Tutorial sobre MATLAB aplicado aos problemas de controle

Prof. Thiago F. K. Cordeiro

Atenção: Assume-se que o leitor já possui um conhecimento prévio sobre MATLAB. Este tutorial apenas ensina comandos relacionados à função de transferência e sistemas de controle.

1) Representando sistemas dinâmicos no MATLAB

Pode-se definir um sistema dinâmico usando 3 representações distintas. A primeira delas, TF (*Transfer Function* - função de transferência), representa a relação entre entrada e saída através de uma razão de polinômios. A segunda representação, chamada de ZPK (zero, polo, ganho), é equivalente à primeira, mas mostra a função de transferência como um produto de zeros e ganho sobre um produto de polos. A terceira representação, SS (*state space* – espaço de estados) utiliza o conceito de espaço de estados, e não será estudada aqui.

A função *tf(num,den)* tem, como entrada, o polinômio do numerador e o polinômio do denominador. Deve-se representar os polinômios do numerador e do denominador através de vetores. Para representar um polinômio de grau *n* é necessário usar um vetor de tamanho *n* + 1, em que o primeiro elemento é o coeficiente de s^n, o segundo é o coeficiente de s^{n-1}, …, e o último elemento é o coeficiente de s^0. Termos inexistentes no polinômio significam coeficientes nulos. Por exemplo, se quisermos representar a função de transferência abaixo:

$$G_1 = \frac{1}{s^2 + s}$$

utilizaremos a sequência de comandos descrita abaixo:

```
>> num = 1;
>> den = [1 1 0];       %s^2 + s + 0, o ultimo coeficiente e nulo
>> sys1 = tf(num, den)

Transfer function:
   1
-------
s^2 + s
```

A função *zpk(zeros,polos,ganho)* possui como entrada uma lista de zeros, uma lista de polos e o ganho que multiplica tudo. Cada uma das duas listas é definida utilizando vetor, cujos elementos são os valores dos zeros ou os polos. O ganho é definido por um escalar. Por exemplo, se quisermos representar a função de transferência abaixo:

$$G_2 = \frac{1}{s(s + 1)}$$

utilizaremos a sequência de comandos descrita abaixo:

```
>> z = [];           %nao ha zeros, vetor(conjunto) vazio
>> p = [0 -1];       %dois polos, 0 e -1 (s e s+1)
>> k = 1;            % ganho unitario
>> sys2 = zpk(z,p,k)

Zero/pole/gain:
   1
-------
s (s+1)
```

Veja que, na verdade, G_1 e G_2 são a mesma função de transferência, mas escritas de formas distintas.

O MATLAB permite a conversão entre uma e outra representação. Para isso, basta utilizar as mesmas funções já estudadas, *tf* para obter o modelo em forma de polinômios e *zpk* para obter o modelo fatorado, colocando como entrada o sistema obtido de outra forma. Por exemplo:

```
>> sys3 = tf(sys2)     % converte sys2 de zpk para tf

Transfer function:
   1
-------
s^2 + s
```

Da mesma forma, também é possível mudar a representação de ou para espaço de estados.

1.1)**Ligando sistemas em série, paralelo ou efetuando realimentação**

É possível ligar sistemas em série, paralelo ou em realimentação negativa usando, respectivamente, os comandos *series*, *parallel* e *feedback*. Em todos os casos, a entrada da função são os dois sistemas a serem conectados.

Exemplo de conexão em série, em que a saída de *sys1* é conectada na entrada do *sys2*:

```
>> sys4 = series(sys1,sys2)

sys4 =

       1
  -----------
  s^2 (s+1)^2
```

Conexão em paralelo, em que *sys1* e *sys2* recebem a mesma entrada, e a saída de ambos é somada ao final:

```
>> sys4 = parallel(sys1,sys2)

sys4 =

   2 s (s+1)
  -----------
  s^2 (s+1)^2
```

Ligando o sistema *sys1* em realimentação unitária negativa:

```
>> sys4 = feedback(sys1,1)

Transfer function:
   1
-----------
s^2 + s + 1
```

Para gerar realimentação positiva, deve-se trocar o sinal do segundo termo. Por exemplo, abaixo o comando para se gerar uma realimentação unitária positiva.

```
>> sys5 = feedback(sys1,-1)

Transfer function:
   1
-----------
s^2 + s + 1
```

