# A First Tutorial for the Matlab Control Systems Toolbox

**This is a first introduction to the capabilities of the Matlab Control System Toolbox. The objective of this tutorial is to give the reader sufficient understanding of the toolbox so that they perform computations to aid in understanding class material or perform various exercises or projects. This first tutorial is limited in scope to continuous time system represented by Laplace transforms. Additional tutorials will be presented as more advanced material is needed.**

**For this tutorial the green text represents input to the Matlab command line (or entries in an m-file) while the blue text shows the displayed result to that input on Matlab's command window. Both green and blue text are indented to emphasize the Matlab environment from the tutorial text and were automatically generated for parts of this document using the notebook feature of Matlab.**

**This tutorial is divided into the following five sections**

## Section I: Defining Transfer Function Objects

**Matlab allows us to define a polynomial as a vector. For instance the polynomial**

$$D(s) = s^3 + 3s^2 + 4^s + 10$$

**is represented as**

```
        Ds = [1 3 4 10]

       Ds =
            1      3      4     10
```

**Ds as entered is a row vector with the coefficients entered in descending order. "Ds" can be any name you want to use. Remember that Matlab is case sensitive.**

**Matlab allows us to find the roots of the polynomial by using the command:**

```
        roots(Ds)

       ans =
         -2.8338
         -0.0831 + 1.8767i
         -0.0831 - 1.8767i
```

**For polynomials such as: $Q(s) = s^3 + 3s^2 = s^2(s+3)$ you need to include a zero place holder when defining the polynomial as illustrated in the following example.**

```
Qs = [1, 3, 0, 0]

Qs =
     1     3     0     0

roots(Qs)

ans =
     0
     0
    -3
```

**For this latter case we used commas to enter the row vector, blanks or commas work the same for row vectors. If you use a semi-colon the data would be entered as a column vector.**

**Using the concept of representing polynomials as row vectors we may now utilize specific commands in the Control Systems Toolbox to generate objects which represent transfer functions (tf), state space models (ss) or zero pole gain models (zpk). These objects allow us to perform operations very easily.**

**Consider the transfer function**

$$G2(s) = \frac{s}{s^3 + 6s^2 + 11s + 6}$$

**First define the numerator and denominator polynomials**

```
num = [1 0]

num = 1     0

den = [ 1 6 11 6]

den =  1     6    11     66
```

**Then find the transfer function G2s**

```
G2s = tf(num, den)

 Transfer function:
          s
----------------------
s^3 + 6 s^2 + 11 s + 6
```

**A more succinct notation would be**

```
G2s = tf([1 0], [1 6 11 6])

Transfer function:
          s
----------------------
s^3 + 6 s^2 + 11 s + 6
```

**G2s is a transfer function object.  This object encapsulates information that allows Matlab to perform operations seamlessly.  For instance you can add tf objects and extract data from them.  This will be discussed in Sections II and III.**

**Consider another transfer function which is in factored form**

$$G(s) = \frac{23(s+5)}{(s+1)(s+10)(s+15)}$$

**One could multiply this out the denominator (see conv() command) and use the tf object but sinceG(s)  is in factored form we can use the zpk object.  Here we enter the zeros, poles as row vectors.  The numerator is entered as [-5]m denominator is entered as [-1, -10 -15] and  the gain is entered as a scalar 23 with no brackets.**

```
Gs = zpk([-5], [-1, -10, -15], 23)

 Zero/pole/gain:
     23 (s+5)
-------------------
(s+1) (s+10) (s+15)
```

**Once we have either a tf, ss, or zpk representation Matlab will allow us to convert to others representations by simply using the existing object as input to one of the functions.**

```
Gs_tf = tf(Gs)

 Transfer function:
        23 s + 115
--------------------------
s^3 + 26 s^2 + 175 s + 150


Gs_ss = ss(Gs)
 a =
          x1   x2   x3
   x1     -1    4    0
   x2      0  -10   16
   x3      0    0  -15
b =
            u1
   x1        0
   x2        0
   x3    2.398
c =
             x1       x2       x3
   y1    0.5995   0.5995        0

d =
         u1
   y1     0

Continuous-time model.
```

