Sistemas Realimentados - Trabalho 1

Nome: Guilherme Goes Zanetti

Data limite para entrega:14/9, meia-noite.

Referências para este trabalho além das notas de aula:

- Manual rápido do Matlab
- Resposta transitória e estacionária

Inicialização:

```
I=16; % Seu valor de I
[Y,G1,G2]=init_t1(I);
datetime('now')
```

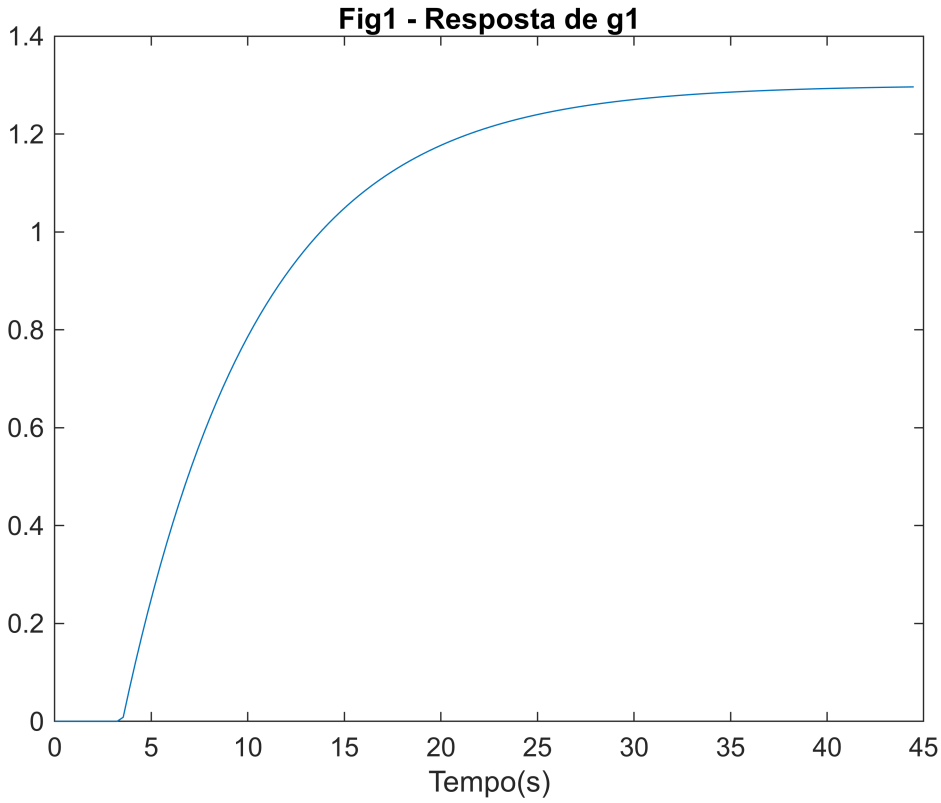
```
ans = datetime
   14-Sep-2023 23:13:11
```

```
warning off
```

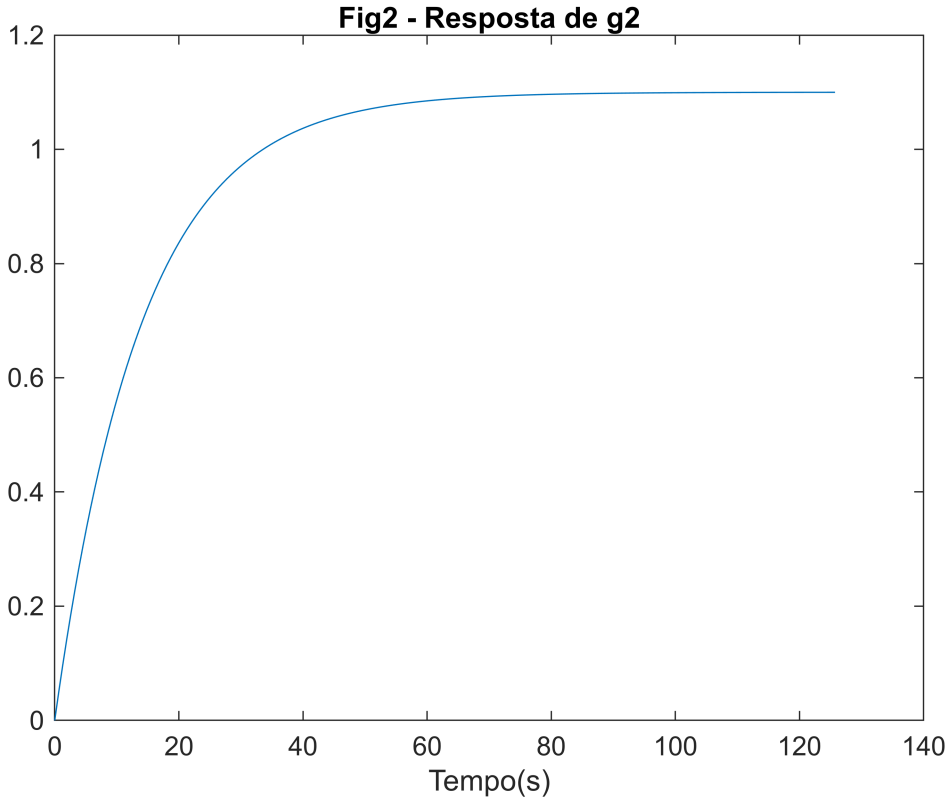
I) Relação entre resposta ao degrau e modelos

Observe nas figura 1,2,3 a resposta ao degrau unitário dos modelos g1, g2, g3.

