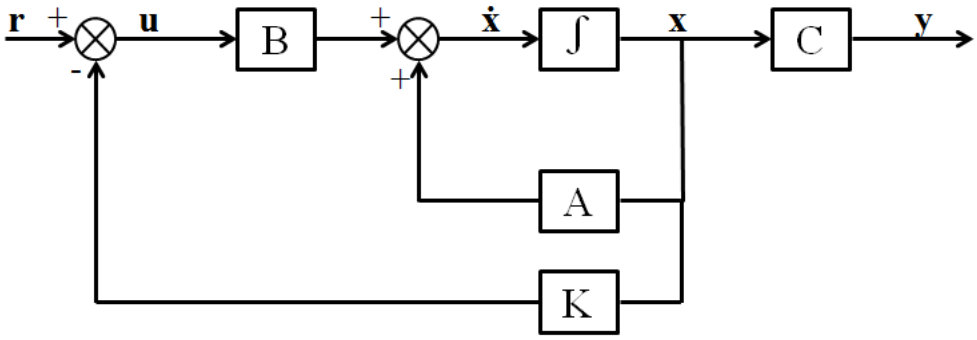
An Introduction to Control Theory Applications with Matlab

Lazaros Moysis Michail Tsiaousis Nikolaos Charalampidis
Maria Eliadou Ioannis Kafetzis

August 31, 2015



An Introduction to Control Theory Applications with Matlab

Lazaros Moysis (moysis.lazaros@hotmail.com)
Michail Tsiaousis (mixalis_1992@hotmail.com)
Maria Eliadou (maria_eliadou@hotmail.com)
Ioannis Kafetzis (ioanniskaf@gmail.com)
Nikolaos Charalampidis (charal.nik@gmail.com)

Editing and coordination: Lazaros Moysis http://users.auth.gr/lazarosm/

Contents

Chapter 0

Preface

During the academic year of 2015, a series of seminars were presented on the Mathematics Department of Aristotle University of Thessaloniki, titled "An introduction to Matlab with Control Theory Applications". These seminars were conducted by PhD student L. Moysis and were part of the undergraduate courses "Classic Control Theory" (7th semester) and "Modern Control Theory" (8th semester), both taught by Prof. N. P. Karampetakis.

The aim of these seminars was to present the programming environment of Matlab, Simulink and the Control Systems Toolbox and cover all the important functions and possibilities that one has to know in order to design and solve a control problem. The syllabus of the seminars was based on [Nise, 2013] and [Ogata, 2009], both of which constitute excellent books on the theory of control systems.

Upon completion of these courses, we decided to gather all the subjects covered into this short, yet thorough book.

This book can serve as a companion manual to all undergraduate and postgraduate students who are taking a course in Control Theory and want to see examples of systems, implemented and solved in Matlab. Problems from Classic and Modern Control Theory are covered, like analysis of 1st and 2nd order systems, root locus techniques, controller design, pole placement, observer design and more. These subjects are engaged through numerous physical applications which we believe the readers will find intriguing.

We hope that the present textbook will be useful to anyone trying to grasp the concept of Control Theory. Since this is the collective work of various people (undergraduate and postgraduate students as well as PhD students) we wish to apologise in advance for any inconsistencies encountered throughout the text. For any suggestions or corrections, feel free to contact us at *moysis.lazaros@hotmail.com*.

The writing team

Chapter 1

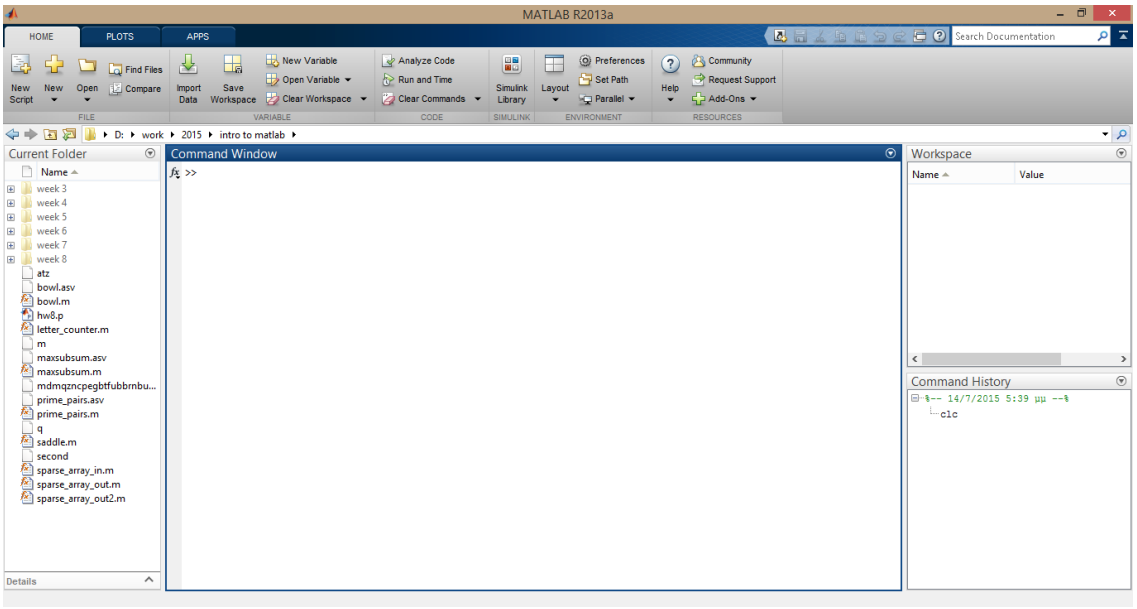
Basic Matlab Commands

1.1 Introduction

Matlab is one of the most powerful tools in computation, numerical analysis and system design. Its user friendly environment, in addition to its powerful computational kernel and graphical visualization capabilities make it an integral part of the control system design, optimization and implementation.

Along with the basic Matlab command package, several additional toolboxes have been developed for specific purposes that extend Matlab's capabilities. Examples are Simulink, Control Systems Toolbox, Fuzzy Logic Toolbox, Image Processing Toolbox, Statistics and Machine Learning Toolbox and many more.

In this chapter, the basic Matlab commands for manipulating and plotting variables shall be presented. Firstly though, the program's interface needs to be explained. The window is divided into three main parts.



The *Command Window* is the main window where the commands are inputed. The *Current Directory* shows the directory from which Matlab runs the files and functions we have created. *Command History* shows the history of all the commands that we input into the Command Window. The main menu has three different tabs, *HOME, PLOTS* and *APPS*. From the Home tab the appearance and layout of Matlab can be changed.

1.2 Input Data

Matlab's basic data structure is the matrix. Of course scalar variables and vector belong to the same category of data. In order to define a new variable in matlab, the following formula is used

```
variable_name= value
% or
variable_name= value;
```

where `value` represents a numerical value. The `;` at the end of the command determines whether the result will be displayed on screen or not. For example, if we define a 1000 column array representing a time interval, there is no point in displaying the result on screen. From the above code it is also seen that we use the symbol `%` to input comments in our code.

There are two more useful methods of inputting data. The first one is the formula to be used when creating matrices, which is

```
M=[a,b; c,d]
```

where the symbols `[]` represent the beginning and end of a matrix and the symbol `;` here is used to denote a new line of the matrix. Needles to say, when defining a matrix variable the number of elements in each row must be the same, otherwise an error message is displayed. To choose among certain elements of a matrix, one can simply use parentheses () to define which element or which parts of the matrix are to be chosen. To do so, inside the parentheses the specific rows and columns are specified. To choose all the columns or all the rows, use the symbol :. This will be made clear in the next example.

The second formula is useful when creating arrays of equally spaced numbers. This is particularly useful when creating time intervals, but is also used inside "for" loops. The command is

```
t=a:step:b
```

where a and b represent numbers and step is the step value used. If the step is omitted, the default value is 1. A second way uses the following command

```
t=linspace(a, b, n)
```

that creates an array of n equally spaced numbers between a and b. As an example, the following commands

```
M=[1 5 10, 6 7 8]
t1=0:2:5
t2=linspace(0,5,4)
M(1,2)
M(:,2)
M(1,[2 3])
```

create three variables with the following values

$$M = \begin{pmatrix} 1 & 5 & 10 \\ 6 & 7 & 8 \end{pmatrix} \qquad t1 = 0 \quad 2 \quad 4 \qquad t2 = 0 \quad 1.6667 \quad 3.3333 \quad 5.0000$$

and the last 2 commands create the arrays

$$5 \qquad \begin{pmatrix} 5 \\ 7 \end{pmatrix} \qquad \begin{pmatrix} 5 & 10 \end{pmatrix}$$

One last useful command is the `clear` command, which deletes any variable in the workspace. Its syntax is the following

```
clear variable_name % deletes variable
clear %deletes all variables
clc % clears the command window (does not delete any variables)
```

1.3 Math Operations and Functions

There are many already available functions that the users can use in order to solve a problem. The typical syntax for a function is to call the function using the desired input arguments. The function will return its outputs, which should be saved in new variables i.e.

```
[output1,output2,. . .]=function_name(input1,input2,. . .)
```

the simplest examples of matlab functions are `cos(x), sin(x), tan(x), log(x), ... exp(x), sqrt(x)` that take as an input a numerical variable and return a unique output. An example of a more complex Matlab function is `ss(a,b,c,d)` that is used to define a new state space system and requires four input arguments. It should be noted here

that for calling a multi-input multi-output function, one uses parentheses for the input arguments and brackets for the multiple variable assignments.

In general, in order to find out about a function's syntax, input and output arguments and general informations about its functionality, the command

```
help function_name
```

displays all the available information on the function, as well as examples and other relevant functions. Try for example `help eig`.