**At this point in the tutorial you should know how to enter transfer functions in at least two different forms (tf, and zpk) and convert between the different forms (tf, ss, zpk). Use Matlab's "help command" or "lookfor command" to find out more about the functions discussed so far or that will be introduced later.**

**There is a short cut that can be used to enter transfer functions. To do this we first define:**

```
 s = tf('s')
Transfer function: s
```

**We may now directly enter a transfer function using the standard mathematical notation for multiply, divide and precedence to define the function as shown below.**

```
TF1 = 5/(s^2 + 5*s +6)

 Transfer function:
      5
-------------
s^2 + 5 s + 6
```

**Since "s" was defined as a tf object, whenever you define a transfer function you get the results in the tf form (i. e., polynomials multiplied out). If data is entered in factored form**

```
TF2 = 5/((s+2)*(s+3))

 Transfer function:
      5
-------------
s^2 + 5 s + 6
```

**The result is displayed and stored as a tf object. If "s" had been defined as a zpk object then the factored zpk form would be generated and displayed.**

## Section II Operations Using Transfer Function Objects

**One can perform operations on transfer function objects using mathematical operators such +, -, * and / as well as precedence operators. Consider two simple transfer functions:**

```
T1 =  1/(s+5)

 Transfer function:
  1
-----
s + 5

T2 = 1/(s+10)
```

```
 Transfer function:
  1
------
s + 10
```

**The sum of two transfer functions (representing a parallel connection) is:**

```
T3 = T1+T2

 Transfer function:
   2 s + 15
---------------
s^2 + 15 s + 50
```

**The product of the two transfer functions (representing a cascade connection) is:**

```
T4 = T1*T2

 Transfer function:
       1
---------------
s^2 + 15 s + 50
```

**A feedback connection can be obtained using the following function**

```
XX = feedback(T2, T1)

 Transfer function:
     s + 5
---------------
s^2 + 15 s + 51
```

**Here T2 is the system transfer function "G" and T1 is the transfer function associated with the feedback loop, "H" and negative feedback is assumed. If you want positive feedback negate H. If H is a scalar such as 10 you can enter it directly.**

## Section III Getting Data from Objects

**In our last example XX represents a closed loop transfer function.  We would like to obtain the values of  the poles and zeros.  To do this use the functions:**

```
pole(XX)

ans =
   -9.7913
   -5.2087

zero(XX)

ans =-5
```

**If desired, one can equate these to variable names so that you can perform some operation with them, for instance**

```
pp = pole(XX)

pp =
   -9.7913
   -5.2087
```

**Note that pp was returned as a column vector.**

**We can also get data from the object representing the transfer function. Consider again the object XX. Executing the function get(XX) shows the property names and values associated with an object**

```
get(XX)

           num: {[0 1 5]}
           den: {[1 15 51]}
      Variable: 's'
            Ts: 0
       ioDelay: 0
    InputDelay: 0
   OutputDelay: 0
     InputName: {''}
    OutputName: {''}
    InputGroup: {0x2 cell}
   OutputGroup: {0x2 cell}
         Notes: {}
      UserData: []
```

**So now if we wanted the denominator coefficients we could easily obtain them by referencing the property name "den". Note that the values of den are enclosed in {}, this represents a cell array structure. To get the data into a variable called num_data use the following syntax.**

```
num_data = XX.num{1}

num_data = 0     1     5
```

**Use the name of the object XX, followed by a "dot", and then the property name, follow that immediately by {1}. The {} reference the first element of the cell array.**

## Section IV Some Tools for Transfer Function Objects

**This is a very brief introduction to some of the tools/functions that you can use to aid in analysis of systems. These are basically static tools, another tutorial will introduce interactive tools.**