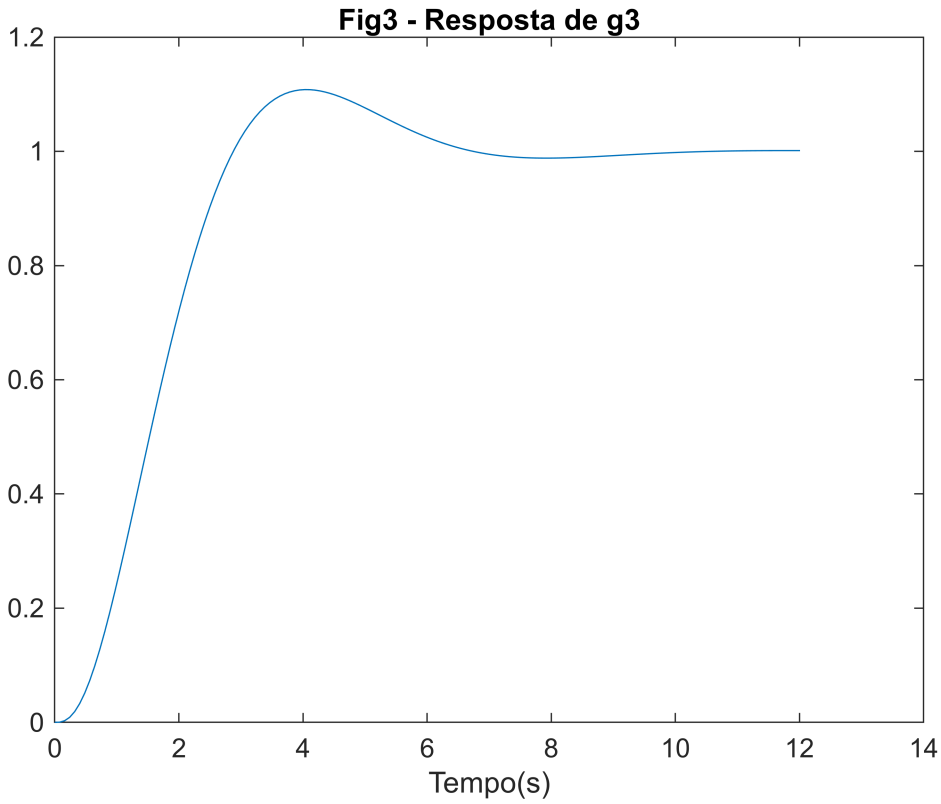
```
figure;plot(Y(1).t,Y(1).y);title('Fig1 - Resposta de g1');xlabel('Tempo(s)');
```




```
figure;plot(Y(2).t,Y(2).y);title('Fig2 - Resposta de g2');xlabel('Tempo(s)');
```



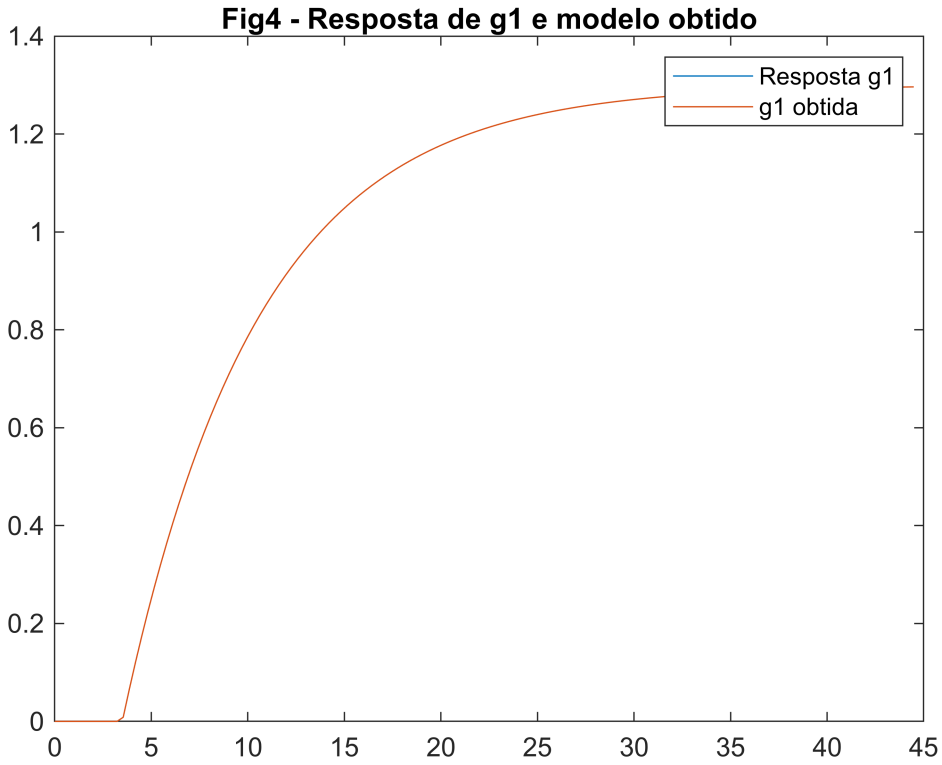
```
figure;plot(Y(3).t,Y(3).y);title('Fig3 - Resposta de g3');xlabel('Tempo(s)');
```



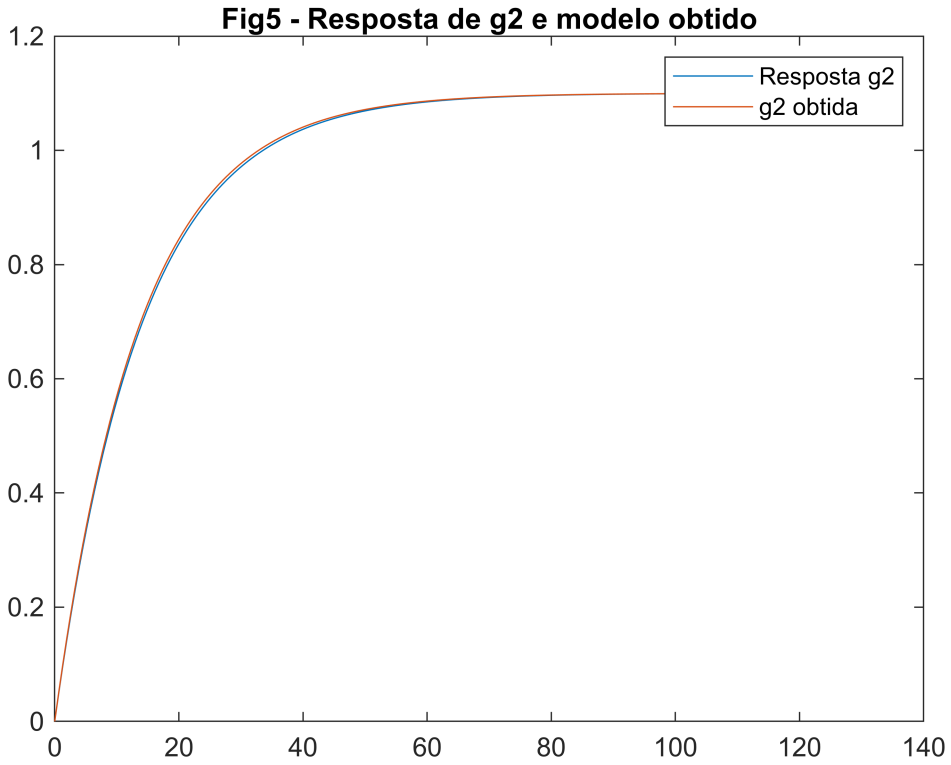
1) Obtenha os modelos g1,g2,g3 que geraram estas respostas ao degrau, defina-os abaixo e compare com as curvas fornecidas usando os comandos dados.

```
s = tf('s');
%Observando o gráfico de g1, irei criar um modelo de primeira ordem com
%tempo morto de 3s , ganho K=1,3 e constante de tempo t = 7,5s
g1=tf(1.3, [7, 1]);
g1.InputDelay=3.5;
%Observando o gráfico de g2, irei criar um modelo de primeira ordem sem tempo morto
% ganho K=1,1 e constante de tempo t = 13.7s
g2=tf(1.1, [13.7, 1]);
%Observando o gráfico de g3, irei criar um modelo de segunda ordem sem tempo morto
%erro nulo a entrada de degrau, sobreelevação = 10% e tempo de
%estabelecimento de 7s
up=10;
ts=7;
a=log(up/100);
zeta=sqrt(a^2/(pi^2+a^2));
wn=4/(ts*zeta);
g3=wn^2/(s^2+2*zeta*wn*s+wn^2);

y1=step(g1,Y(1).t);
y2=step(g2,Y(2).t);
```