Regarding mathematical operations, Matlab uses the traditional symbols `+ - * ... ^ /`. What should be noted though, is that Matlab treats the operators using linear algebra rules. So if one wishes to use element-wise operations, this should be specified using `.* .^ ./` instead. For example, in order to obtain the square of each element in a vector, the following commands should be used

```
t=0:0.1:10; % Example of a vector
t.^2 % In order to obtain the square of each element
% The command t^2 tries to multiply t*t, which is wrong since it does ...
   not satisfy the linear algebra rules.
```

Some other useful matrix functions are the following

`inv(M)`	Matrix inverse
`pinv(M)`	Pseudoinverse
`eig(M)`	Eigenvalues of a matrix
`eye(r,c)`	Create an $r \times c$ matrix with ones in the diagonal
`zeros(r,c)`	Zero $r \times c$ matrix
`rand(r,c)`	An $r \times c$ matrix of pseudo random values drawn from the standard uniform distribution on the open interval(0,1)
`diag(M)`	Diagonal matrix with the elements M in its diagonal
`m'`	Conjugate transpose of matrix m

1.4 Graphs and Figures

Plotting is one of the most useful applications of any programming language. Matlab offers a variety of plotting tools that help visualize data, both continuous and discrete.

In order to plot a function, three basic steps are required. First define the range of values over which we wish to plot the function, then define the function and lastly, call a plotting command.

Depending on the kind of data we wish to visualize, different plot commands should be used. The most basic are covered in the following table

`plot(x,y,'details')`	Plots vector y versus vector x. In `'details'`, graph details are entered (optional)
`plot3(x,y,z)`	Plots a line in $\mathbb{R}^3$ through the points whose coordinates are (x,y,z)
`surf(x,y,z)`	Plots the coloured parametric surface defined by (x,y,z)
`surfc(x,y,z)`	Plots the coloured parametric surface defined by (x,y,z) in combination with a contour plot
`mesh(x,y,z)`	Plots the coloured parametric mesh defined by (x,y,z)
`meshc(x,y,z)`	Plots the coloured parametric mesh defined by (x,y,z) in combination with a contour plot
`contour(x,y,z)`	Contour plot, i.e. the level curves of function z over the coordinates (x,y)

in the `plot` command, the optional argument `'details'` is a string containing information about the way the data are plotted. The string can contain one character from each of the following columns

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

For example, the commands

```
t=0:0.2:2*pi;
x=cos(t);
plot(t,x,'--')
```

will plot the cosine signal over $[0,2\pi]$ and plot its graph as a dashed line, as will be seen in the following examples.

Other useful commands to change the appearance of a figure are the following

`xlabel('text')`	Text on the x-axis
`ylabel('text')`	Text on the y-axis
`title('text')`	Title text
`grid`	Puts grid on the graph
`axis on`	Displays the axes
`axis off`	Hides the axes
`axis square`	Makes the figure square
`axis equal`	Makes the unit of measure equal to both axes
`axis([ xmin xmax ymin ymax])`	Sets axes limits
`hold on`	Holds the current plot
`legend('first', 'second',. . .)`	Inserts legend

Now, what happens if the user desired to plot different functions on the same figure, while keeping the previous ones too. That is useful for example when we want to observe the changes in the response of the system under perturbations of a single parameter. By default, the plot command will plot a new figure in the same window, deleting the old ones. In order to keep the previous plots, one can use the command `hold all`. For example, using

```
t=0:0.2:2*pi;
x=cos(t);
y=5*cos(t);
plot(t,x)
hold all
plot(t,y)
```

will draw these 2 functions on the same graph, using different colours for them.

On the other hand, what if we want to plot new data on different windows? Matlab offers two options for this. We can either create a new blank figure with a different key number, or create a subplot.

In the first option, simply by inputting the command `figure(n)` Matlab creates a new blank figure under the key number $n = 1, 2, ...$ all plot commands following this command will be visualised in this new window.

On the other hand, the command `subplot(n,m,i)` creates a figure and splits it in n rows and m columns. Every new plot command will be visualised in the subfigure at position i, counting from right to left and top to bottom. To move to another subfigure j we enter the command `subplot(n,m,j)`. So for example:

```
t=0:0.2:2*pi;
x=cos(t);
y=sin(t);
subplot(1,2,1)
plot(t,x)
subplot(1,2,2)
```

```
plot(t,y)
```

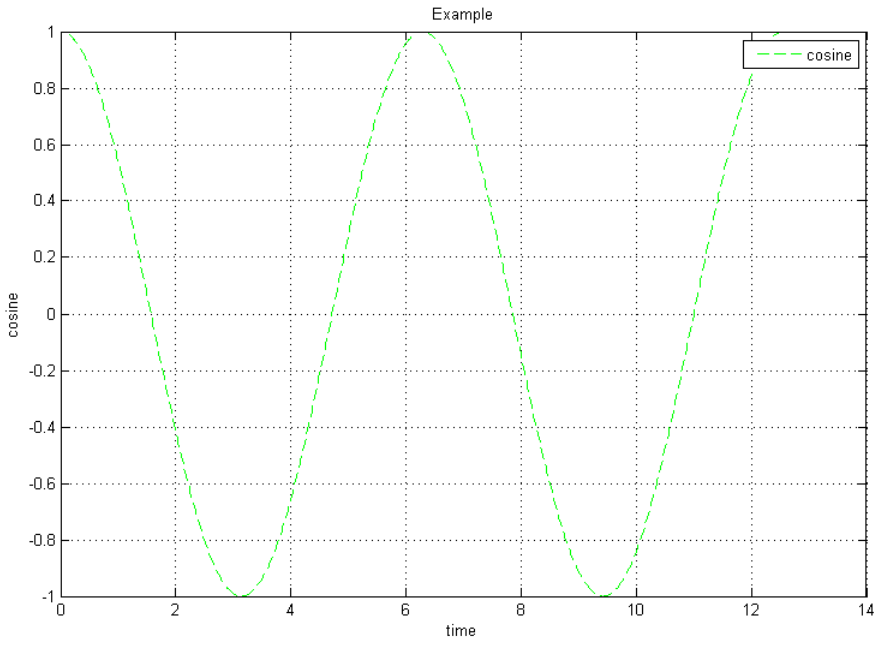
will plot a 2×2 figure having the cosine graph on the first window and the sine graph on the second window. Examples of these commands are given in the next section.

1.5 Examples

Example 1.5.1. Plot the function $x(t) = cos(t)$ for $0 < t < 4\pi$.

Solution.

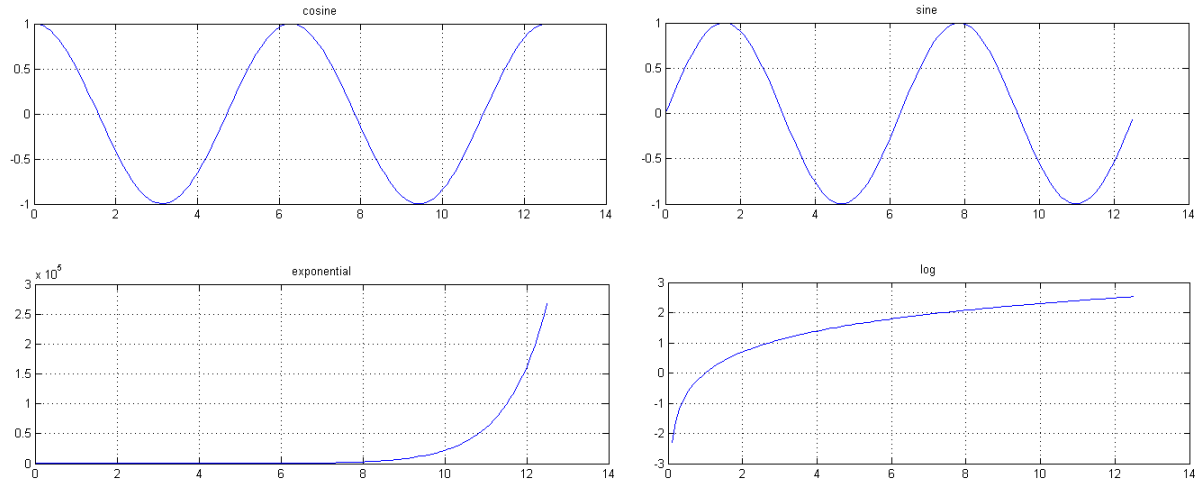
```
t=0:pi/180:4*pi;
x=cos(t);
plot(t,x,'g--')
xlabel('time')
ylabel('cosine')
title('Example')
grid
legend('cosine')
```



Example 1.5.2. Create a 2 by 2 figure with the functions $cos(t)$, $sin(t)$, e^t and $log(t)$ for $0 \leq t < 4\pi$.

Solution.

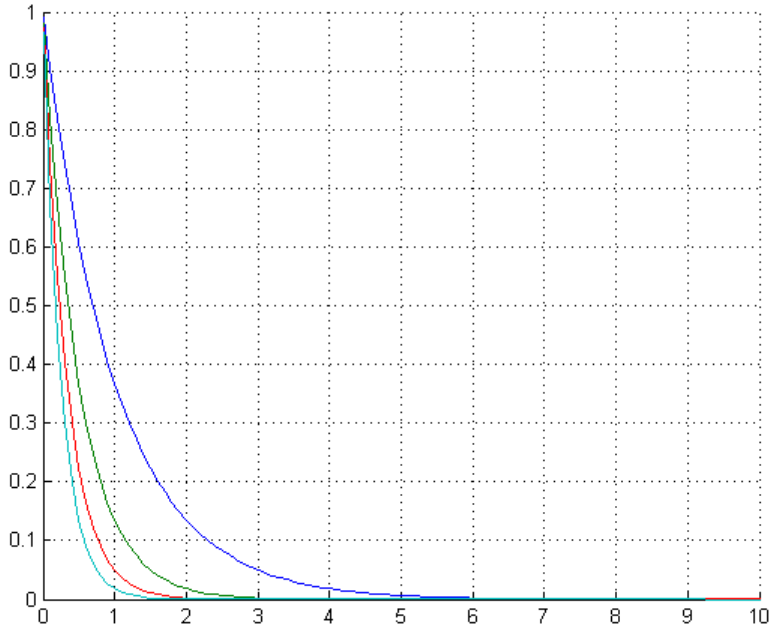
```
t=0:0.1:4*pi;
subplot(2,2,1)
plot(t,cos(t))
title('cosine')
grid
subplot(2,2,2)
plot(t,sin(t))
title('sine')
grid
subplot(2,2,3)
plot(t,exp(t))
title('exponential')
grid
subplot(2,2,4)
plot(t,log(t))
title('log')
grid
```



Example 1.5.3. Plot the function $x(t) = e^{\lambda}t$ for $\lambda = -1, -2, -3, -4$ and $0 \leq t < 10$ on the same graph.