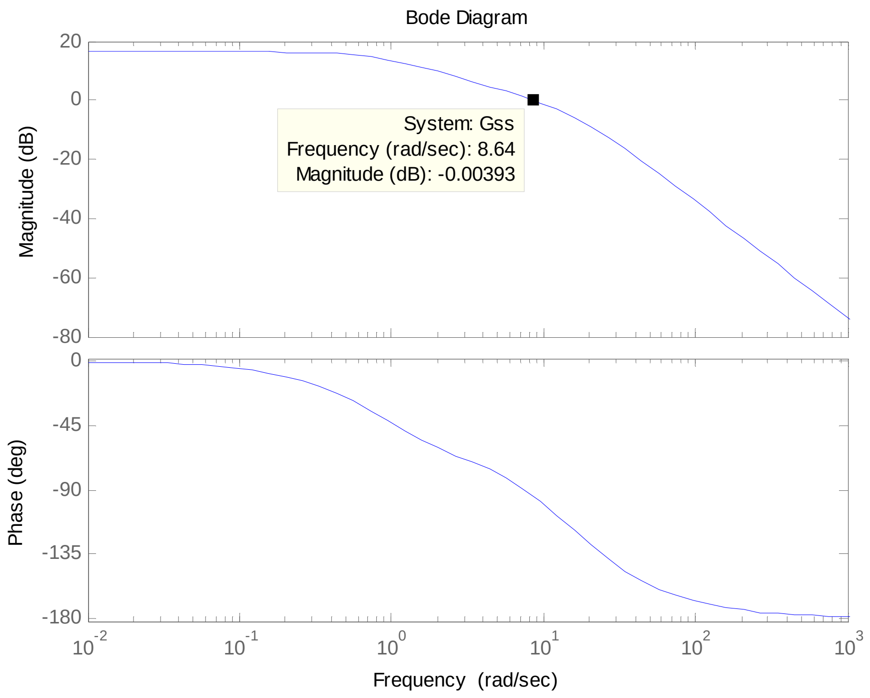
**Starting with the transfer function Gss**

```
Zero/pole/gain:
    200 (s+5)
-------------------
(s+1) (s+10) (s+15)
```

**Executing the following command will find the gain margin, phase margin and corresponding frequencies for the tf object.**

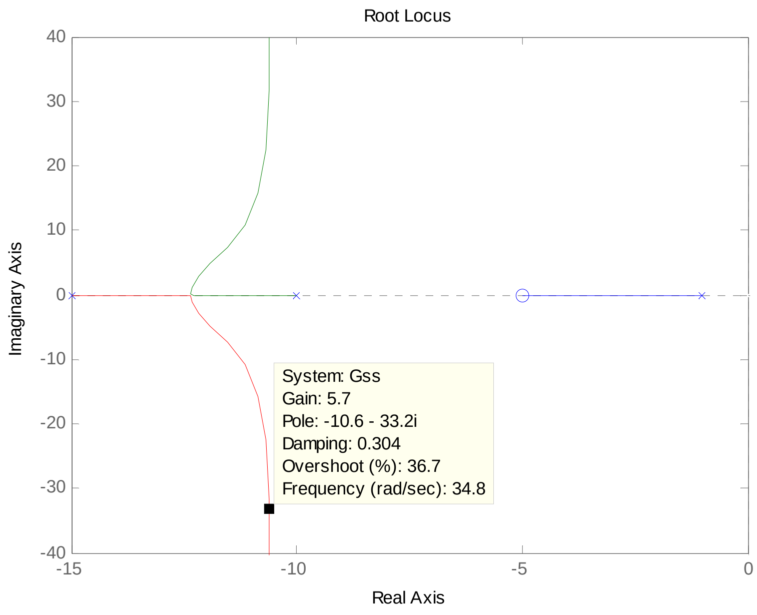
```
 [Gm,Pm,Wcg,Wcp] = margin(Gss)
Gm = Inf
Pm = 85.6459
Wcg = Inf
Wcp = 8.6720
```