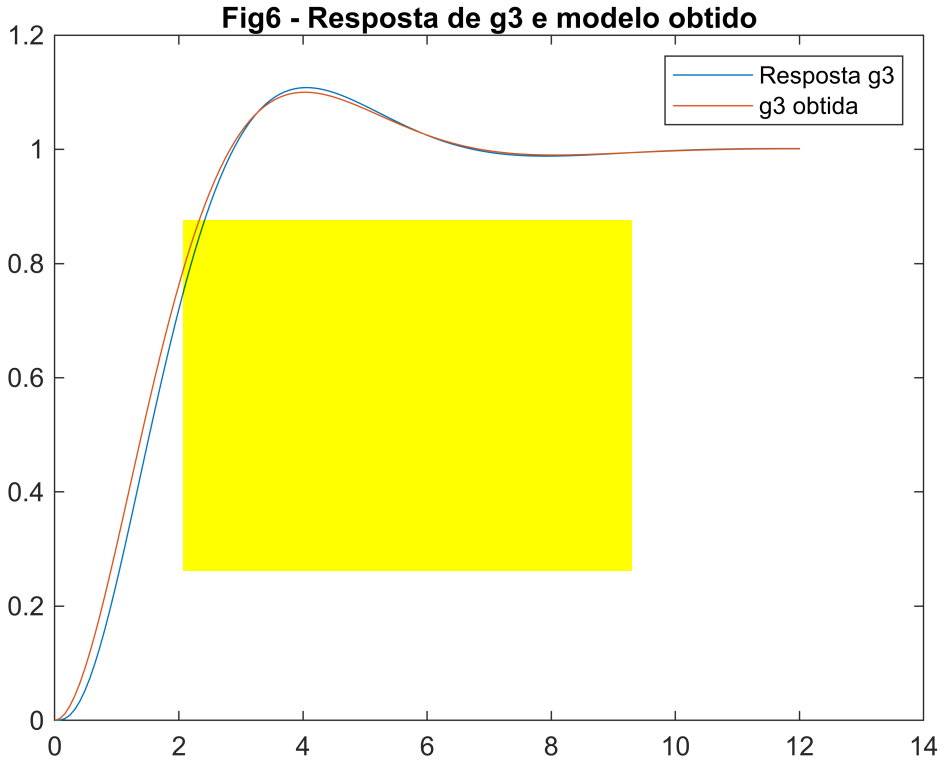
```
y3=step(g3,Y(3).t);

figure;plot(Y(1).t,Y(1).y,Y(1).t,y1);title('Fig4 - Resposta de g1 e modelo
obtido');legend('Resposta g1', 'g1 obtida');
```




```
figure;plot(Y(2).t,Y(2).y,Y(2).t,y2);title('Fig5 - Resposta de g2 e modelo
obtido');legend('Resposta g2', 'g2 obtida');
```




```
figure;plot(Y(3).t,Y(3).y,Y(3).t,y3);title('Fig6 - Resposta de g3 e modelo
obtido');legend('Resposta g3', 'g3 obtida');
```

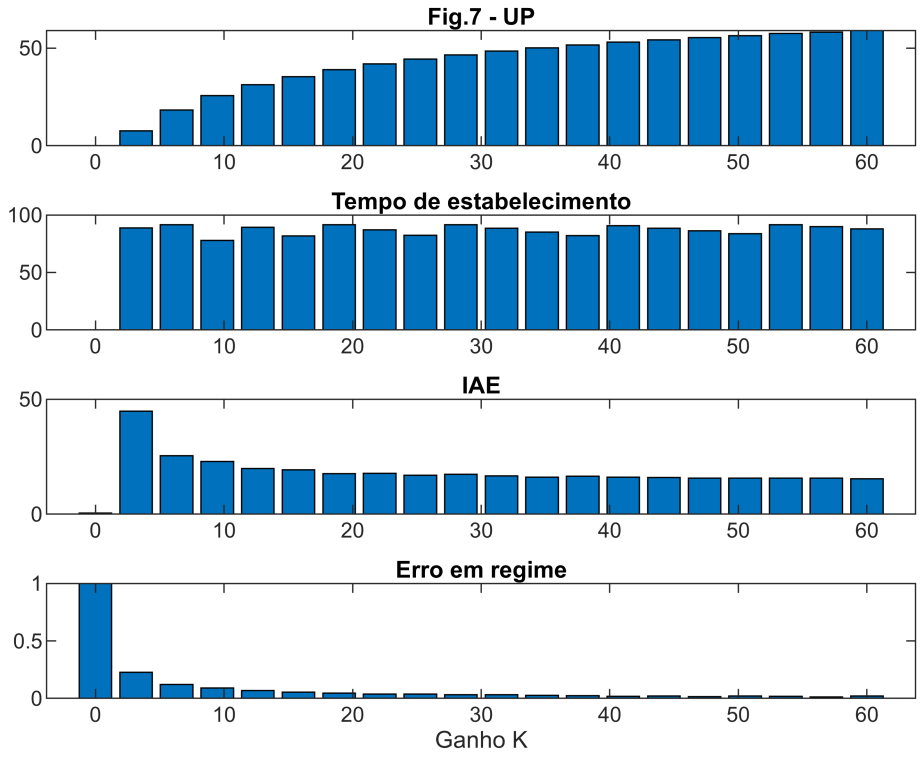


II) Desempenho em função do ganho K.