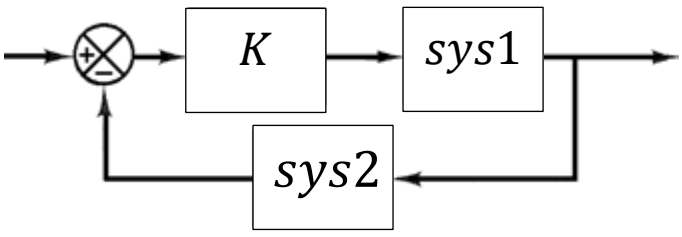
Para uma realimentação não unitária, basta colocar a função de transferência no segundo parâmetro.

```
>> sys4 = feedback(sys1,sys2)

sys4 =

             s (s+1)
 ---------------------------------------------
 (s^2 + 2.6s + 2.081) (s^2 - 0.6005s + 0.4805)
```

É bastante comum, em sistemas realimentados, haver um ganho K ajustável logo após o somador:



Podemos incluir facilmente o *K*, conforme ilustra o exemplo abaixo. No exemplo, multiplica-se sys1 pelo ganho K = 2, e faz-se uma realimentação unitária.

```
>> sys4 = feedback(2*sys1,1)

sys4 =

       2
  -----------
  s^2 + s + 2
```

2) **Obtendo a resposta ao impulso, degrau, e outras entradas**

Pode-se obter facilmente a resposta do sistema (seja ele TF, ZPK ou SS) a uma entrada impulso ou degrau, utilizando, respectivamente, as funções *impulse* e *step*. A forma mais simples de se usar as funções é chama-las passando apenas o sistema como parâmetro:

```
>> impulse(sys1)
>> step(sys1)
```

Também é possível usar as funções de transferência obtidas por conexão em série e em paralelo, assim como por realimentação. Em todos os casos, também é possível incluir um ganho K multiplicando a função de transferência pelo valor de K.

Caso se deseje salvar a figura para inseri-la em algum trabalho, recomenda-se salvá-la usando a própria interface do MATLAB, clicando no botão de salvar (disquete). O formato .PNG é adequado para inserir a figura no Word, enquanto que o formato .EPS é adequado para trabalhar com TeX. O formato .JPG não é apropriado para esse timo de imagem, pois sua técnica de compressão gera artefatos (defeitos) na imagem.

Pode-se também salvar a janela do gráfico que o MATLAB gera escolhendo o formato .FIG. Nesse caso, ao abrir o arquivo, a janela reaparece no MATLAB, permitindo todo o tipo de ajuste existente na interface dessa janela, como zoom, adição/remoção/alteração de legendas, adição e remoção de dados e etc.

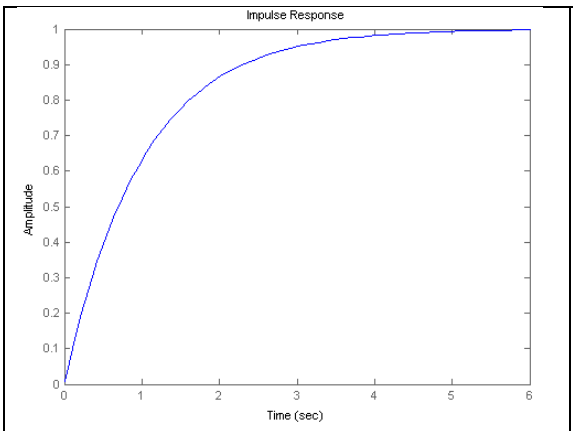


Figura 1 – Resposta ao impulso.

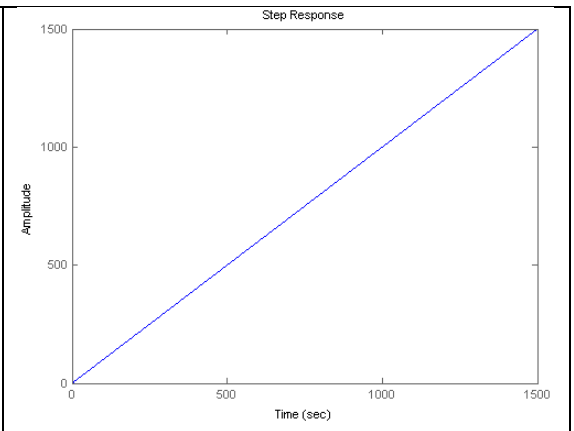


Figura 2 – Resposta ao degrau.

