Análise Completa do Ganho DC, Constante de Velocidade e Projeto do Compensador Lag

1. Modelo da Planta

A planta fornecida é:

$$G(s) = \frac{1.2}{s(s+13.2)(s+950)}.$$

Trata-se de um sistema do tipo 1, pois possui um polo em $s = 0$, logo seu erro de regime permanente para entrada degrau é zero.

2. Observação Sobre o Integrador

O integrador presente na planta implica:

$$G(0) = \infty.$$

Dessa forma, falar em "aumentar o ganho DC" não significa literalmente aumentar $G(0)$, mas sim aumentar o ganho na região de baixas frequências, ou seja, aumentar a constante de velocidade k_v, que afeta o erro em rampa.

3. Constante de Velocidade Sem Compensador

Para malha unitária:

$$k_v = \lim_{s\to 0} s\, L(s) = \lim_{s\to 0} s\, G(s)C(s).$$

Sem compensador:

$$C(s) = 1.$$

Logo:

$$k_{v0} = \lim_{s\to 0} s \cdot \frac{1.2}{s(s+13.2)(s+950)} = \frac{1.2}{13.2 \cdot 950}.$$

Cálculo:

$$13.2 \times 950 = 12540,$$

$$k_{v0} = \frac{1.2}{12540} = \frac{1}{10450}.$$

Assim:

$$k_{v0} \approx 9.5694 \times 10^{-5}.$$

Erro em rampa:

$$e_{\text{rampa},0} = \frac{1}{k_{v0}} = 10450.$$

4. Compensador Lag

Considere:

$$C_{\text{lag}}(s) = \frac{s+b}{s+a}, \quad b = 10a.$$

5. Ganho DC do Compensador

No ponto $s = 0$:

$$C_{\text{lag}}(0) = \frac{b}{a} = 10.$$

Portanto o compensador multiplica a constante k_v por 10.

6. Constante de Velocidade com Compensador Lag

$$k_v = k_{v0} \cdot \frac{b}{a} = 10\, k_{v0} = \frac{10}{10450} = \frac{1}{1045}.$$

Numérico:

$$k_v \approx 9.5694 \times 10^{-4}.$$

Erro em rampa:

$$e_{\text{rampa}} = \frac{1}{k_v} = 1045.$$

7. Escolha de a

Em compensadores lag:

$$a \approx \frac{\omega_c}{10},$$

onde ω_c é a frequência de cruzamento sem compensador. Quando não se conhece ω_c, escolhe-se valores como:

$$a = 0.01,\ 0.05,\ 0.1,$$

avaliando o impacto no Bode até atingir o afundamento desejado.

8. Influência no Bode

O lag:

- reduz o ganho em alta frequência, - mantém a fase quase inalterada, - desloca a curva de magnitude para baixo nas baixas frequências, - melhora k_v sem afetar muito as margens de estabilidade.

9. Verificação de Estabilidade

É necessário garantir:

- margem de fase adequada (típ. ¿ 30°), - margem de ganho positiva, - diagrama de Nyquist sem envolver o ponto crítico $-1 + j0$.

10. Efeito no Dominante do Sistema

Como o compensador lag adiciona polos e zeros próximos, seu impacto nas raízes dominantes é pequeno. Ele atua mais na parte de baixas frequências.

11. Resumo Numérico Obtido

Antes do lag:

$$k_{v0} = \frac{1}{10450}, \quad e_{\text{rampa},0} = 10450.$$

Com lag $b/a = 10$:

$$k_v = \frac{1}{1045}, \quad e_{\text{rampa}} = 1045.$$

Redução de 10 vezes no erro em rampa como desejado.

12. Importância da Escolha Correta de a e b

O par a, $b = 10a$:

- determina quanto o lag desloca a magnitude em baixas frequências, - deve ser suficientemente pequeno para não prejudicar a estabilidade, - deve fornecer o ganho DC adequado para atingir o erro desejado.