Seja a função de transferência G1 já definida.

Os comandos abaixo permitem obter os vários parâmetros de desempenho em malha fechada quando o ganho K varia. **Escolha kmax** de modo que todos gráficos mostrem claramente o efeito do ganho em cada parâmetro. Sugestão: escolha kmax de modo que UP vá de 0 a 60%.

```
kmax=60; % Escolhido kmax 60
up=[];ts=[];iae=[];erro=[];
K=linspace(0,kmax,20);
for i=1:length(K)
    m=feedback(K(i)*G1,1);
    S=stepinfo(m);
    [y,t]=step(m);
    up=[up;S.Overshoot];
    ts=[ts;S.SettlingTime];
    iae=[iae;trapz(t,abs(1-y))];
    erro=[erro;1-y(end)];
end

figure;
subplot(4,1,1);bar(K,up);title('Fig.7 - UP');
subplot(4,1,2);bar(K,ts);title('Tempo de estabelecimento');
subplot(4,1,3);bar(K,iae);title('IAE');
subplot(4,1,4);bar(K,erro);title('Erro em regime');xlabel('Ganho K');
```



2.1 Escolha o ganho K para o qual considera que houve uma boa resposta, informando o ganho K, UP, ts, IAE e o erro em regime correspondentes.

Resposta: Escolhi o ganho K = 9.47, que julgo como um bom ponto de equilibrio entre uma sobreelevação de 25,65% ainda não tão alta e um IAE e Erro em

regime razoavelmente baixos (IAE = 22.84 e ess = 0.09). Neste ganho ainda há um tempo de estabelecimento ótimo de 77.9 segundos.

2.2 Feche a malha com este ganho K, obtenha os polos de malha fechada e mostre que a resposta ao degrau é a especificada.

Resposta: Os polos de malha fechada do sistema são -0.0429 + 0.0991i e -0.0429 - 0.0991i.

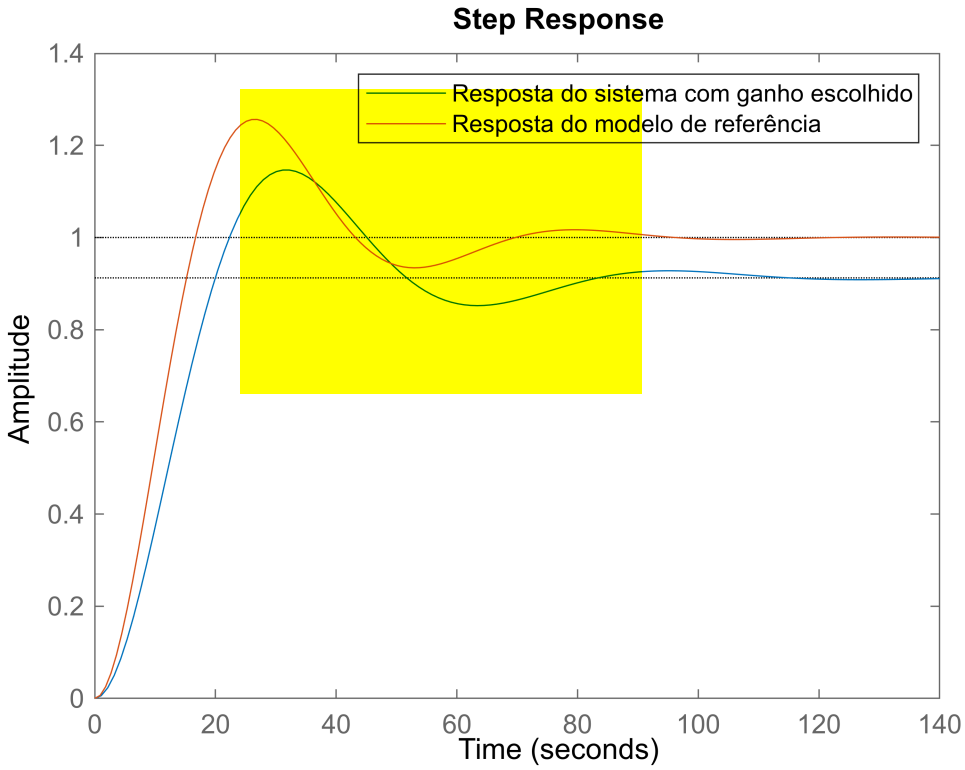
```
% comandos para mostrar a resposta
K = 9.47;
m=feedback(K*G1,1)
```

```
m =

         10.42
  ----------------------
  980 s^2 + 84 s + 11.42

Continuous-time transfer function.
Model Properties
```

```
P = pole(m)
```

```
P = 2×1 complex
  -0.0429 + 0.0991i
  -0.0429 - 0.0991i
```

```
figure;
%Criação de um modelo de referência com up = 25.65% e ts = 77.9
up=25.65;
ts=77.9;
a=log(up/100);
zeta=sqrt(a^2/(pi^2+a^2));
wn=4/(ts*zeta);
modelo_ref=wn^2/(s^2+2*zeta*wn*s+wn^2);

step(m, modelo_ref); legend("Resposta do sistema com ganho escolhido", "Resposta do
modelo de referência");
```



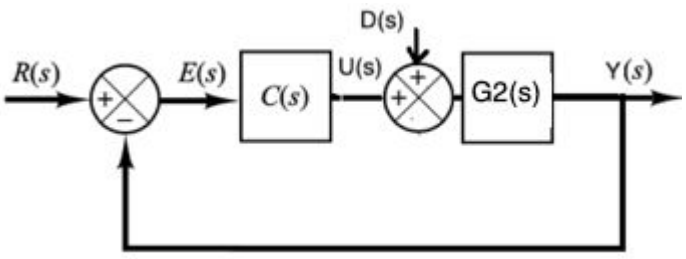
2.3 Calcule à mão o erro em regime usando G1 e o ganho K selecionado em 2.1, e compare com o valor obtido no gráfico na Fig7.

ess = lim s->0 (sE(s)) = lim s->0 s(R(s) - Y(s)) = lim s->0 s(R(s) - M(s)R(s)) = lim s->0 s(1/s - 10.42/(980s^2 + 84s + 11.42) * 1/s) = lim s->0 (1 - 10.42/980s² + 84s + 11.42) = 1-(10.42/11.42) = 0.0876

Que está de acordo com o valor de erro estacionário de 0.09 obtido na figura 7.