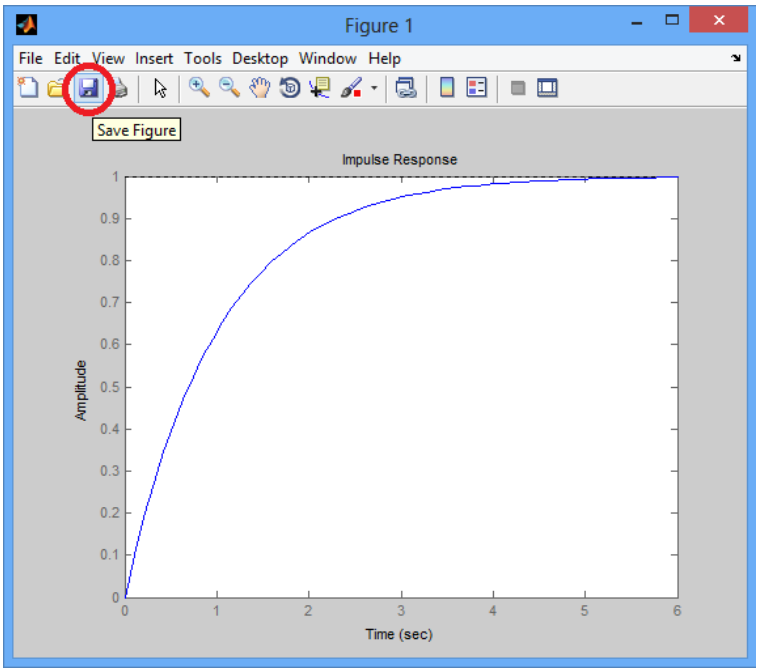


Figura 3 – Salvando uma figura.

As funções *impulse* e *step* podem receber mais parâmetros. Pode-se, por exemplo, definir o tempo final do gráfico. Caso se deseje que o gráfico da resposta ao impulso vá de zero a 10 segundos, ao invés de 0 a 6, basta usar o comando abaixo:

```
>> impulse(sys1, 10)
```

Para gerar o gráfico de impulso e degrau, as funções geram uma quantidade pré-determinada de pontos, e liga esses pontos com uma reta. Nem sempre essa quantidade de pontos é adequada. Por exemplo, pode-se estar interessado em avaliar com precisão o máximo sobre-sinal de uma resposta ao degrau. Por isso, as funções *impulse* e *step* permitem que se definam os instantes de "amostragem" da resposta. Por exemplo, para gerar um gráfico da resposta ao impulso que comece em 0, vá até 6 segundos e que possua intervalo de amostragem de 0,01 segundo, pode-se usar os comandos abaixo. A primeira linha gera um vetor contendo os elementos 0; 0,01; 0,02; … ; 5,99; 6,00.

```
>> t = 0:0.01:6;
>> impulse(sys1, t)
```

Algumas vezes é necessário ter acesso aos valores da saída do sistema, ao invés de simplesmente observá-los na forma de gráfico. Abaixo é ilustrado um exemplo de como coletar a resposta ao impulso, verificar qual é o valor máximo de saída e descobrir em que instante ocorre.

```
>> [y, t1] = impulse(sys1, t);    % saida: valores e instantes
de tempo
>> [val_max, indice_max] = max(y) % obtem o maior valor da
saida, e em qual índice do vetor esse valor ocorreu
```

```
val_max =

    0.9975

indice_max =

   601

>> t1(indice_max)

ans =

     6
```

Na primeira linha, armazena-se no vetor *y* o valor de saída em cada instante de amostragem, e no vetor *t1* cada valor de tempo (nesse caso, como fornecemos *t* na entrada, ele simplesmente retorna o mesmo vetor). No segundo comando, varre-se o vetor que contém todos os valores de saída, buscando o valor mais alto. Obtém-se também qual o índice do vetor y (ou seja, em que posição do vetor y) ocorre esse valor mais alto. Veja que o índice do vetor é algo interno à linguagem de programação, enquanto que o que interessa a um projetista é o instante de tempo. O tempo é obtido usando o vetor t1. Colocando o mesmo índice em y e t1, obtemos respectivamente um valor de saída e o instante de tempo que essa saída ocorreu.

Comentários:

a) Pode-se questionar porque o MATLAB não permite que se escreva y(0.09) para se obter a resposta *y* em *t* = 0,09. Deve-se lembrar que *y* é um vetor e, em linguagens de programações diversas, os elementos do vetor só podem ser enumerados usando inteiros positivos.
b) Neste exemplo, *t1* é simplesmente uma cópia de *t*. Obter *t1* é útil quando se usa os comandos *impulse* ou *step* sem inserir os instantes de amostragem.