**We can find the Bode plot using** `bode(Gss)`

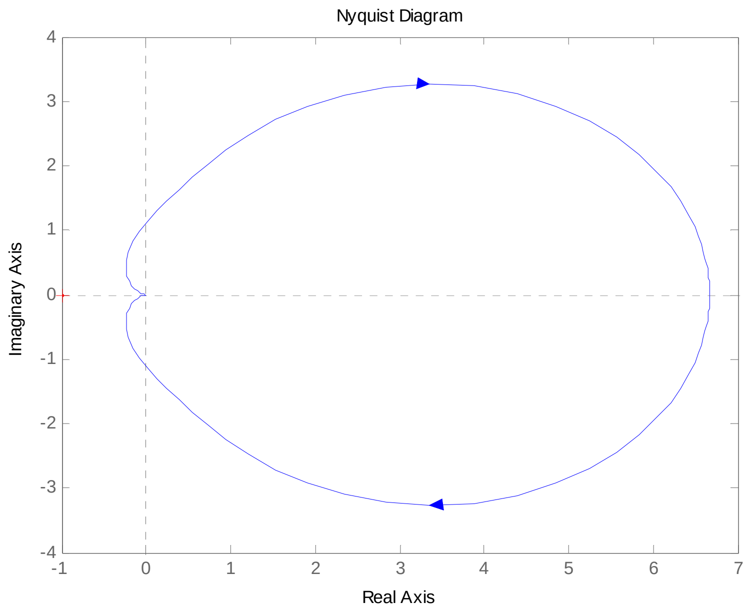


**By placing the cursor over the magnitude plot and holding down the left mouse button you get an active display of the frequency and magnitude ( the same is true for the phase plot). This technique works for all plots that follow.**

**A root locus plot is obtained using  rlocus(Gss)**



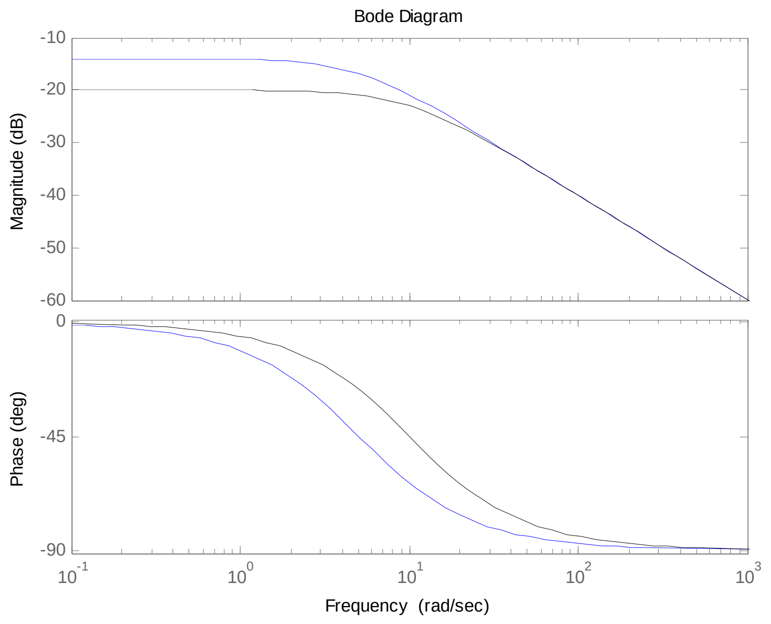
**The command to generate a Nyquist  plot is  nyquist(Gss)**



**As suggested one should use the help function to gain better understanding of these and all functions.  For instance if bode is used with an output,  [MAG,PHASE] = BODE(SYS,W), the plot will not be generated but vectors of the magnitude and angle for the input  frequencies specified in W are obtained.**

**One interesting characteristic of many functions using tf object is to be able to overlay multiple plots with you choice of colors.  By executing bode(T1, 'b', T2, 'k')**

**we obtain both functions on the same plot, the code for the colors may be found by typing help plot at the command line.**



## Section V Partial Fraction Expansion s-domain

**Matlab can also do partial fraction expansion.  However it only works with polynomial input arrays.  So you must either define the numerator and denominator polynomials or extract them from tf objects as previously shown.  Consider T4**

```
T4
 Transfer function:
       1
---------------
s^2 + 15 s + 50
```

**To obtain the partial fraction expansion execute**

```
 [R,P,K] = residue([1], [,1 15, 50])

R =
   -0.2000
    0.2000
P =
   -10
    -5
K =
     []
```

**The  elements of the column vector R  and the column vector P contains the corresponding residue and pole.  K should be zero or empty unless the order of the numerator is greater than the denominator.**