III) Projeto de controlador PID via métodos de sintonia baseados na resposta ao degrau

Use o método de sintonia Ziegler-Nichols e projete controladores C(s) da forma P, PI e PID para G2 já definida.



3.1) Compare a resposta ao degrau Y(s)/R(s) de cada controlador em termos de UP, ts, erro em regime, plotando a resposta de P,PI,PID na mesma figura.

```
cp=pidtuning(G2,'method','zie','type','P')
```

```
cp =

  Kp = 1

P-only controller.
Model Properties
```

```
mp = feedback(cp*G2, 1);
S=stepinfo(mp);
[y,t]=step(mp);
up=S.Overshoot;
ts=S.SettlingTime;
erro=1-y(end);

fprintf('\nControlador P:\nUP: %f\nts: %f\nErro em regime: %f\n', up, ts, erro)
```

```
Controlador P:
UP: 43.007186
ts: 40.718573
Erro em regime: 0.336390
```

```
cpi=pidtuning(G2,'method','zie','type','PI')
```

```
cpi =

            1
  Kp + Ki * ---
            s

  with Kp = 0.9, Ki = 0.0601

Continuous-time PI controller in parallel form.
Model Properties
```

```
mpi = feedback(cpi*G2, 1);
S=stepinfo(mpi);
[y,t]=step(mpi);
up=S.Overshoot;
ts=S.SettlingTime;
```

```
erro=1-y(end);

fprintf('\nControlador PI:\nUP: %f\nts: %f\nErro em regime: %f\n', up, ts, erro)
```

```
Controlador PI:
UP: 17.673937
ts: 48.486843
Erro em regime: 0.007665
```

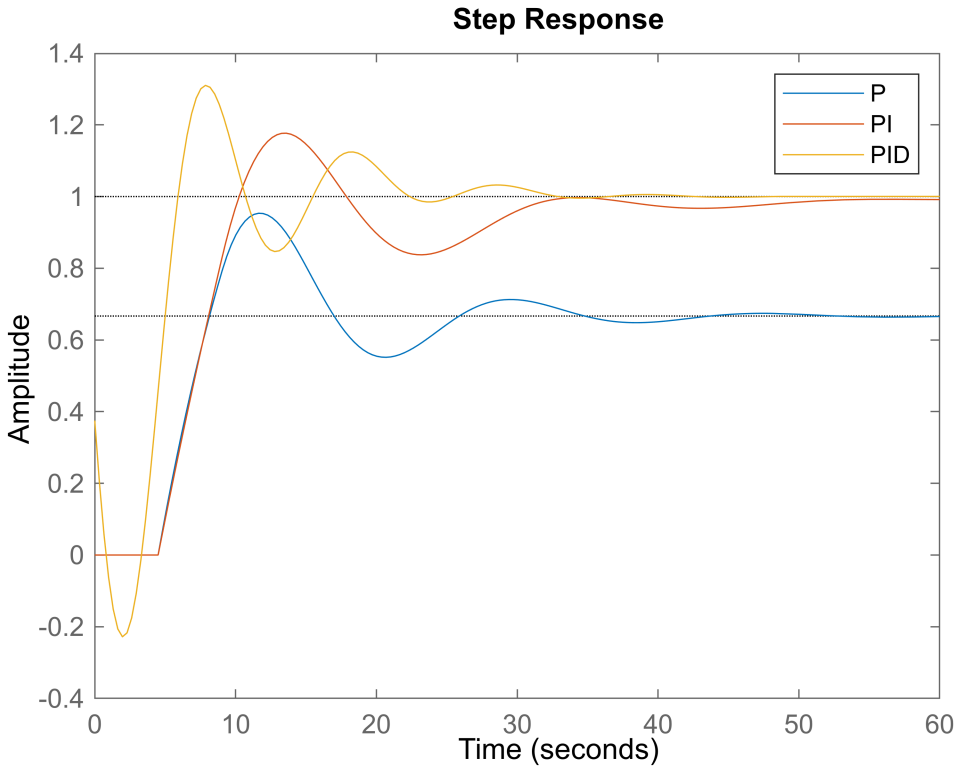
```
cpid=pidtuning(G2,'method','zie','type','PID')
```

```
cpid =

             1
  Kp + Ki * --- + Kd * s
             s

  with Kp = 1.2, Ki = 0.133, Kd = 2.7

Continuous-time PID controller in parallel form.
Model Properties
```

```
mpid = feedback(cpid*pade(G2, 2), 1);
S=stepinfo(mpid);
[y,t]=step(mpid);
up=S.Overshoot;
ts=S.SettlingTime;
erro=1-y(end);

fprintf('\nControlador PID:\nUP: %f\nts: %f\nErro em regime: %f\n', up, ts, erro)
```

```
Controlador PID:
UP: 31.016304
ts: 30.636620
Erro em regime: -0.002710
```

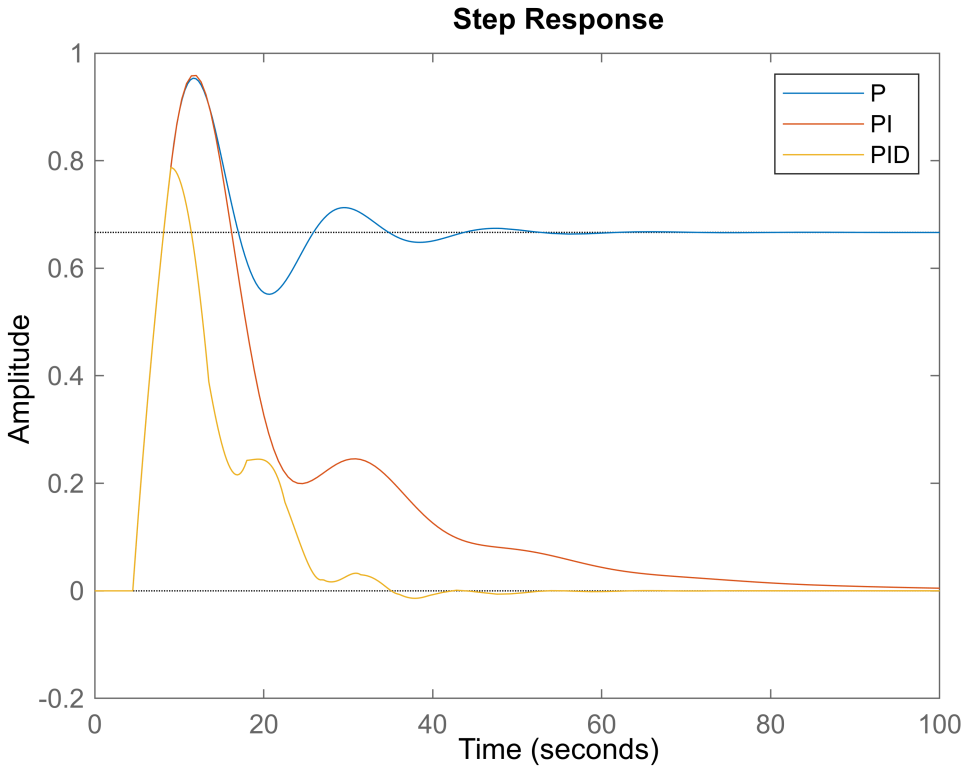
```
step(mp, mpi, mpid);legend('P', 'PI', 'PID');shg
```