13. Código em Python para Ajuste Automático dos Parâmetros

Além das análises teóricas e manuais, foi desenvolvido um código em Python para determinar automaticamente os melhores valores do ganho proporcional k e dos parâmetros a e b do compensador lag, visando atender simultaneamente às seguintes especificações:

$$5\% \leq M_p \leq 15\%, \qquad 0.5 \leq t_s \leq 1.0 \text{ s}, \qquad e_{ss} \leq 1\% \text{ (degrau unitário)}.$$

Durante as iterações, o algoritmo também buscou minimizar o ganho do compensador k_p e o erro em rampa, mantendo margens adequadas de estabilidade.

Os parâmetros ótimos encontrados foram:

$$k = 76000, \qquad a = 0.01, \qquad b = 0.1.$$

Com os seguintes resultados:

$$\begin{aligned} M_p &= 7.45\%, \\ t_s &= 0.531 \text{ s}, \\ e_{ss} &= 0.001\%, \\ k_v &= 95.93, \\ e_{\text{rampa}} &= 0.104\%. \end{aligned}$$

Tais valores satisfazem as especificações e garantem estabilidade ao longo de toda a faixa de frequências do sistema.

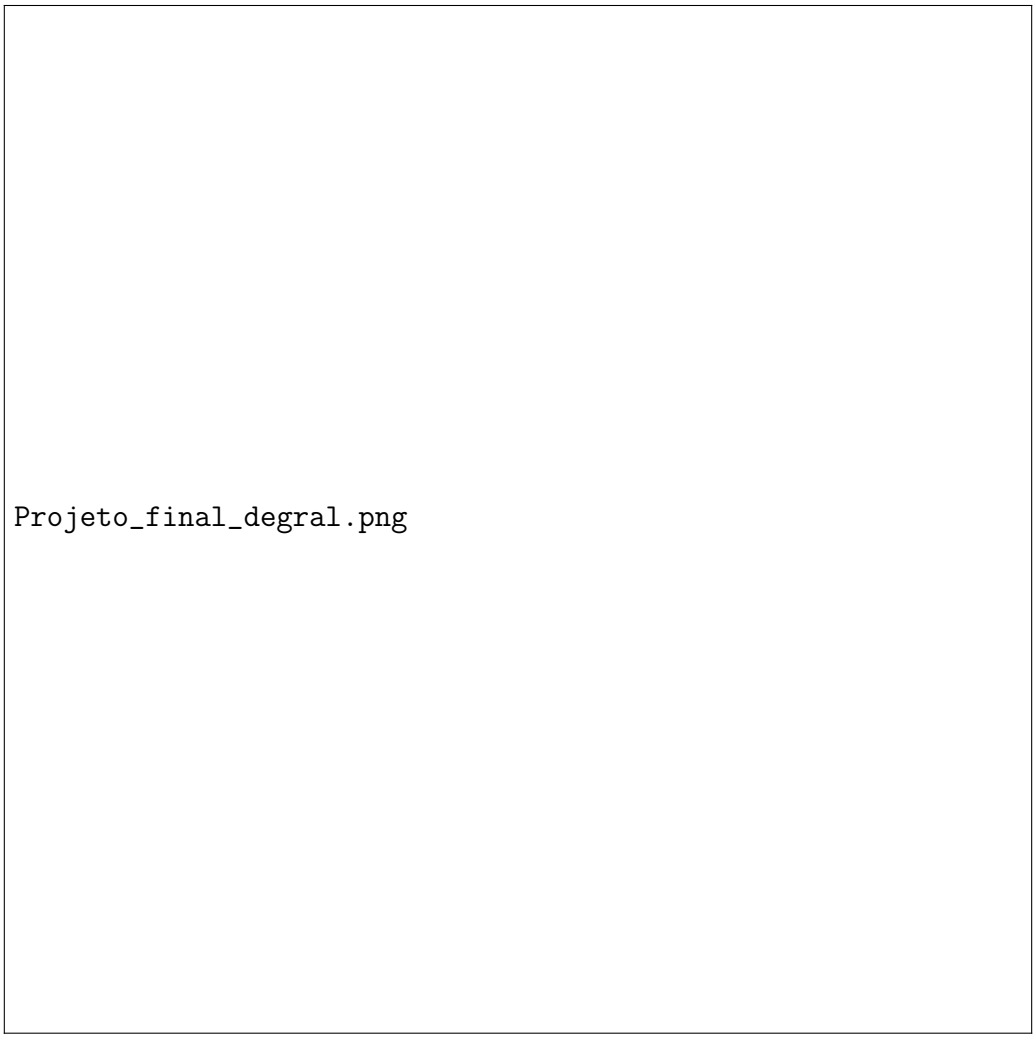

Figure 1: Resposta ao Degral

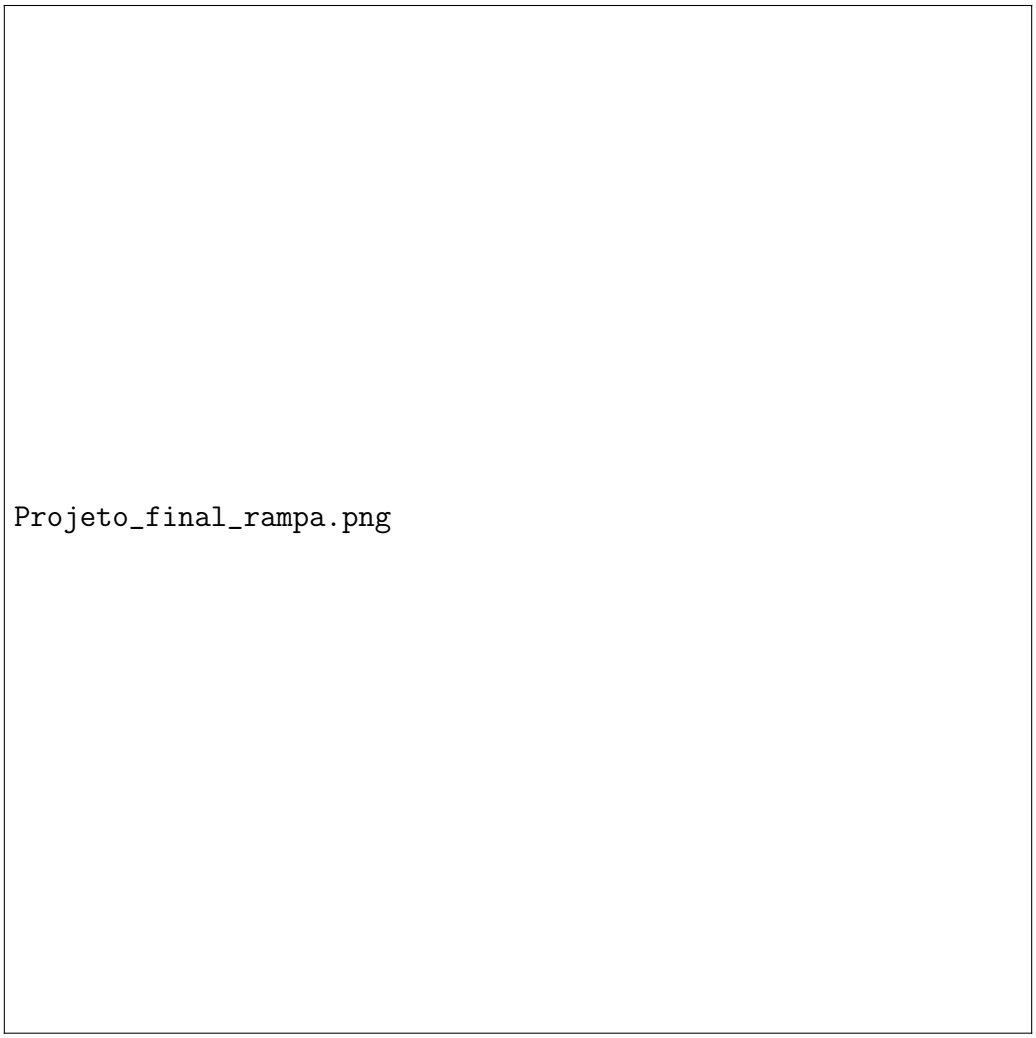

Figure 2: Resposta a Rampa