Veja que, no exemplo de busca dado, o maior valor foi obtido em 6 segundos, que é o instante final da simulação. Isso faz sentido, já que a resposta ao impulso é monotônica crescente. Então o maior valor ocorre no maior valor de tempo considerado, ou seja, em 6 segundos. O valor final da resposta (ou seja, o valor de y para $t \rightarrow \infty$) é de 1. Pode-se verificar que a resposta já está próxima a esse valor em t = 6.

Além da entrada degrau, existem outras entradas comuns em controle, como rampa e parábola unitárias, e funções senoidais. Entretanto, o MATLAB não possui funções prontas para essas entradas.

Podemos contornar esse fato utilizando a função *lsim*, que permite que se forneça ao sistema uma entrada qualquer. Ao usar essa função, passa-se como parâmetro o sistema, um vetor contendo amostras do sinal de entrada e um vetor contendo os instantes de tempo referentes às amostras.

Abaixo são dados dois exemplos, em que são aplicadas as entradas rampa e parábola unitárias (respectivamente $u(t) = t$ e $u(t) = 1/2 * t^2$). O exemplo abaixo utiliza o vetor tempo que já foi criado em um exemplo anterior. Destaca-se que, como o MATLAB utilizará integração

numérica para calcular a saída, é importante que o sinal de entrada seja amostrado em uma taxa suficientemente alta.

Para gerar a entrada parábola, destaca-se que foi utilizada uma operação elemento a elemento (a operação .^2), que eleva individualmente os termos do vetor ao quadrado. Caso não fosse inserido o ponto (ou seja, se fosse utilizada a operação ^2), o MATLAB tentaria, sem sucesso, efetuar a multiplicação matricial t*t.

```
>> u1 = t;
>> lsim(sys1,u1,t)
>> u2 = 0.5*t.^2;
>> lsim(sys1,u2,t)
```

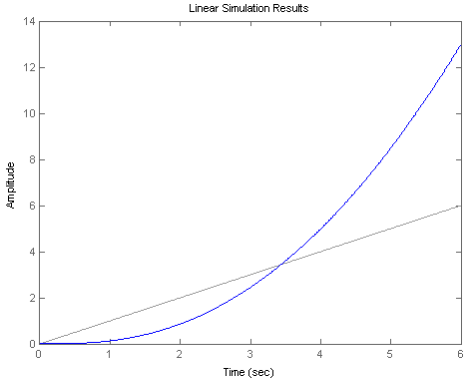


Figura 4 – Resposta à rampa unitária.

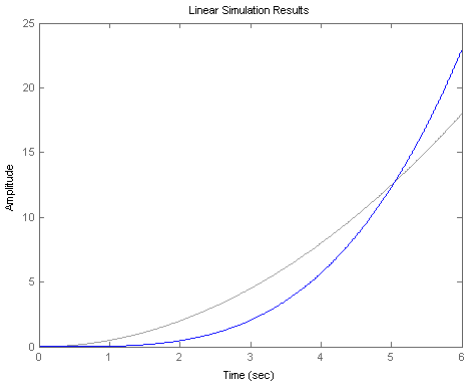


Figura 5 – Resposta à parábola unitária.

Por fim, para consultar quais as entradas e saídas possíveis dessas e de outras funções, pode-se usar o comando *help nome_da_funcao*. O MATLAB possui uma excelente documentação, e recomenda-se consultá-la sempre que necessário.

3) Análise de sistemas dinâmicos através de métodos gráficos

O MATLAB permite gerar gráficos para análise. Por exemplo, pode-se gerar o lugar geométrico das raízes através do comando abaixo:

```
>> rlocus(sys1)
```

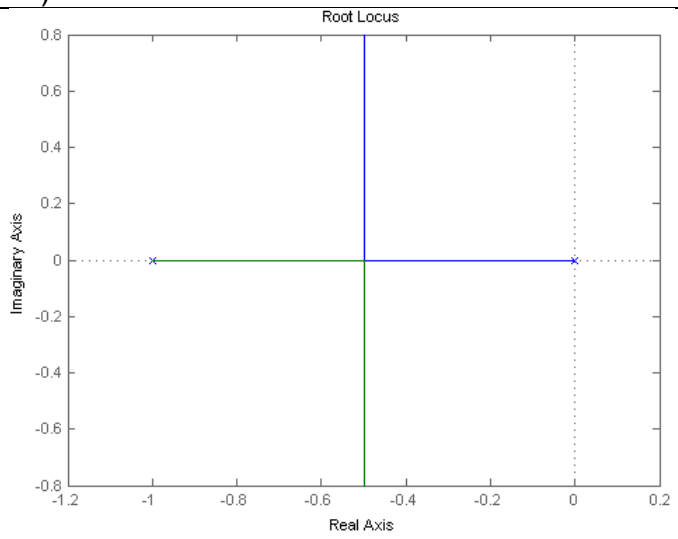


Figura 6 – LGR.