Resposta: Pode-se ver que o controlador P gerado faz com que o sistema tenha uma elevada sobreelevação de 43% e um grande erro de regime de 0.33 (33% do valor total), ao se adicionar a componente integrativa com o controlador PI, houve uma grande melhora em ambos os valores (UP=17.67% e ess=0.008) porém o tempo de estabelecimento do sistema aumentou ligeiramente, indo de 40.7s para 48.5s. Já o controlador PID parece ter a resposta mais equilibrada, com os menores erros de regime e tempos de estabelecimento (ess=-0.002 e ts=30.6s) e uma sobreelevação intermediária entre o P e PI, de 31%.

3.2) Compare a resposta a um distúrbio Y(s)/D(s) de cada controlador, verificando qual faz a melhor rejeição ao distúrbio, plotando a resposta de P,PI,PID na mesma figura.

```
dp = feedback(G2, cp);
dpi = feedback(G2, cpi);
dpid = feedback(G2, cpid);

step(dp, dpi, dpid);legend('P', 'PI', 'PID');shg
```



Analizando os gráficos das respostas ao distúrbio para os três diferentes controladores, pode-se ver que o controlador P não consegue ter uma boa rejeição ao distúrbio, sendo que ele não tem seu efeito anulado em qualquer momento. Comparando os controladores PI e PID, o PID possui uma melhor rejeição ao distúrbio, tendo uma variação inicial menor e um retorno ao zero mais rápido do que o PI.

IV) Projeto de controlador PID via métodos de sintonia baseados em modelo

Projeto com G2:

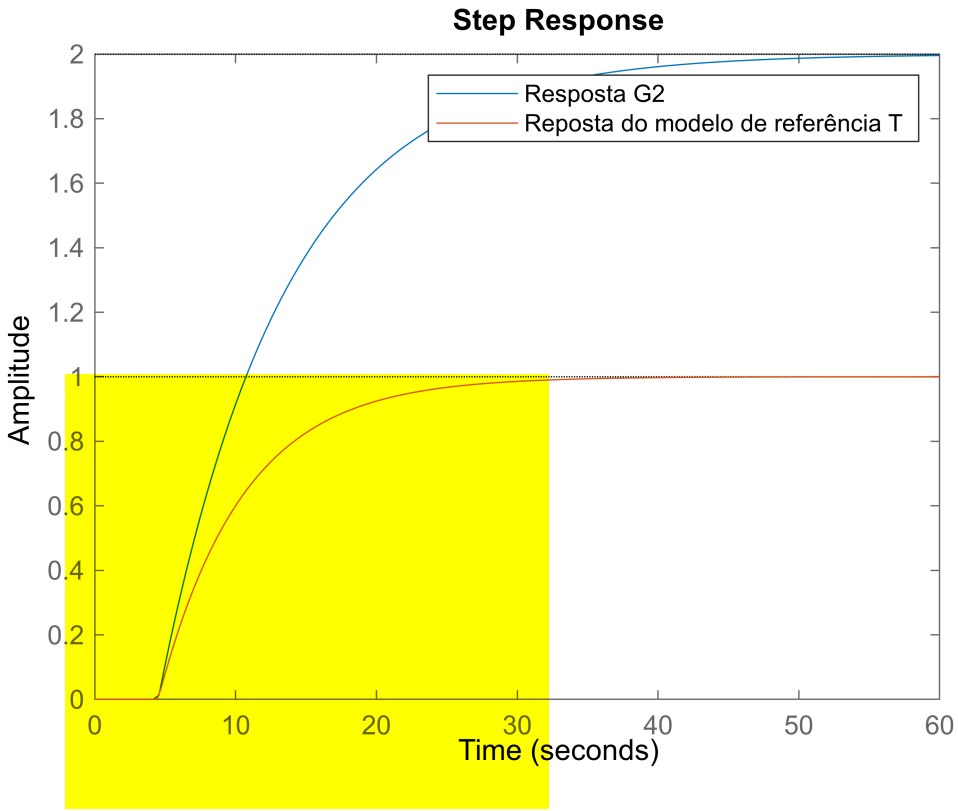
4.1) Seja o modelo G2(s) já definido. Plote em uma mesma figura a resposta ao degrau de G2(s) e de um modelo de referência T(s) que dê a resposta desejada em malha fechada para este processo, de forma que seja mais rápido e sem erro em regime.

```
%Criando modelo de referência de primeira ordem
%Com tempo morto de 4,5 segundos
%E constante de tempo de 6 segundos (menor que a constante de tempo
%original mas maior do que o tempo morto)
T = tf(1, [6 1], 'InputDelay', 4.5)
```

```
T =

                1
exp(-4.5*s) * -------
              6 s + 1
```

```
Continuous-time transfer function.
Model Properties
```

```
step(G2, T);legend("Resposta G2", "Reposta do modelo de referência T");
```



Escolhido um modelo de referência com ganho 1 para eliminar o erro de regime e uma constante de tempo aproximadamente 25% menor que a constante de tempo de malha aberta.