Solution.

```
t=0:0.1:10;
hold all
for i=1:4
    plot(t,exp(-i*t))
end
grid
```

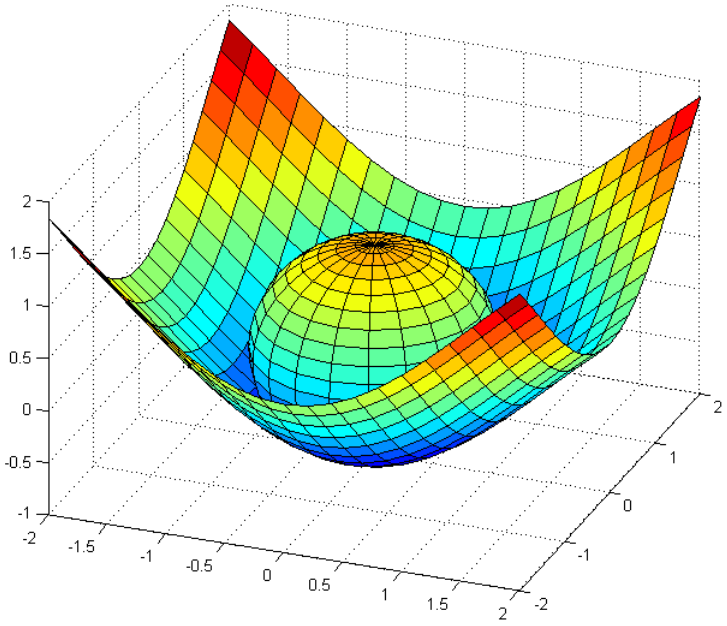


Example 1.5.4. Making use of the commands `sphere` and `meshgrid`, plot the unit sphere and the function $f(x, y) = -cos(x) - cos(y) + 1$ on the same figure.

Solution. We will plot the unit sphere for N=20, and then the surface over the region $-2 \leq x, y \leq 2$.

```
%% For the sphere
[x,y,z]=sphere(20);
surf(x,y,z)
hold all

%% For the surface
[x,y]=meshgrid(-2:0.2:2);
surf(x,y,-cos(x)-cos(y)+1)
grid
```

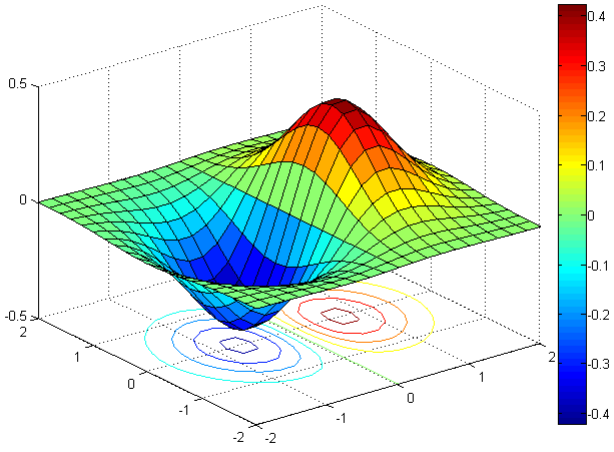
The resulting graph is a sphere "sitting "on the bottom of the surface.



Example 1.5.5. Making use of the command `meshgrid`, plot the surface $f(x, y) = x \cdot e^{-x^2-y^2}$ along with its contour plot.

Solution. We will plot the surface over the region $-2 \leq x, y \leq 2$ using the command `surfc`.

```
[x,y] = meshgrid([-2:.2:2]);
surfc(x,y,x.*exp(-x.^2-y.^2))
```



Chapter 2

Transfer Function Models

2.1 Introduction

In this chapter, we introduce the concept of the transfer function. We will present different ways of creating a transfer function both in polynomial and factored form and show how to convert from one form to the other. To do so, we first present and give examples of basic polynomial functions, since the use of polynomials is required in defining transfer functions.

2.2 Basic polynomial functions

We define polynomials in Matlab using row vectors. Specifically, in order to define a polynomial we create a row vector where the elements of the latter are the coefficients of the polynomial we want to insert.
Generally, if we have the polynomial $p(t) = a_n t^n + a_{n-1} t^{n-1} + \cdots + a_1 t + a_0$ we can define it in Matlab by typing `p=[an . . . a1 a0]` in the command window.

In order to find the polynomial value for a specific value of the variable, we use the command

```
p_value = polyval(p,k)
```

One can also calculate the roots of a polynomial, using

```
r = roots(p)
```

where p is the polynomial.

It is possible to convert from a symbolic polynomial to a polynomial matrix and vice versa. We can achieve this with the commands

```
sym2poly(sym)
```

and

```
poly2sym(vector)
```

respectively. The variable *sym* denotes the symbolic polynomial we use, to convert to a polynomial vector and the variable *vector* denotes the vector polynomial we use, to convert to a symbolic polynomial. A similar command to `sym2poly(sym)` is the command

```
coeff = coeffs(sym)
```

Returns the coefficients of the polynomial *sym* with respect to all variables. The difference between these two commands is that `coeffs(sym)` returns non-zero coefficients starting from the lowest order variable.

2.2.1 Examples

Example 2.2.1. Consider the polynomial $p(t) = t^2 - 2t + 1$

1. Define the polynomial.
2. Calculate the polynomial's roots.
3. Calculate the value of the polynomial for t = 1.

Solution.

```
p = [1 -2 1]; % We insert the polynomial as a row vector
r = roots(p)
p_value = polyval(p,1)
```

Example 2.2.2. Consider the symbolic polynomial $p(t) = t^3 + 2t + 6$.

1. Convert the symbolic polynomial to polynomial matrix.
2. Convert the polynomial matrix back to the symbolic polynomial.

Solution.

```
syms t
sym = t^3+2*t+6;
poly = sym2poly(sym) % Convert symbolic polynomial to polynomial matrix
sym_again = poly2sym(poly) % Convert back to symbolic polynomial
```

Example 2.2.3. Consider the polynomial $p(t) = 8t^6 + 4t^5 - 2t^3 + 7t + 2$.
Extract the coefficients of this polynomial.

Solution.

```
syms t
p = 8*t^6+4*t^5-2*t^3+7*t+2;
coeff = coeffs(p) % Extract coefficients to an array
```

2.3 Analysis of a rational function to partial fractions

Analysing a rational function to partial fractions is important due to the fact that, we have the ability to represent the function (transfer function) in a different way, such as having the poles of the system as the denominators of the partial fractions. There are also practical reasons, such as making it easier to find the inverse laplace transform of a transfer function. A partial fraction expansion of a rational function is the following

$$\frac{b(s)}{a(s)} = \frac{c_1}{s - p_1} + \frac{c_2}{s - p_2} + \cdots + \frac{c_n}{s - p_n} + k_s$$

where $p_i, i = 1, \ldots, n$ the poles of the system, $c_i, i = 1, \ldots, n$ the residues and k_s the quotient We analyse a rational function to partial fractions by using the command

```
[c,p,k] = residue(num,den)
```

Matlab then returns residues c_i, poles p_i, $i = 1, 2, \ldots, n$ and direct quotient k_s in column-wise order. Variable k (k_s), is usually constant or zero. In case k equals zero, Matlab will return [ ]. On the other hand, command

```
[num,den] = residue(c,p,k)
```

returns the numerator and denominator of the rational function that corresponds in this particular analysis.

2.3.1 Examples

Example 2.3.1. Consider the rational function

$$X(s) = \frac{s + 2}{s^3 + 4s^2 + 3s}$$

Analyse the rational function $X(s)$ as partial fractions.

Solution.

```
num=[1 2];
den=[1 4 3 0];
[c,p,k]=residue(num,den)
```

We derive that

$$X(s) = \frac{s+2}{s^3+4s^2+3s} = -\frac{0.1667}{s+3} - \frac{0.5}{s+1} + \frac{0.6667}{s}$$

Example 2.3.2. Find the rational function $X(s)$ that corresponds to the following sum of partial fractions

$$\frac{3}{s-1} + \frac{1.5}{s+4.3} + \frac{-1}{s-2} + 2$$

Solution.

```
c = [3;1.5;-1];
p = [1;-4.3;2];
k = 2;
[num,den] = residue(c,p,k)
```

From the above results, we conclude that

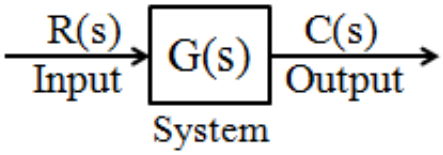
$$\frac{3}{s-1} + \frac{1.5}{s+4.3} + \frac{-1}{s-2} + 2 = \frac{2s^3+6.1s^2-22.7s-1.3}{s^3+1.3s^2-10.9s+8.6} = X(s)$$

2.4 Transfer function

A transfer function is a rational function of a complex variable that represents a linear time invariant dynamical system with zero initial conditions. It describes the relation between the input and the output of the system. Given $r(t)$ to be the input signal with Laplace transform $R(s)$ and $c(t)$ the output signal with Laplace transform $C(S)$, the ratio of the output $C(S)$ to the input $R(s)$ is

$$\frac{C(s)}{R(s)} = G(s) = \frac{b_m s^m + b_{m-1}s^{m-1} + \cdots + b_0}{a_n s^n + a_{n-1}s^{n-1} + \cdots + a_0}$$

We call this ratio $G(s)$ a transfer function. The above equation separates the input $R(s)$, output $C(s)$ and the system $G(s)$, a feature that we are not able to achieve with differential equations. We can represent the transfer function as a block diagram



In order to create a transfer function, one can use the command

```
sys = tf(num,den)
```

It creates a continuous-time transfer function with numerator and denominator specified by *num* and *den*.