O MATLAB usou duas cores nesse gráfico para destacar o caminho que cada um dos polos faz.

Pode-se usar o comando *sgrid* para traçar retas de amortecimento constante e curvas de frequência natural não amortecida constante.

```
>> sgrid
```

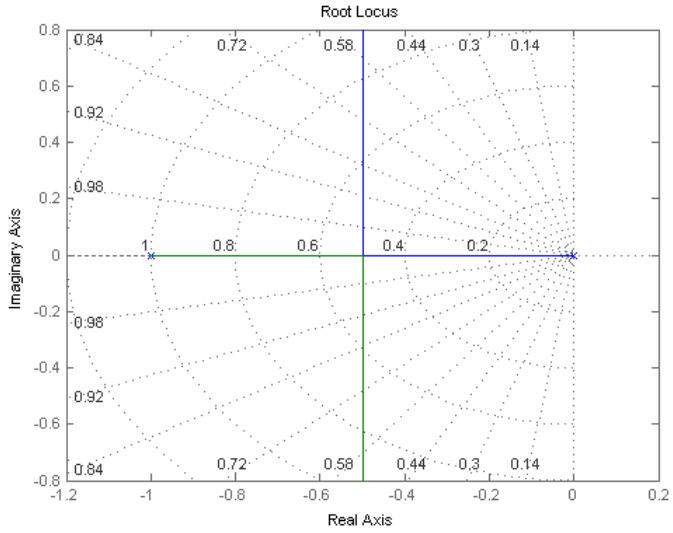


Figura 7 – LGR com curvas de amortecimento e frequência de amortecimento constante.

Pode-se usar a ferramenta *Data Cursor* para clicar em um ponto qualquer do LGR e descobrir o ganho K que posiciona os polos na posição escolhida.

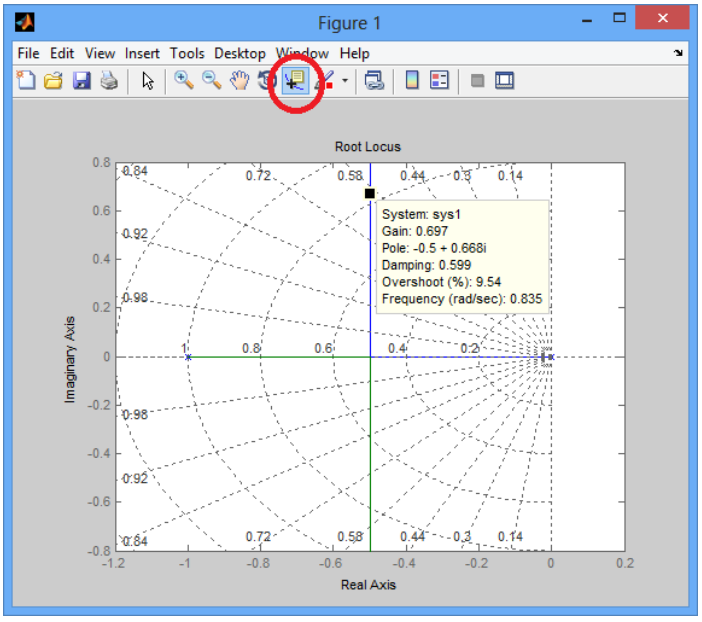


Figura 8 – Utilizando a ferramenta Data Cursor

Nem sempre o cursor ficará posicionado onde se deseja. Isso se deve ao fato de que a função *rlocus* escolhe os valores K automaticamente para desenhar o LGR, de forma a gerar um

gráfico suave e, ao mesmo tempo, reduzir carga computacional. Da mesma forma que fornecemos os instantes de tempo à função *impulse*, podemos fornecer valores de K desejados à função *rlocus*. No exemplo abaixo, são gerados valores de K com intervalo de 0,001.

```
>> K = 0:0.001:5;
>> rlocus(sys1, K)
```