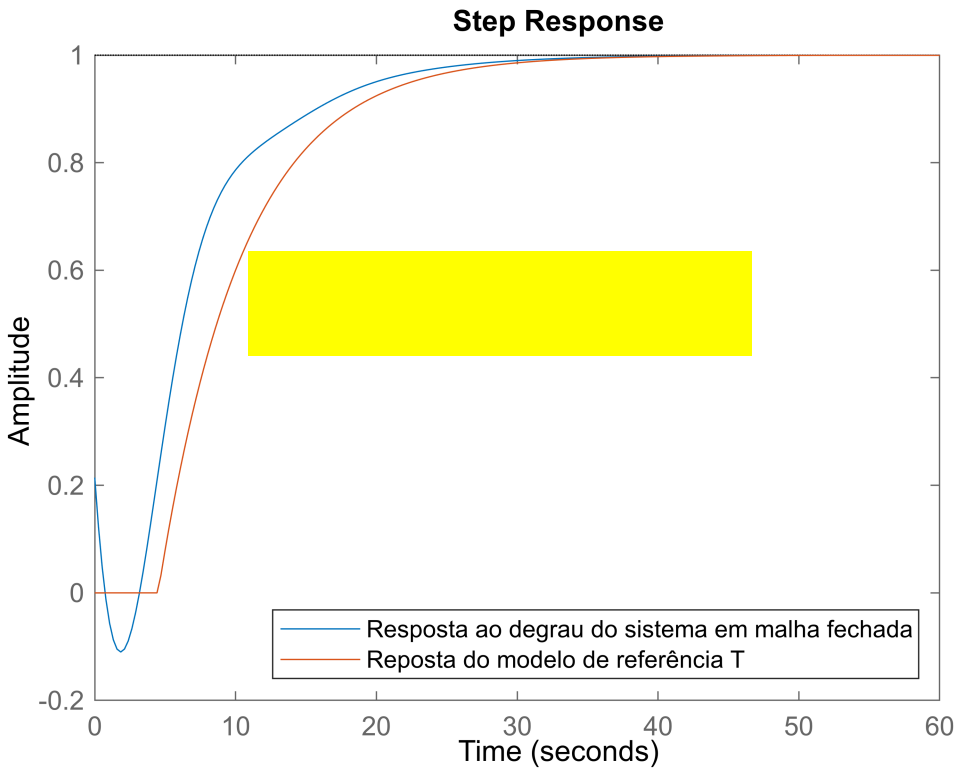
4.2 Explique as escolhas e os cálculos para obter os parâmetros do controlador C(s) pelo método de síntese direta ou IMC. Obtenha os ganhos do controlador C(s) e plote no mesmo gráfico a resposta ao degrau do sistema em malha fechada e do modelo de referência T(s), mostrando o atendimento das especificações.

```
%comandos matlab (ver slides de aula)
C2 = pidtuning(G2, 'method', 'lambda', 'type', 'PID', 'param', 6)
```

```
C2 =

             1
  Kp + Ki * --- + Kd * s
             s

  with Kp = 0.682, Ki = 0.0606, Kd = 1.23

Continuous-time PID controller in parallel form.
Model Properties
```

```
m = feedback(C2*pade(G2,2), 1);
```

```
figure;
step(m,T); legend("Resposta ao degrau do sistema em malha fechada", "Reposta do
modelo de referência T", 'location', 'southeast');
```



No método de sintese direta, é necessário apenas escolher o valor da constante de tempo de malha fechada para o sistema e fornecer a função de tranferência da planta G2. Com base nisso, cria-se um modelo de referência T com mesmo tempo morto de G2 e com a constante de tempo escolhida. Assim, ao se substituir na equação Gc = 1/G * T/(1-T) e usando a aproximação de padé de ordem 2 para se obter um PID, tem-se o controlador Gc que resulta numa resposta próxima de T. E como podemos ver no gráfico, a resposta de malha fechada do sistema com o controlador PID calculado com Kp = 0.833, Ki=0.0741 e Kd=1.5 é aproximadamente a resposta de referência desejada, com tempo de estabelecimento próximos e sem sobreelevação.

Projeto com G1:

4.3 Projete um controlador PID para G1(s) pelo método de síntese direta ou IMC de modo que seu tempo de estabelecimento seja reduzido à metade, com UP<5%. Plote a saída Y(s) para malha aberta e malha fechada, mostrando o atendimento da especificação, e mostre os valores de UP, ts e IAE.

```
[C1pid, iae] = pidtuning(G1, 'method', 'lambda', 'type', 'PID', 'param', 36)
```

```
C1pid =

           1
```

```
  Kp + Ki * --- + Kd * s
            s

  with Kp = 2.12, Ki = 0.0253, Kd = 24.7

Continuous-time PID controller in parallel form.
Model Properties
iae = 33.9415
```

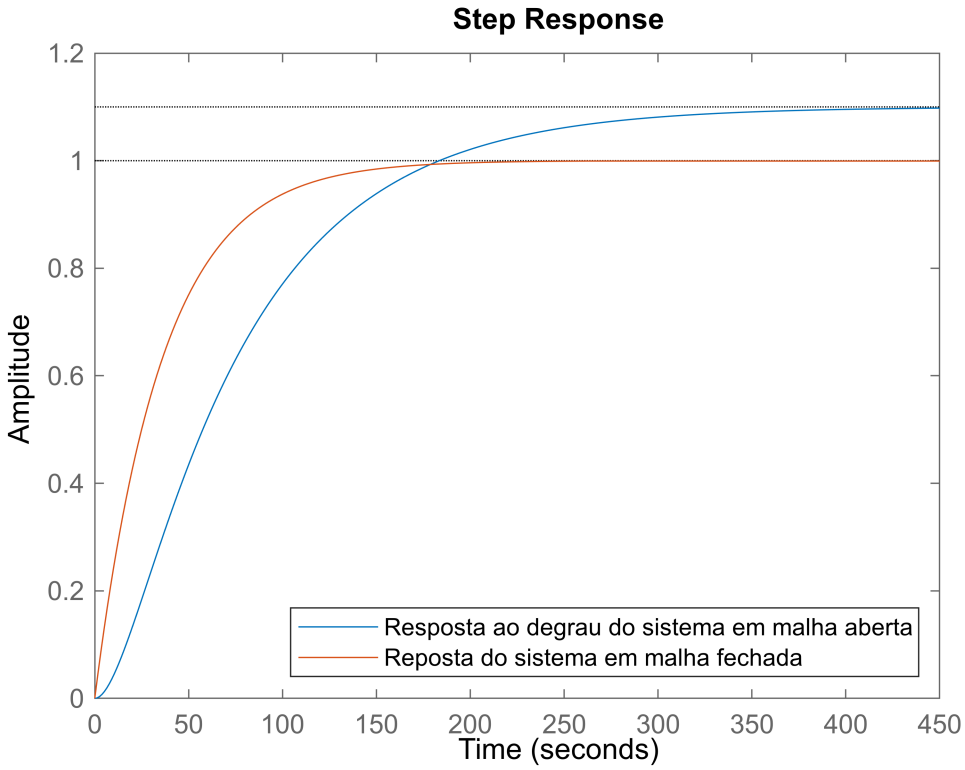
```
info = stepinfo(G1)
```

```
info = struct with fields:
         RiseTime: 159.0919
    TransientTime: 289.4646
     SettlingTime: 289.4646
      SettlingMin: 0.9918
      SettlingMax: 1.0991
        Overshoot: 0
       Undershoot: 0
             Peak: 1.0991
         PeakTime: 514.4896
```

```
mpid = feedback(C1pid*G1, 1);
info2 = stepinfo(mpid)
```

```
info2 = struct with fields:
         RiseTime: 79.0922
    TransientTime: 140.8347
     SettlingTime: 140.8347
      SettlingMin: 0.9045
      SettlingMax: 0.9993
        Overshoot: 0
       Undershoot: 0
             Peak: 0.9993
         PeakTime: 263.5999
```

```
step(G1, mpid);legend("Resposta ao degrau do sistema em malha aberta", "Reposta do
sistema em malha fechada", 'location', 'southeast');
```