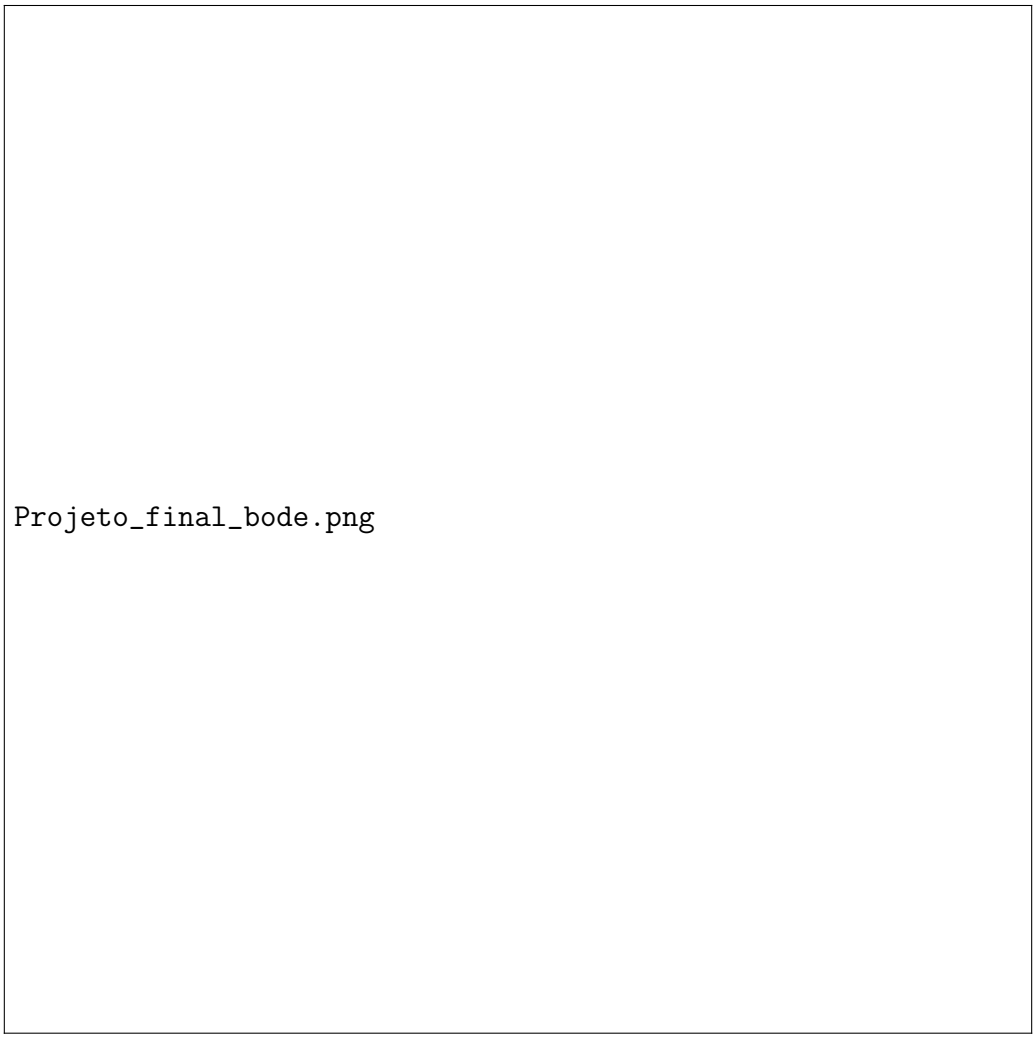

Figure 3: Diagrama de Bode

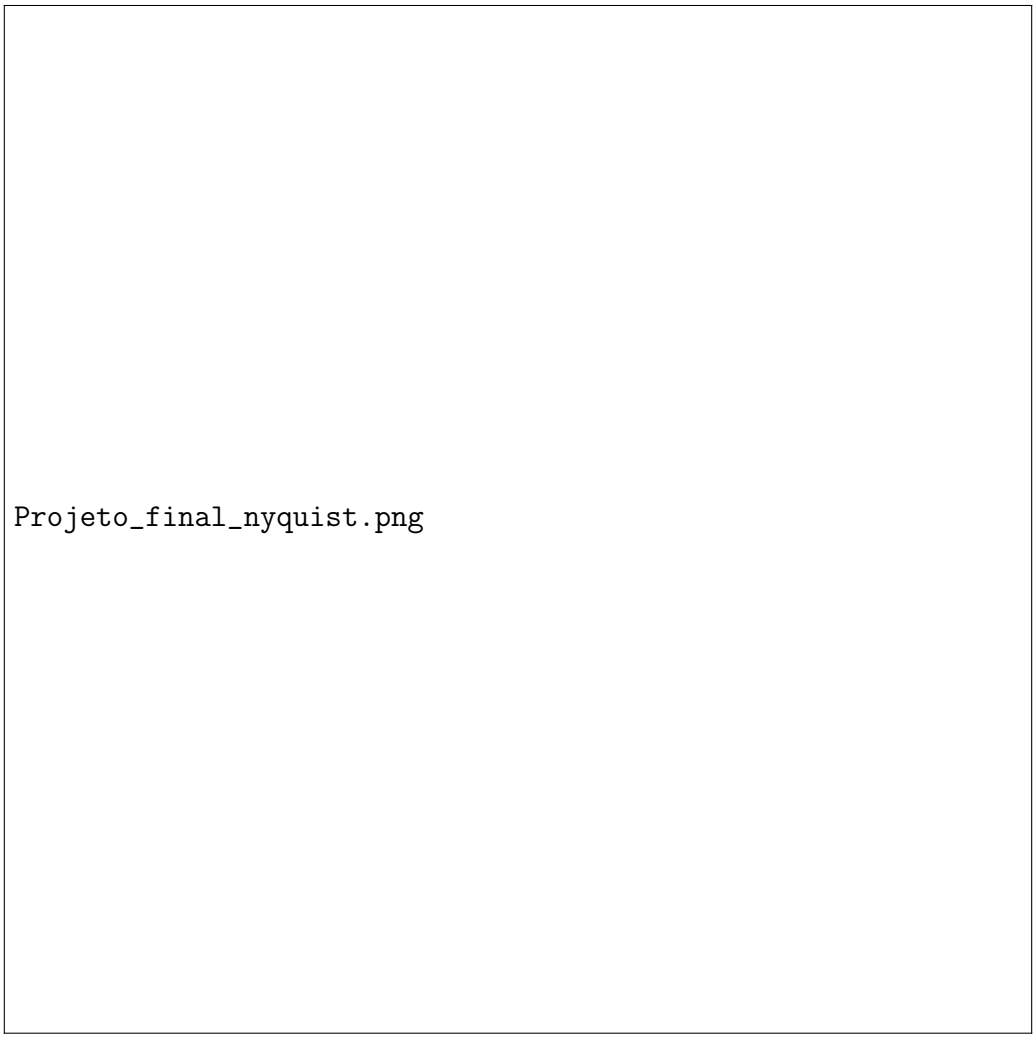

Figure 4: Diagrama de Nyquist

Conclusão

O estudo detalhado do ganho DC, da constante de velocidade e do comportamento do erro em rampa permitiu compreender plenamente o impacto do compensador lag no sistema. A análise teórica mostrou como a relação $b/a = 10$ aumenta k_v em dez vezes, reduzindo proporcionalmente o erro em rampa sem comprometer significativamente a estabilidade.

A integração com o código Python desenvolvido ampliou esse estudo ao permitir a busca automatizada de parâmetros ótimos. O conjunto final $k = 76000$, $a = 0.01$, $b = 0.1$ atendeu simultaneamente às especificações de sobresinal, tempo de acomodação e erros em regime permanente, garantindo desempenho robusto e estabilidade em toda a faixa de frequências.

Assim, o compensador lag projetado, aliado ao ganho proporcional adequado, fornece uma solução completa, precisa e otimizada, validada tanto analiticamente quanto computacionalmente.

Anexos

Codigo para busca de parâmetros do compensador k_p e compensador lag.