Another way of creating a transfer function is by using the zero-pole-gain model, in order to create a transfer function in factored form. The advantage of this model in comparison to the previous one, is that it gives us a straight-forward way of finding the zeros and the poles of our system. A zero-pole-gain model has the following form:

$$H(s) = K\frac{(s-z_1)(s-z_2)\dots(s-z_n)}{(s-p_1)(s-p_2)\dots(s-p_n)}$$

where z_i, $i = 1, \dots, n$ are the zeros of our system, p_i, $i = 1, \dots, n$ are the poles of our system and K is the gain. For a zero-pole-gain model we use

```
sys = zpk(z,p,k)
```

where variable z and p are arrays, containing the zeros and the poles of our system respectively and variable k is the gain, which is a constant. In case we don't have zeros in our transfer function, just input `z = [ ]`.

Additionally, we can convert from one model to the other, i.e from polynomial form to factored form and the opposite, using the same commands. Consider that variable *sys* contains the transfer function created by `tf(num,den)`. In order to convert to a zero-pole-gain model, we can use

```
sys_zpk = zpk(sys)
```

We can also use the following commands

```
[z,p,k] = tf2zp(num,den);
sys = zpk(z,p,k)
```

Command `tf2zp(num,den)` takes as input the numerator and the denominator of the transfer function and returns variables z,p,k containing the zeros, poles and the gain respectively. Then, we simply take the factored form with the command `zpk(z,p,k)`.

On the other hand, let variable *sys* contain a transfer function created by `zpk(z,p,k)`. In order to convert to the tf model (polynomial form) we can use

```
sys_tf = tf(sys)
```

Alternatively we can use,

```
[num,den] = zp2tf(z,p,k);
sys = tf(num,den)
```

Command `zp2tf(z,p,k)` does the exact opposite to `tf2zp(num,den)`. This time, inputs are the zeros, poles and the gain and the result is the numerator and the denominator of the transfer function.

Finally, a third way of creating a transfer function, is by assigning a variable, for example `'s'`, as a variable of the transfer function. Then, simply type the transfer function in the command window. This is a more convenient way, in case we handle subsystems, every one of them with its own transfer function, and the transfer function of the final system is still unknown. In such a case, one can use

```
s = tf('s');
% or
s = zpk('s')
```

in order to assign variable s as a variable of the transfer function.

We can still convert from polynomial form to factored form and the opposite. By using

```
s = tf('s');
```

and then writing the transfer function in factored form, Matlab will return the transfer function in polynomial form. On the other hand, by using

```
s = zpk('s');
```

and then writing the transfer function in polynomial form, Matlab will return the transfer function in factored form.

2.4.1 Examples

Example 2.4.1. Create the following transfer function using `tf(num,den)`

$$H(s) = \frac{s+1}{s^2+3s+1}$$

and then convert from tf model (polynomial form) to zpk model (factored form).

Solution.

```
num = [1 1];
den = [1 3 1];
sys = tf(num,den) % Create transfer function
sys_zpk = zpk(sys) % Convert to zpk model
```

Example 2.4.2. Create the following transfer function using `zpk(z,p,k)`

$$G(s) = \frac{s+2}{(s+1)^2(s+3)}$$

and then convert from *zpk* model to tf model.

Solution.

```
z = -2;
p = [-1 -1 -3];
k = 1;
sys = zpk(z,p,k) % Create zpk model
sys_tf = tf(sys) % Convert to tf model
```

Example 2.4.3. Create the following transfer function by assigning a variable, as a variable of the transfer function.

$$K(s) = \frac{s+1}{s^2+2s+1}$$

Then, convert this transfer function from polynomial form to factored form.

Solution.

```
s = tf('s'); % Assign variable s, as variable of the transfer function
sys = (s+1)/(s^2+2*s+1)

s = zpk('s');
sys = (s+1)/(s^2+2*s+1) % Returns transfer function in factored form
```

Example 2.4.4. Create the following transfer function by assigning a variable, as a variable of the transfer function

$$T(s) = \frac{(s+2)(s-1)}{(s+3)^2(s-5)}$$

Then, convert this transfer function from factored form to polynomial form.

Solution.

```
s = zpk('s'); % Assign variable s, as variable of the transfer function
sys = ((s+2)*(s-1))/((s+3)^2*(s-5))

s = tf('s');
sys = ((s+2)*(s-1))/((s+3)^2*(s-5)) % Returns transfer function in ...
   polynomial form
```

Chapter 3

System Characteristics and Responses

In this chapter methods for calculating the response of a system to different inputs (step, impulse, arbitrary) are presented. In addition, the different characteristics of the transient (rise time, overshoot, settling time) and steady state response (steady state error) of first and second order systems are presented.

3.1 System Characteristics

Some of the most basic characteristics that play an important role in system analysis are of course the poles and the zeros of a system. In order to obtain these, one can use the following commands:

`pole(sys)`	Poles of the transfer function
`zero(sys)`	Zeros of the transfer function
`[w,z,p]=damp(sys)`	Returns the natural frequency and damping factor of each pole in the vector p
`pzmap(sys)`	Pole-Zero map of the transfer function

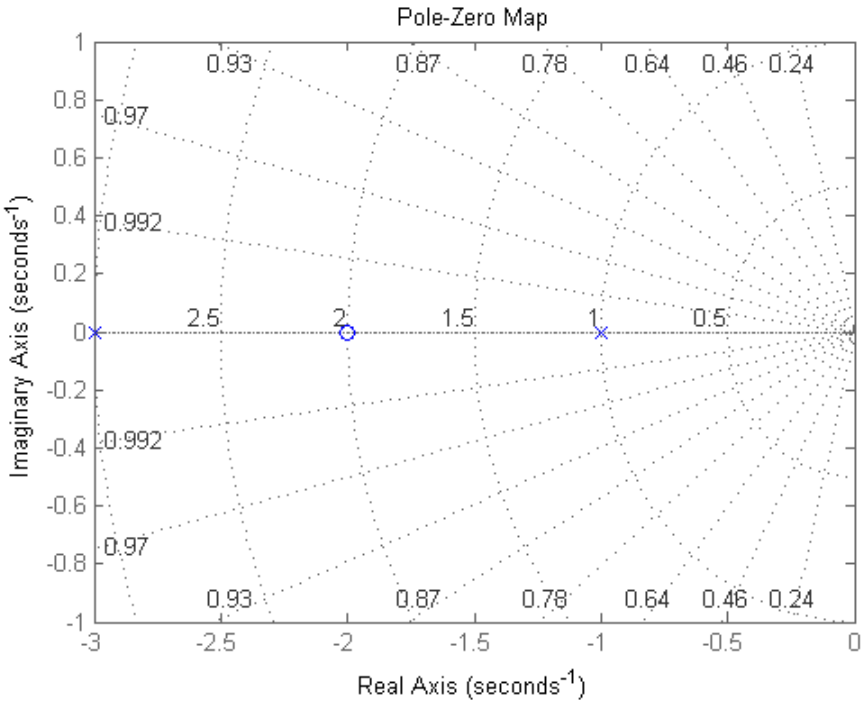
The last two commands are especially interesting, since the `damp` command can give useful information for the frequency and damping factor of the system poles. The `pzmap` although interesting, is usually not used, since `rlocus` has much more interesting features.

Example 3.1.1. Find the poles and zeros of the following transfer function and plot them in the Complex plane.

$$H(s) = \frac{s+2}{(s+1)^2(s+3)}$$

Solution.

```
z=[-2];  % This is the same as z=-2
p=[-1 -1 -3];
k=1;
sys=zpk(z,p,k) % You can also try the command tf(sys)
pole(sys); % in order to obtain the poles
zero(sys); % in order to obtain the zeros
pzmap(sys);
grid
```



3.2 Responses

The signals received by a system are called excitations or inputs of the system and the signals generated by the system because of these excitations are called responses or outputs of the system. Some of the basic responses are the following:

Step Response: It's the dynamic response of the system (assuming zero initial conditions) when the input is the step function $u(t) = 1$, $t \geqslant 0$. In order to obtain the step response of the system, the command `step` is used with its variations

`step(sys)`	Plots the step response of the system sys
`step(sys, Tfinal)`	Plots the step response from $t = 0$ to the final time $t = Tfinal$
`step(sys, t)`	uses the user-supplied time vector t for simulation
`data=stepinfo('sys')`	Computes the step response characteristics
`v=step(sys)`	Saves the response in a vector

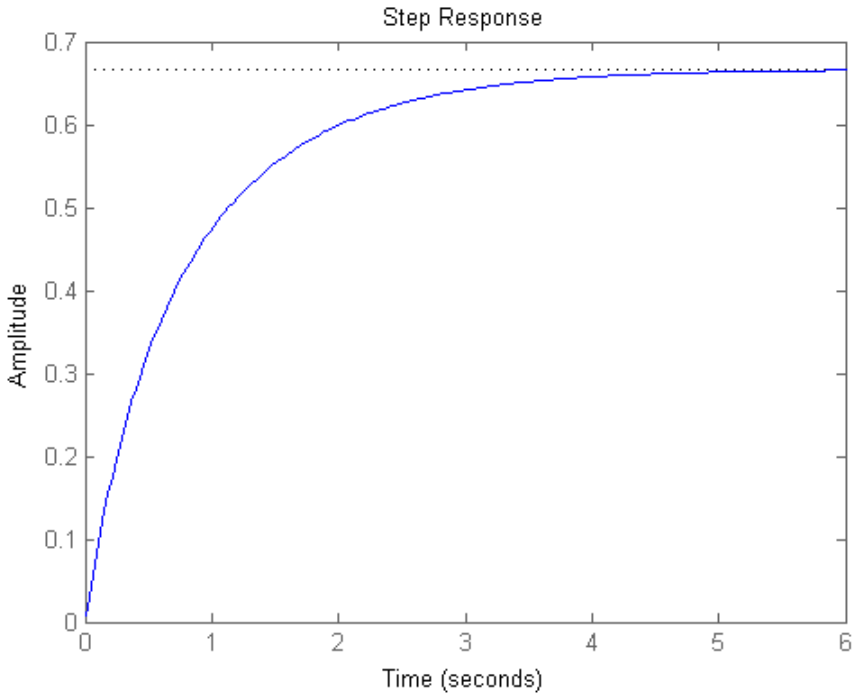
Example 3.2.1. Consider a system with transfer function:

$$G(s) = \frac{s+2}{s^2+4s+3}$$

Find the step response of the system.

