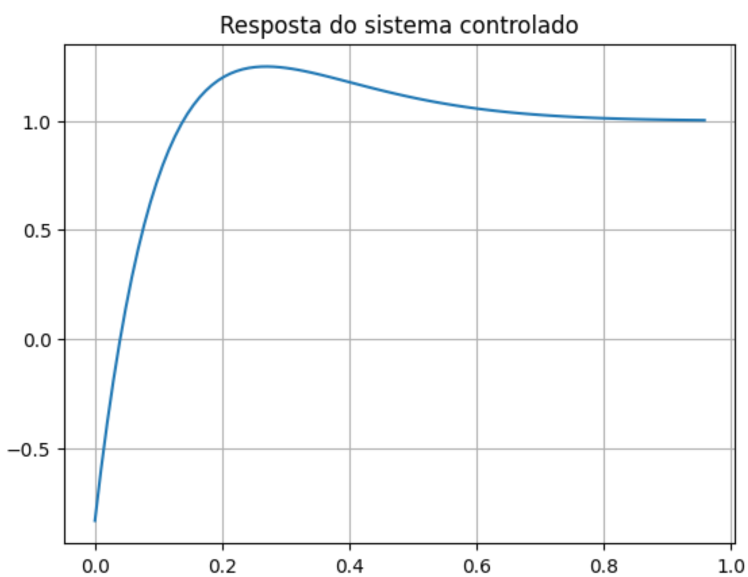

Figure 10: Resposta ao degrau do sistema controlado

Podemos comparar o efeito do controlador fazendo:

```
            Temp = np.arange(0, 15, 0.001)

            yout2, T2 = control.step_response(control.feedback(Gs), T = Temp)
            y, T = control.step_response(control.feedback(control.series(Gw,
    PI)), T = Temp)

            plt.plot(y,T, label = "Resposta do sistema controlado")
            plt.plot(yout2, T2, label = 'Sistema sem controle')
            plt.legend()
            plt.grid()

```

Listing 20: Comparação das respostas

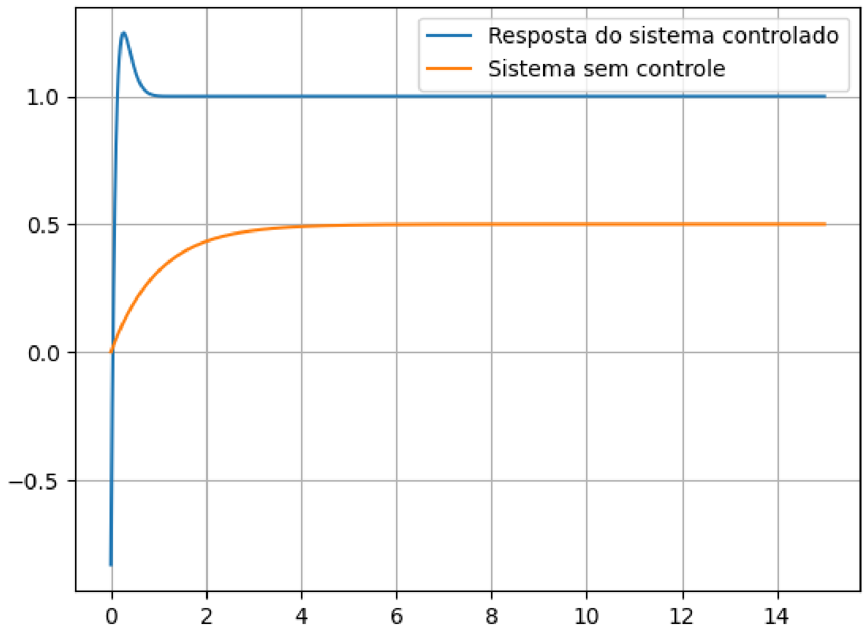

Figure 11: Comparação dos sistemas

De 11 podemos perceber que não só o controlador zerou o erro (devido ao integrador) ele também reduziu o tempo de acomodação com um custo de introduzir um *overshoot* na resposta do sistema.

(g) Para discretização do controlador temos:

```
            PIz = control.sample_system(PI, Ts)
            PIz

```

Listing 21: Controlador Z

Temos o seguinte controlador em Z:

$$G_{PI}(Z) = \frac{18.16z - 11.34}{z - 1} dt = 0.1$$

Temos a seguinte resposta para o sistema discretizado:

```
            y2, T = control.step_response(control.feedback(control.series(Gz,
     PIz)))
            plt.plot(y2, T)
            plt.grid()
            plt.title("Resposta do sistema controlado e discretizado")

```

Listing 22: Resposta em Z

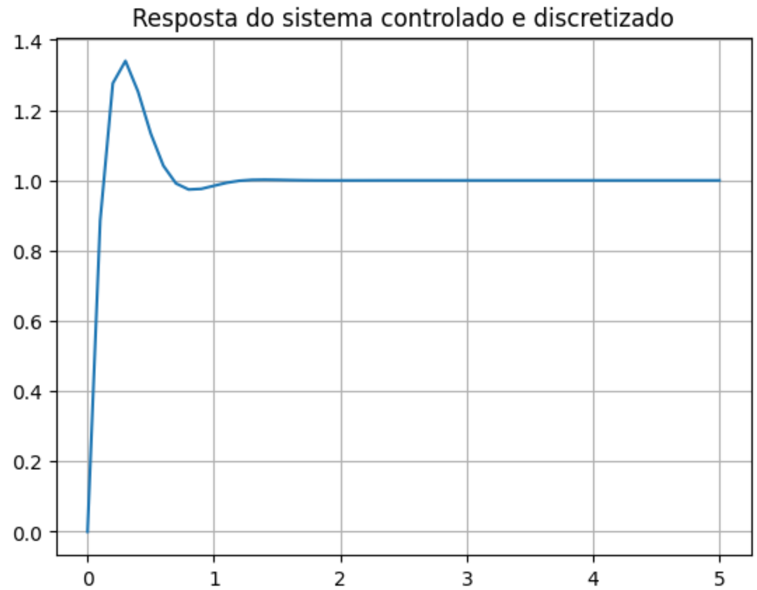

Figure 12: Resposta do sistema discretizado

Por fim podemos comparar a resposta do sistema controlado em S e em Z:

```
        y2, T = control.step_response(control.feedback(control.series(Gz, PIz)))
        y, T2 = control.step_response(control.feedback(control.series(Gw, PI)), T = np.arange(0, 5, 0.001))

        plt.plot(y2, T, label = "Resposta do sistema controlado e discretizado")
        plt.plot(y,T2, label = "Resposta do sistema controlado em s")
        plt.grid()
        plt.legend()

```

Listing 23: Comparação de s e z

Figure 13: Comparação da resposta do sistema em s e em z