Para determinar o lambda para a sintese direta, foi especificado um tempo de estabelecimento de 145 segundos (metade de 289 segundos), que resulta numa constante de tempo de 36 segundos (145/4). Assim, foi calculado o PID para o lambda 36 e se obteve Kp=2.12, Ki=0.0253 e Kd=24.7. Ao se fechar a malha com esse controlador, obtem-se um tempo de estabelecimento de 140 segundos, próximo dos 145 desejados. Além disso, a sobreelevação é zero, menor do que 5% e o IAE calculado para o sistema em malha fechada é de 33,9.

4.4 Repita para um controlador PI, explicando a diferença de projeto para o controlador PID, bem como a diferença entre os valores de UP, ts e IAE obtidos aqui e em 4.3

```
[C1pi, iae] = pidtuning(G1, 'method', 'polealoc', 'type', 'PI', 'param', [3 120])
```

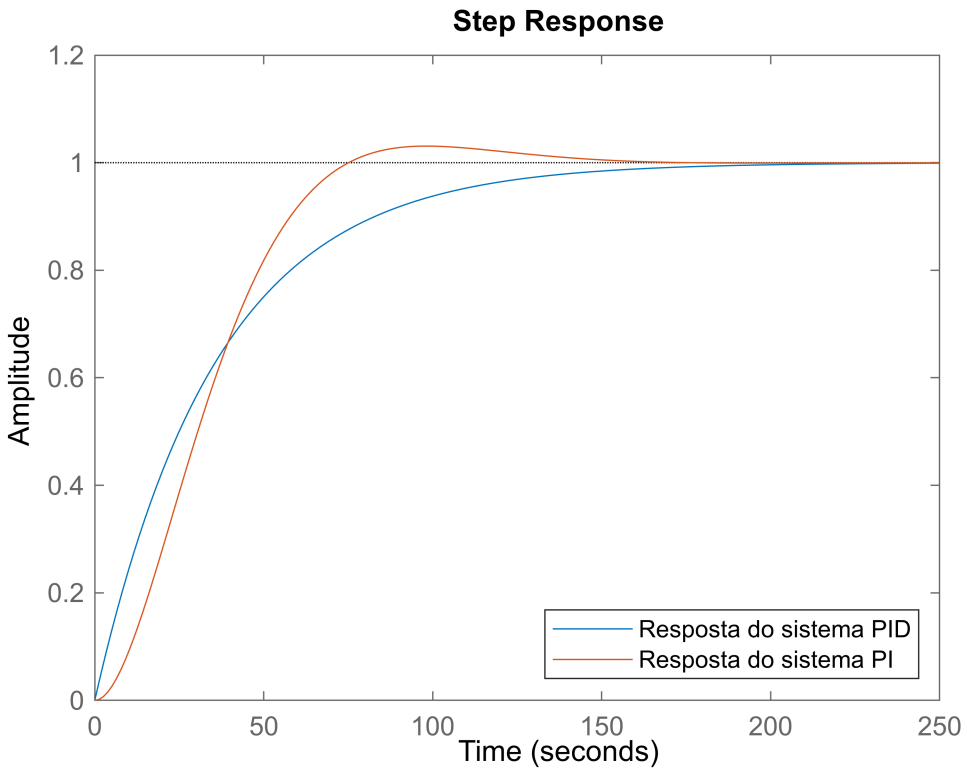
```
C1pi =

           1
  Kp + Ki * ---
           s

  with Kp = 2.05, Ki = 0.0294

Continuous-time PI controller in parallel form.
Model Properties
iae = 34.6029
```

```
mpi = feedback(C1pi*G1, 1);
```

```
info3 = stepinfo(mpi)
```

```
info3 = struct with fields:
         RiseTime: 47.2748
    TransientTime: 121.7539
     SettlingTime: 121.7539
      SettlingMin: 0.9019
      SettlingMax: 1.0309
        Overshoot: 3.0879
       Undershoot: 0
             Peak: 1.0309
         PeakTime: 98.1120
```

```
step(mpid, mpi);legend("Resposta do sistema PID", "Resposta do sistema PI",
'location', 'southeast');
```



Para criar um controlador PI para um sistema de segunda ordem, como o G1, é necessário usar o método polealoc em vez do lambda. Neste método, são escolhidos como parâmetros a sobreelevação e o tempo de estabelecimento. Escolhendo esses valores como 3 e 120 respectivamente, pode-se ver que com o PI o sistema tem uma resposta mais rápida do que com PID, com uma sobreelevação baixa de 3% (<5%). Entretando, o IAE do sistema com PI é ligeiramente maior, tendo valor de 34,6 em comparação com 33,9 do PID.

Essa resposta mais rápida do PI pode ser explicada pois foi utilizado um modelo de referência de segunda ordem, em comparação com o de primeira ordem que é o único disponivel para o método lambda implementado. Assim, foi possível ter dois graus de liberdade ao se projetar o controlador, em vez de se ter apenas o parâmetro lambda.