Solution.

```
sys=tf([1 2],[1 4 3]);
step(sys)
```



Impulse Response: It's the dynamic response of the system when our input is the impulse function $\delta(t)$. In order to obtain the step response of the system, the command `step` is used with its variations

`impulse(sys)`	Plots the impulse response of the system sys
`impulse(sys, Tfinal)`	Plots the impulse response from $t = 0$ to the final time $t = Tfinal$
`impulse(sys, t)`	uses the user-supplied time vector t for simulation

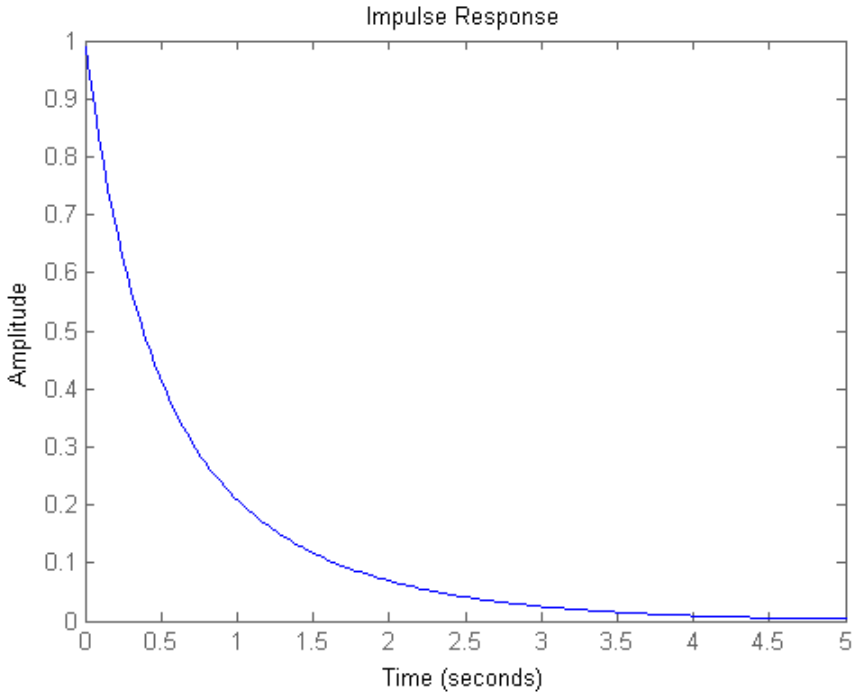
Example 3.2.2. Consider the following transfer function:

$$G(s) = \frac{s+2}{s^2+4s+3}$$

Find the impulse response of the system.

Solution.

```
sys=tf([1 2],[1 4 3]);
impulse(sys)
```



Arbitrary Response: We can compute the response of a system to any desired signal, by creating our own input. In order to do so, we first need to specify the time vector t and the input vector u. After these vectors are defined, the command `lsim(sys,u,t)` along with its variations, simulates the response of the system to the given input.

`lsim(sys,u,t)`	Response of the system to input u
`lsim(sys,u,t,x0)`	Response of the system to input u with initial conditions x_0
`v=lsim(sys,u,t)`	Saves the response in a vector

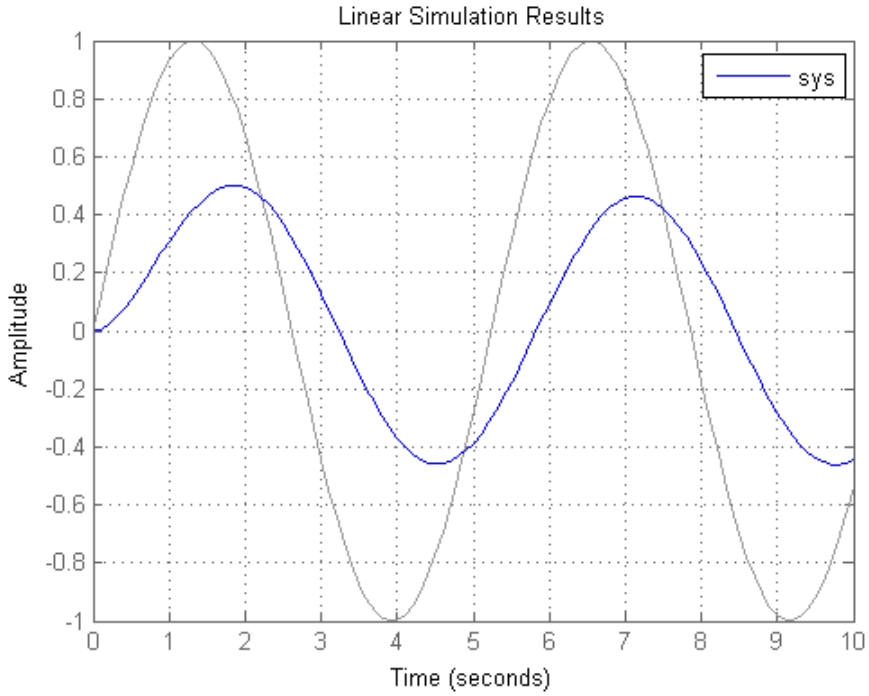
Example 3.2.3. Consider the following transfer function:

$$G(s) = \frac{s+2}{s^2+4s+3}$$

Find the arbitrary response of the system, with $t = 0 : 0.1 : 10$ and $u = sin(1.2 \cdot t)$.

Solution.

```
sys=tf([1 2],[1 4 3]);
% Create time variable
t=0:0.1:10
% Create input
u=sin(1.2*t)
% Plot the response
lsim(sys,u,t)
grid
```



3.3 Examples

Example 3.3.1. Consider the following transfer function

$$G_1(s) = \frac{1}{(s+2)(s+3)}$$

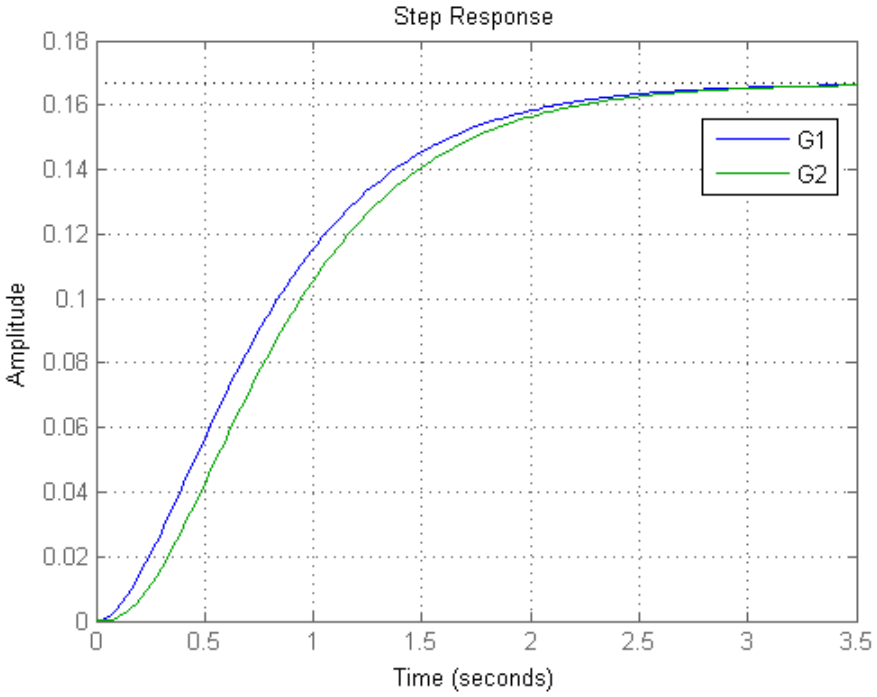
Design the step response of the system.
Then consider a new transfer function

$$G_2(s) = \frac{10}{(s+2)(s+3)(s+10)}$$

and design on the same diagram the step response for the second system.