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

# Planta do Projeto
km = 1.2
am = 13.2
ae = 950

s = ctl.TransferFunction.s
Gs = km/(s*(s+am)*(s+ae))

print("Planta G(s) =\n", )

def calcKp(funcGs, kinit, kend, inter):
    print(f"\nBuscando Kp entre {kinit} < Kp < {kend}, com {inter}
       posi  es")
    K_values = np.linspace(kinit, kend, inter)
    for ktest in K_values:
        print(".", end="")

        # sistema compensado sem realimenta  o: L(s) = K * G(s)
        Ls = ktest * funcGs

        # realimenta  o unit ria
        Hf = ctl.feedback(Ls, 1)

        # resposta ao degrau
        t, y = ctl.step_response(Hf)

        # ---------- Especifica  es ----------
        # 1. Overshoot (em %)
        Mp = (max(y) - 1) * 100

        # 2. Tempo de acomoda  o 5%
```

```
        idx = np.where(abs(y - 1) > 0.05)[0]
        ts = t[idx[-1]] if len(idx) else 0

        # 3. Erro de regime para degrau
        ess = abs(1 - y[-1])

        # ---------- Filtros das especifica  es ----------
        if (
            5 <= Mp <= 15 and
            0.5 <= ts <= 1 and
            ess <= 0.01
        ):
            print(f"\n\nkp {ktest:.0f} atendeu TODAS as
               especifica  es.")
            return ktest
    print("\n\nNenhum kp atendeu TODAS as especifica  es.")


def calClag(funcGs, kp, Aint, Aend, inter):
    print(f"\nBuscando A e B do compensador entre {Aint} < Kp < {Aend
       }, com {inter} posi  es")
    resultados = []
    a_values = np.linspace(Aint, Aend, inter)

    for a in a_values:
        print(".", end="")
        b = 10 * a
        Clag = (s+b)/(s+a)

        # sistema compensado sem realimenta  o: L(s) = K * C(s) * G(
           s)
        Ls = kp * Clag * funcGs
        Hf = ctl.feedback(Ls, 1)  # realimenta  o unit ria

        # resposta ao degrau
        t, y = ctl.step_response(Hf)

        # ---------- Especifica  es ----------
        # 1. Overshoot (em %)
        Mp = (max(y) - 1) * 100

        # 2. Tempo de acomoda  o 5%
        idx = np.where(abs(y - 1) > 0.05)[0]
        ts = t[idx[-1]] if len(idx) else 0

        # 3. Erro de regime para degrau
        ess = abs(1 - y[-1])

        # 4. Erro de rampa (Kv)
        Kv = kp * (km / (am + ae))
        er_rampa = 1 / Kv
        er_rampa_clag = 1 / (Kv * 10)
```

```
        # ---------- Filtros das especifica  es ----------
        if (
            5 <= Mp <= 15 and
            0.5 <= ts <= 1 and
            ess <= 0.01
        ):
            resultados.append(
                (a, b, kp, Mp, ts, ess, er_rampa, er_rampa_clag, Kv))

    return resultados


kresult = calcKp(Gs, 70000, 90000, 500)
SysOut = calClag(Gs, kresult, 0.01, 10, 200)