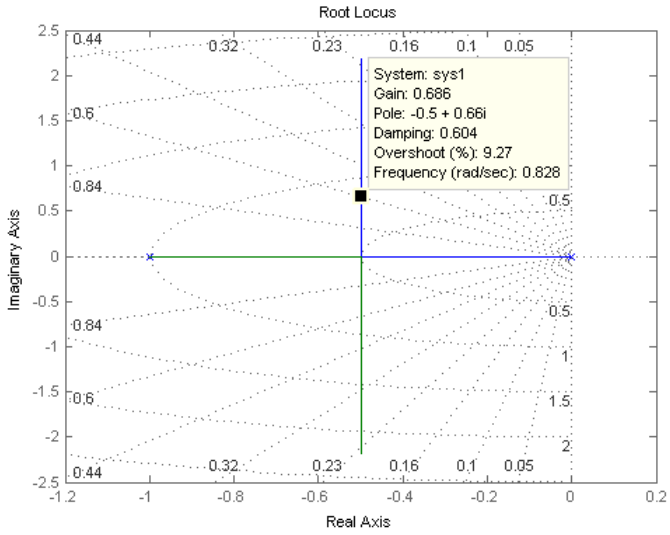


Figura 9 – LGR com mais pontos, e ponto com amortecimento 0,6 destacado. Esse ponto é obtido escolhendo o ganho K = 0,686.

Verificou-se então que obteremos o desempenho desejado se realimentarmos o sistema e ajustarmos o ganho para K = 0,686. Fazemos isso conforme comando abaixo:

```
>> sys6 = feedback(0.686*sys1,1)

Transfer function:
     0.686
---------------
s^2 + s + 0.686
```

O sistema realimentado tem que ter o par de polos aonde projetamos, ou seja, em -0.5 ± 0.66j. Pode-se confirmar a posição dos pólos, e seus amortecimentos utilizando o comando damp (da palavra damping, amortecimento, em inglês).

```
>> damp(sys6)

      Eigenvalue                Damping    Freq. (rad/s)

 -5.00e-001 + 6.60e-001i    6.04e-001     8.28e-001
 -5.00e-001 - 6.60e-001i    6.04e-001     8.28e-001
```

O nome *eigenvalue* (autovalor, em inglês), aqui, pode ser entendido como sinônimo de polo. Veremos qual a relação entre ambos quando estudarmos espaço de estados.

O sistema com ganho de 0,686 e realimentação unitária estudado, então, apresenta um par de polos complexos conjugados em (-0,5 ± 0,66j). O amortecimento 0,604 e frequência natural não amortecida de 0,828 rad/s são valores de referência, considerando que o par de polos são os únicos polos do sistema, e que não há zeros. No exemplo, isso realmente é verdade, mas não é o usual.

Verifique que os polos de malha fechada e, dessa forma, os valores de amortecimento e frequência natural desses polos, estão de acordo com os obtidos com o projeto via LGR, o que é esperado, já que é justamente esse o objetivo do LGR: escolher a posição dos polos de malha fechada.

4) **Diagramas de Bode**

Outra ferramenta gráfica é o diagrama de Bode. Esse gráfico pode ser plotado com o comando abaixo:

```
>> bode(sys1)
```

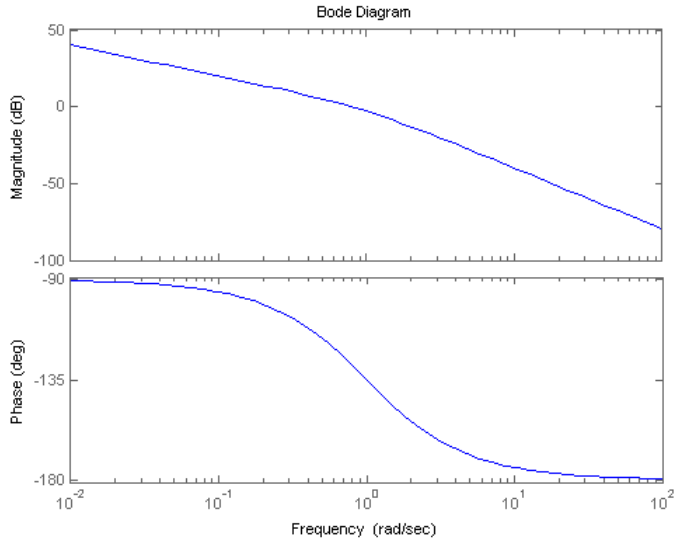


Figura 10 – Diagrama de Bode.