Solution.

```
z=[];
p=[-2 -3];
k=1;
G1=zpk(z,p,k)
step(G1)
hold on
G2=zpk([],[-2 -3 -10],10)
step(G2)
grid
```



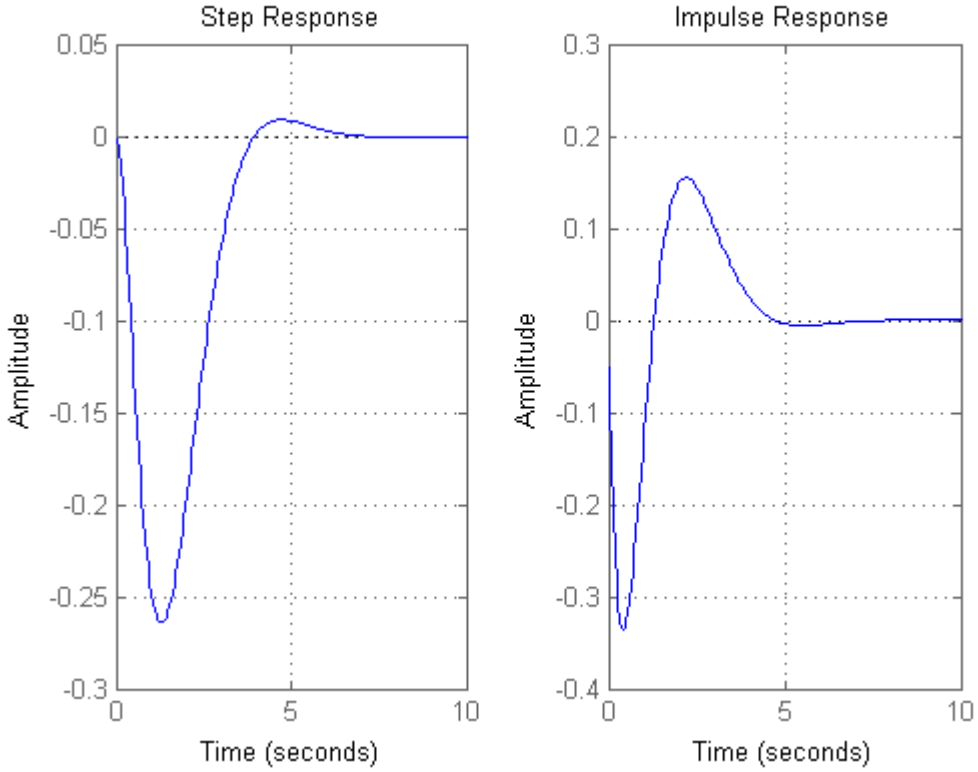
Example 3.3.2. Consider a system with transfer function

$$H(s) = -2\frac{s}{(s+2)(s^2+2s+2)} = \frac{-2s}{s^3+4s^2+6s+4}$$

Define the transfer function to Matlab. Then in a tab with two sub-windows design the impulse and step response of the system.

Solution.

```
Hsys=tf([-2 0],[1 4 6 4])
subplot(1,2,1)
step(Hsys,10)
grid
subplot(1,2,2)
impulse(Hsys,10)
grid
```



Example 3.3.3. Consider a system with the following transfer function:

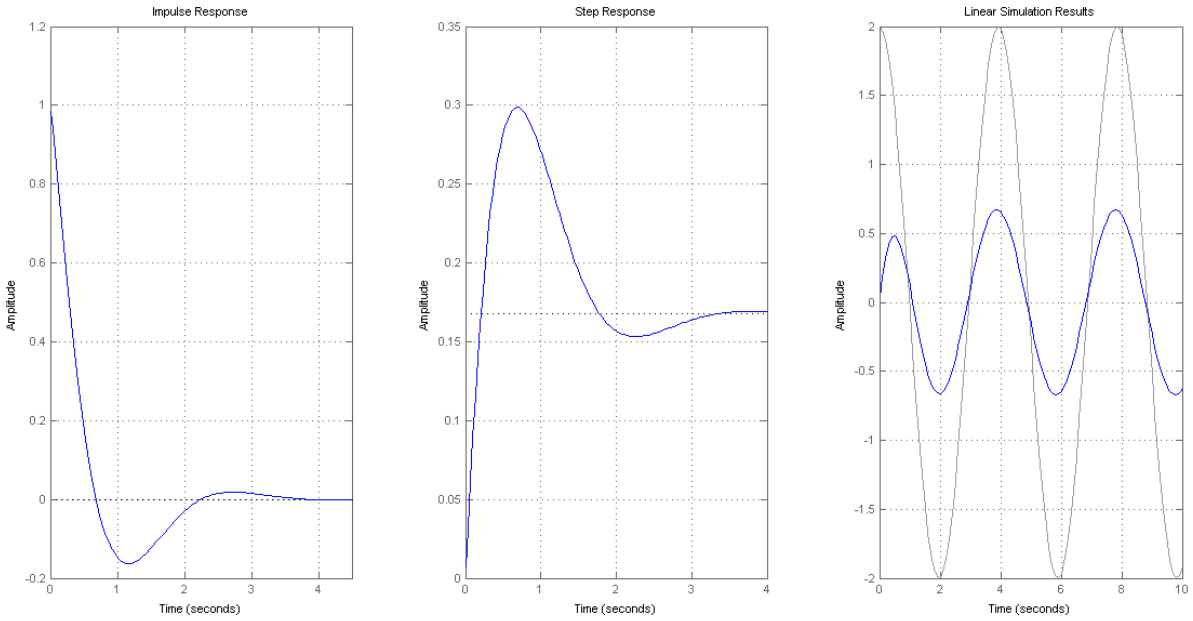
$$G(s) = \frac{s^2+2s+1}{s^3+3.8s^2+8.76s+5.96}$$

Find the poles and zeros of the system and then in a tab with 3 sub-windows plot:

1. The impulse response of the system
2. The step response of the system
3. The response when we have as input the signal $2cos(1.6t)$, in the interval $[0, 10]$.

Solution.

```
% Define transfer function model
sys=tf([1 2 1],[1 3.8 8.76 5.96])
p=pole(sys) % The poles are   -1.4+2i, -1.4-2i, -1
z=zero(sys) % The zeros are -1,-1
% Define time vector and the desired input
t=0:0.1:10;
u=2*cos(1.6*t);
% Begin plotting
% Create a 1-by-3 figure
subplot(1,3,1) %First plot
impulse(sys)
grid
subplot(1,3,2) %Second plot
step(sys)
grid
subplot(1,3,3) %Third plot
lsim(sys,u,t)
grid
```



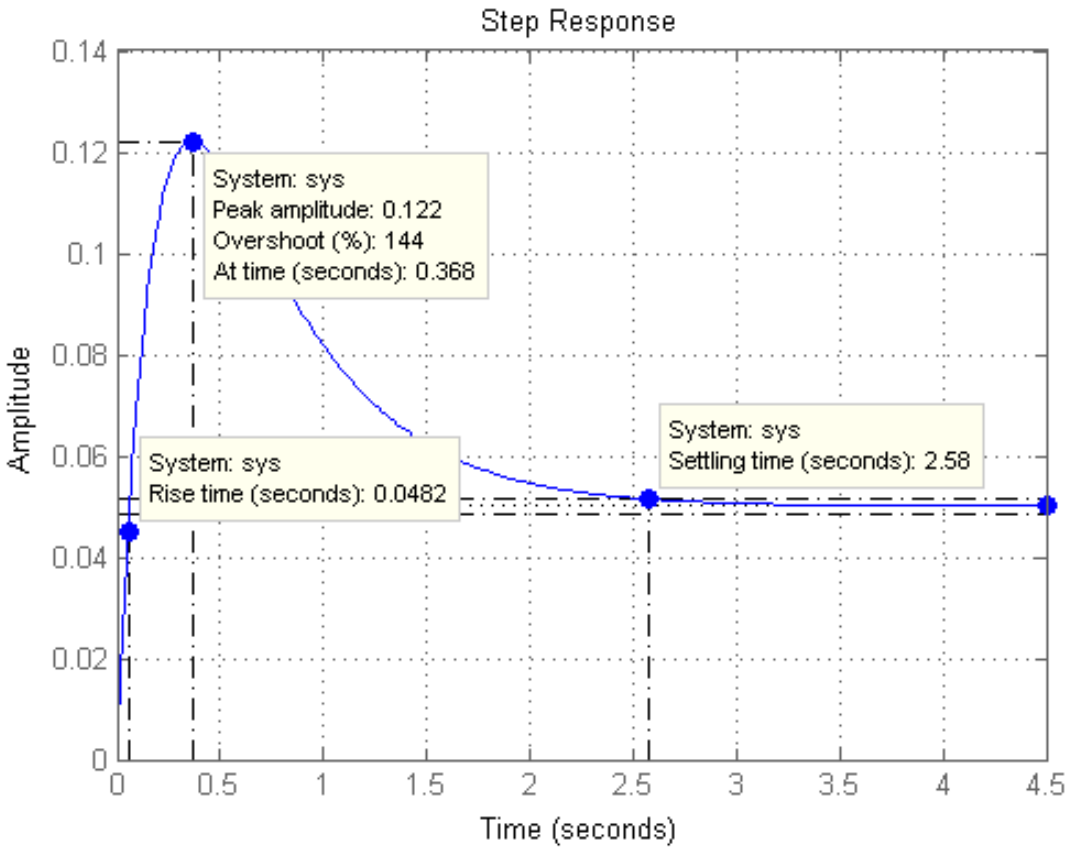
Example 3.3.4. [Nise, 2013, Stefani, 1973] During an experiment, a human sitting in front of a switch reacts to an optical signal by lowering the switch. The transfer function which connects the human response $P(s)$ (output) to the optical stimulus $V(s)$ is

$$G(s) = \frac{P(s)}{V(s)} = \frac{s + 0.5}{(s + 2)(s + 5)} \tag{3.1}$$

What is the step response of the above system and after find the system's features (overshoot, rise time, peak time, settling time)

Solution.

```
z=-0.5;
p=[-2 -5];
k=1;
sys=zpk(z,p,k)
step(sys)
grid
```



Example 3.3.5. [Nise, 2013, Schneider, 1992] An industrial robot is used in the factory to move 55 pounds of salt bags by using a head of compressed air. Such a robot can move up to 12 bags per minute. Consider that the model describing the rotating control head is:

$$G(s) = \frac{\omega_o(s)}{V_i(s)} = \frac{100}{(s+10)(s^2+4s+10)}$$

Where $\omega(s)$ the Laplace transform of the rotational speed of the robot and $V(s)$ the voltage applied to the controller.

1. Can the system be approximated by a second order system?
2. Find rise time, peak time, settling time.

Solution. To find if the system can be approximated by a second order system, we need to find the system poles and study their relations in order to find out if there exist dominant poles. Using the command `roots([1 4 10])` we find that the system has 3 poles, -10 and -2± 2.4495i. It is obvious that the pole at -10 is five times further on the left that the other two complex poles. So, according to [Nise, 2013], an approximation can be made using just the dominant poles.