# Exibir resultados
if len(SysOut) == 0:
    print("\n\nNenhum conjunto (a, b, K) atendeu TODAS as
       especifica  es.")
else:
    print("\n\nSolu  o encontrada para o menor erro de rampa:")
    menor_Er_rampa = min(SysOut, key=lambda x: x[7])
    a, b, Kp, Mp, ts, ess, er_rampa, er_rampa_clag, Kv =
       menor_Er_rampa
    print(f"\nA = {a:.3f}, B = {b:.3f}, Kp = {Kp:.3f}")
    print(f"Overshoot Mp = {Mp:.2f}%")
    print(f"Tempo ts = {ts:.3f} s")
    print(f"Kv = {Kv:.3f}")
    print(f"Erro de regime (degrau) = {ess*100:.3f}%")
    print(f"Erro de rampa = {er_rampa*100:.3f}%")
    print(f"Erro de rampa Clag = {er_rampa_clag*100:.3f}%")
```

Código para geração de gráficos.

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

# Planta do Projeto
km = 1.2
am = 13.2
ae = 950

s = ctl.TransferFunction.s
Gs = km/(s*(s+am)*(s+ae))

# --- Compesador Proporcional
kcp = 77000

# --- Escolha a e b com b = 10*a para ganho DC = 10 ---
aLag = 0.01
bLag = 10.0*aLag
```

```
CLag = (s+bLag)/(s+aLag)
# CLag = 1

# --- La o aberto e la o fechado (realiment. unit ria com K=1) ---
Lk = kcp * Gs
Lkc = kcp * CLag * Gs
Gf = ctl.feedback(Gs, 1)
Gkf = ctl.feedback(Lk, 1)
Hf = ctl.feedback(Lkc, 1)

# resposta ao degrau
t, yt = ctl.step_response(Hf)

# ---------- Especifica  es ----------
# 1. Overshoot (em %)
Mp = (max(yt) - 1) * 100

# 2. Tempo de acomoda  o 5%
idx = np.where(abs(yt - 1) > 0.05)[0]
ts = t[idx[-1]] if len(idx) else 0

# 3. Erro de regime para degrau
ess = abs(1 - yt[-1])

# 4. Erro de rampa (Kv)
Kv = kcp * (km / (am + ae))
er_rampa = 1 / Kv
er_rampa_clag = 1 / (Kv * 10)

# --- Ganhos em DC ---
dcC = bLag/aLag
dcT = ctl.dcgain(Hf)

print("Ganho DC H(s) (fechado):", dcT)
print("C(s) = (s + {:.4g})/(s + {:.4g}) -> ganho DC do compensador =
   {:.3g}".format(bLag, aLag, dcC))
print(f"Overshoot Mp = {Mp:.2f}%")
print(f"Tempo ts = {ts:.3f} s")
print(f"Erro de regime (degrau) = {ess*100:.3f}%")
print(f"Kv = {Kv:.2f}")
print(f"Erro de rampa em k*G(s) = {er_rampa*100:.3f}%")
print(f"Erro de rampa em k*G(s)*C(s) = {er_rampa_clag*100:.3f}%")

# --- Bode ---
plt.figure(1)
Ws=np.logspace(-3,3,1000)
ctl.bode_plot(Gs, Ws, dB=True, label='G(s)')
ctl.bode_plot(Lk, Ws, dB=True, label='k*G(s)')
ctl.bode_plot(Lkc, Ws, dB=True, label='k*G(s)*C(s)')
plt.legend()

# --- Resposta ao degral ---
t1 = np.linspace(0, 3, 1000)
```

```
t1, y1 = ctl.step_response(Gf, t1)
t1, y2 = ctl.step_response(Gkf, t1)
t1, y3 = ctl.step_response(Hf, t1)
plt.figure(2)
plt.plot(t1, y1, label='G(s)')
plt.plot(t1, y2, label='k*G(s)')
plt.plot(t1, y3, label='k*G(s)*C(s)')
plt.title('Resposta ao degrau do sistema em malha fechada')
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)

# --- Resposta a Rampa ---
t = np.linspace(0, 3, 1000)
rampa = t   # r(t) = t
t_out, y1 = ctl.forced_response(Gf, T=t, U=rampa)
t_out, y2 = ctl.forced_response(Gkf, T=t, U=rampa)
t_out, y3 = ctl.forced_response(Hf, T=t, U=rampa)

plt.figure(3)
plt.plot(t_out, y1, label="Sa da G(s)")
plt.plot(t_out, y2, label="Sa da k*G(s)")
plt.plot(t_out, y3, label="Sa da k*G(s)*C(s)")
plt.plot(t_out, rampa, '--', label="Entrada rampa")
plt.title("Resposta    Rampa do sistema em malha fechada")
plt.xlabel("Tempo (s)")
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()

# --- Nyquist ---
plt.figure(4)
ctl.nyquist(Hf)
f_lim = 1 / (1 + 1j * np.array([0, 0, 1000, -1000]))
plt.plot(f_lim.real, f_lim.imag, 'mx')

plt.show()
```