So we will define a new system that has only these two complex poles. In addition, using the final value theorem, we must find the numerator such that these two systems have the same steady state response.

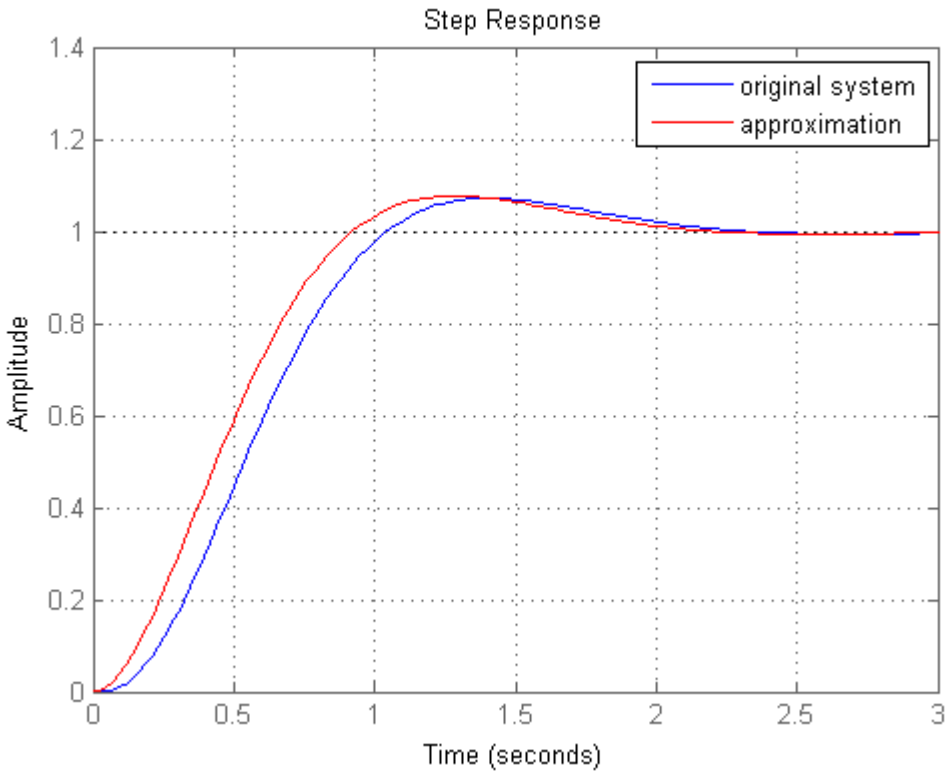
$$e_s = \lim_{s\to 0} sG(s)U(s) = 1$$

$$e_{apr} = \lim_{s\to 0} sG_{apr}(s)U(s) = \lim_{s\to 0} \frac{K}{s^2+4s+10} = 1 \Rightarrow K = 10$$

So overall, the approximate system is

$$G_{apr} = \frac{10}{s^2+4s+10} \tag{3.2}$$

```
roots([1 4 10]) % to find the polynomial roots
sys=zpk([],[-10, -2+2.4495i, -2-2.4495i],100)
sys_app=zpk([],[-2+2.4495i, -2-2.4495i],10)
step(sys)
hold on
step(sys_app,'r')
grid
legend('original system','approximation')
data1=stepinfo(sys)
data2=stepinfo(sys_app)
```



	System	Approximation
RiseTime	0.6492	0.614
SettlingTime	2.0020	1.8951
Overshoot	7.2125	7.6894
Undershoot	0	0
Peak	1.0721	1.0769
PeakTime	1.4046	1.2894

Example 3.3.6. [Nise, 2013, Kuo et al., 2008] A crosslapper is a machine that takes as an input a light fiber fabric and produces a heavier fabric by laying the original fabric in layers rotated by 90 degrees. A feedback system is required in order to maintain consistent product width and thickness by controlling its carriage velocity. The transfer function from servomotor torque, T(s), to carriage velocity, Y(s) , was developed for such a machine. Assume that the transfer function is:

$$G(s) = \frac{Y(s)}{T(s)} = \frac{33s^4 + 202s^3 + 10061s^2 + 24332s + 170704}{s^7 + 8s^6 + 464s^5 + 2411s^4 + 52899s^3 + 167829s^2 + 913599s + 1076555}$$

Find an approximation of the above system using its real pole and plot the step response of the two systems.

Solution. In order to make an approximation of the above system by a first order system, we must first compute its poles. Using `pole` we find that the only real pole is at -1.3839. So the first order system is

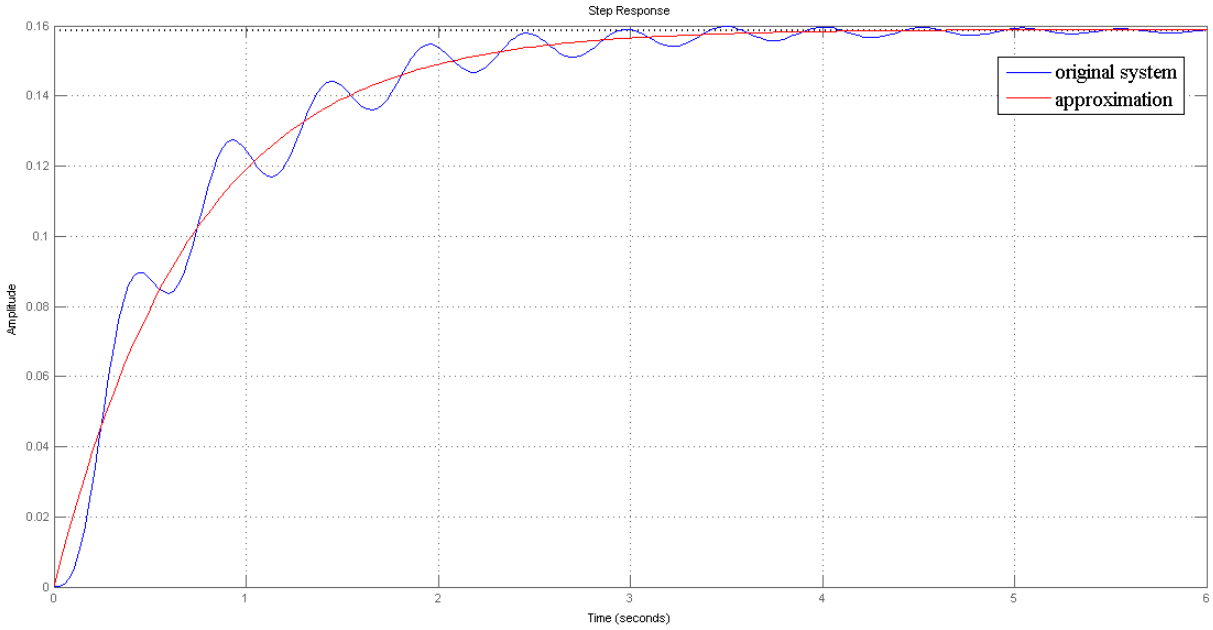
$$G_{apr} = \frac{K}{s + 1.3839}$$

using the final value theorem, we must find the numerator such that these two systems have the same steady state response.

$$e_s = \lim_{s \to 0} sG(s)U(s) = 0.159$$

$$e_{apr} = \lim_{s \to 0} sG_{apr}(s)U(s) = \lim_{s \to 0} \frac{K}{s + 1.3839} = 0.159 \Rightarrow K = 0.22$$

```
num=[33 202 10061 24332 170704];
den=[1 8 464 2411 52899 167829 913599 1076555];
sys= tf( num,den)
[ c,p,k]=residue( num,den)
step(sys) % in order to check the steady-state.
sys_app= zpk([],[ -1.3839],0.159*1.3839)
hold on
step( sys_app,'r')
grid
legend('original system','approximation')
```



Example 3.3.7. [Nise, 2013, Linkens, 1992] Anesthesia induces muscle relaxation (paralysis) and unconsciousness in the patient. Muscle relaxation can be monitored using electromyogram signals from nerves in the hand; unconsciousness can be monitored using the cardiovascular systems mean arterial pressure. The anesthetic drug is a mixture of isoflurane and atracurium. An approximate model relating muscle relaxation to the percent isoflurane in the mixture is

$$G(s) = \frac{P(s)}{U(s)} = \frac{0.0763}{s^2 + 1.15s + 0.28}$$

where P(s) is muscle relaxation measured as a fraction of total paralysis (normalized to unity) and U(s) is the percent mixture of isoflurane.

1. Plot the step response of paralysis if a 2% mixture of isoflurane is used.

2. What percent isoflurane would have to be used for 100% paralysis?

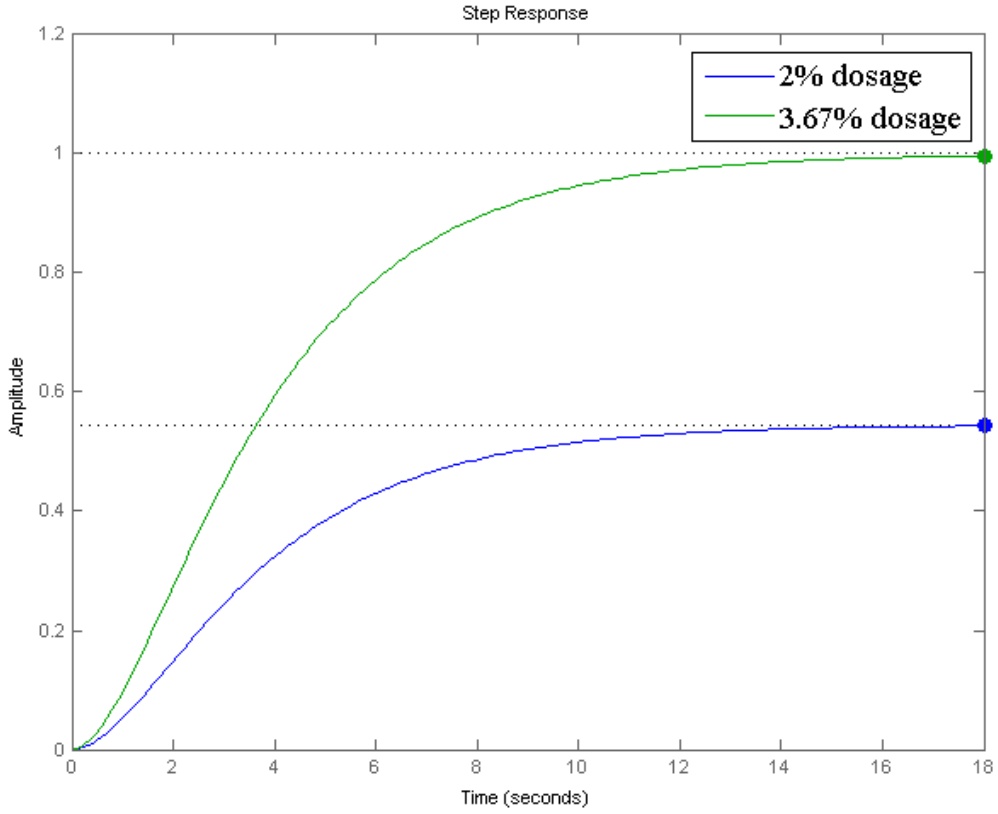
Solution. After we define the system in simulink, we will plot its step response for an input of u=2%, which is equivalent to a step command of the system with an added gain value of 2. For this dosage we observe that the paralysis rises at 54.5%

Now, in order to find the amount of dosage required for complete paralysis, we will use the final value theorem

$$\begin{gathered} p_{final}(t) = \lim_{t\to 0} sG(s)U(s) = 1 \\ \lim_{t\to 0} s\frac{k}{s}\frac{0.0763}{s^2+1.15s+0.28} = 1 \\ \frac{k0.0763}{0.28} = 1 \\ k = 3.67\% \end{gathered}$$

So a dosage of 3.67% is required for full paralysis. Now we will plot the two responses in Matlab and verify the results.

```
sys=tf([0.0763],[1 1.15 0.28])
step(2*sys) % first dosage
hold all
step(3.67*sys) % second dosage
legend('2% dosage','3.67% dosage')
```


Step Response
2% dosage
3.67% dosage
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
0
2
4
6
8
10
12
14
16
18

Chapter 4

Dynamic Behavior of First-Order and Second-Order Systems

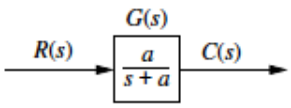
We shall begin by describing some basic features of first-order and second-order systems and then we will give examples of them.

4.1 First-Order and Second-Order System's Features

First-order systems are, by definition, systems whose input-output relationship is a first-order differential equation. A first-order differential equation contains a first-order derivative but no derivative higher than first-order. The order of a differential equation is the order of the highest order derivative present in the equation.

First-order systems contain a single energy storage element. In general, the order of the input-output differential equation will be the same as the number of independent energy storage elements in the system. Independent energy storage cannot be combined with other energy storage elements to form a single equivalent energy storage element.

First-order systems are the simplest dynamic systems to analyse. Some common examples include cruise control systems and RC circuits.



The general form of the first-order's differential equation is as follows

$$\dot{y} + ay = bu$$

The first-order's transfer function is

$$G(s) = \frac{b}{s+a}$$

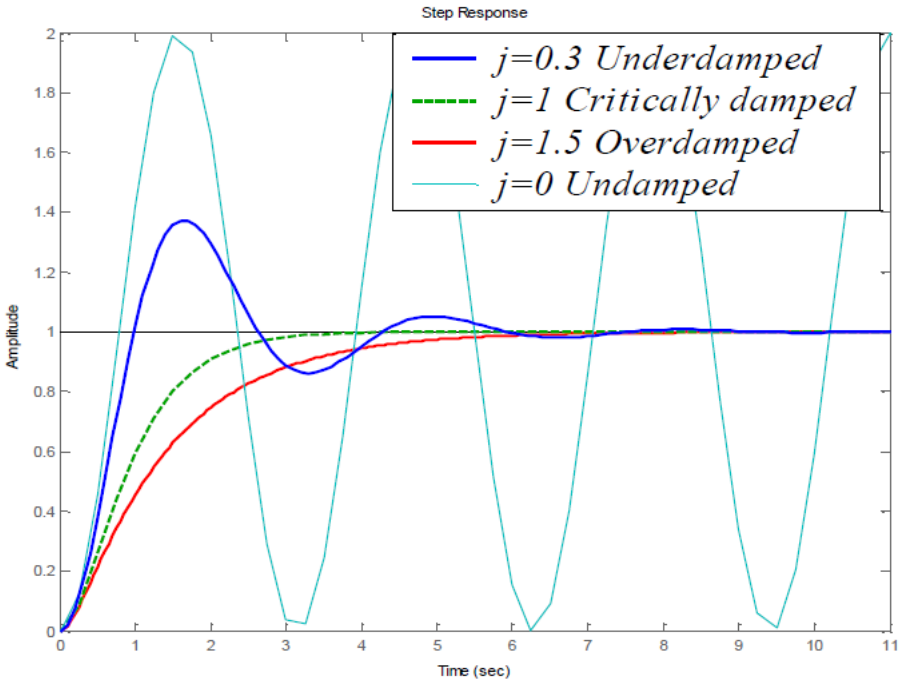
Second-order systems are commonly encountered in practice, and are the simplest type of dynamic system to exhibit oscillations. In fact many real higher order systems are modelled as second-order to facilitate analysis. Typical examples are the mass-spring-damper systems and RLC circuits.

The second-order's transfer function is:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

where ω_n is natural frequency and ζ the damping ratio. Depending on the value of the damping ratio the system exhibits different behavior.

$\zeta = 0$ *Undamping*	The system has two imaginary poles. There is no damping
$0 < \zeta < 1$ *Under-damping*	The system has stable imaginary poles. Quickly tends to equilibrium, but with oscillation
$\zeta > 1$ *Over-damping*	The system has stable real poles. Tends slower to equilibrium, but when it reach, remains in balance
$\zeta = 1$ *Critical damping*	The system has a double real stable pole. Tends to balance the maximum possible time without oscillation.



The qualitative analysis of first and second order systems regards the characteristics of the transient response and the steady state errors. The analysis of these characteristics helps determine the quality of the response regarding specific design requirements. For example, when controlling the flow of a water tank, we want to stabilize the water capacity to a steady level before the tank overflows. This in control theory terms is the minimization of the overshoot. If in addition we desire this transition to a specific water lever to be done in as little time as possible, then we must study the rise and settling time of the system.

As another example, when we design a remote controlled navigation system, it is our aim to input specific coordinates for the system to follow. So in this case we want to minimize the difference between input and output, i.e. the steady state error.

Overall, the basic characteristics that we study are the following:

Overshoot: Describe the difference in the response of the system between the transitional and permanent state, when the system is stimulated by the unit step input. We are interested in the maximum elevation as well as the time it happens.

Rise time: The time required to switch the system's response (in step input)from 10% to 90% of it's final value.

Settling time: The time needed to switch the system's response (in step input) and remain within a certain range of the final price (typically $\pm 2\%$).

Peak time: The time required to reach the first, or maximum peak.

4.2 Examples

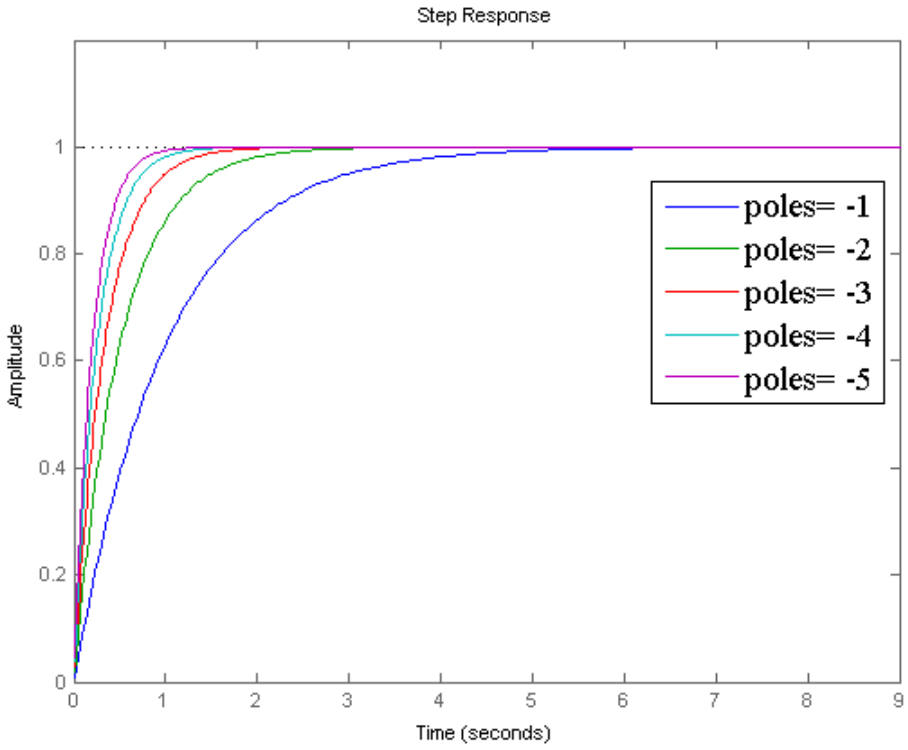
Example 4.2.1. Create a program that will plot the step response of the following transfer function

$$G(s) = \frac{a}{s+a}.$$

for a=1,2,..5

Solution.

```
leg=[];
for i=1:5
    sys=zpk([],[-i],i);
    hold all
    step(sys);
    line=horzcat('poles= -',num2str(i));
    leg=strvcat(leg,line);
end
legend(leg)
```



Example 4.2.2. Create a program that will plot the step response and pole-zero map of the following transfer function

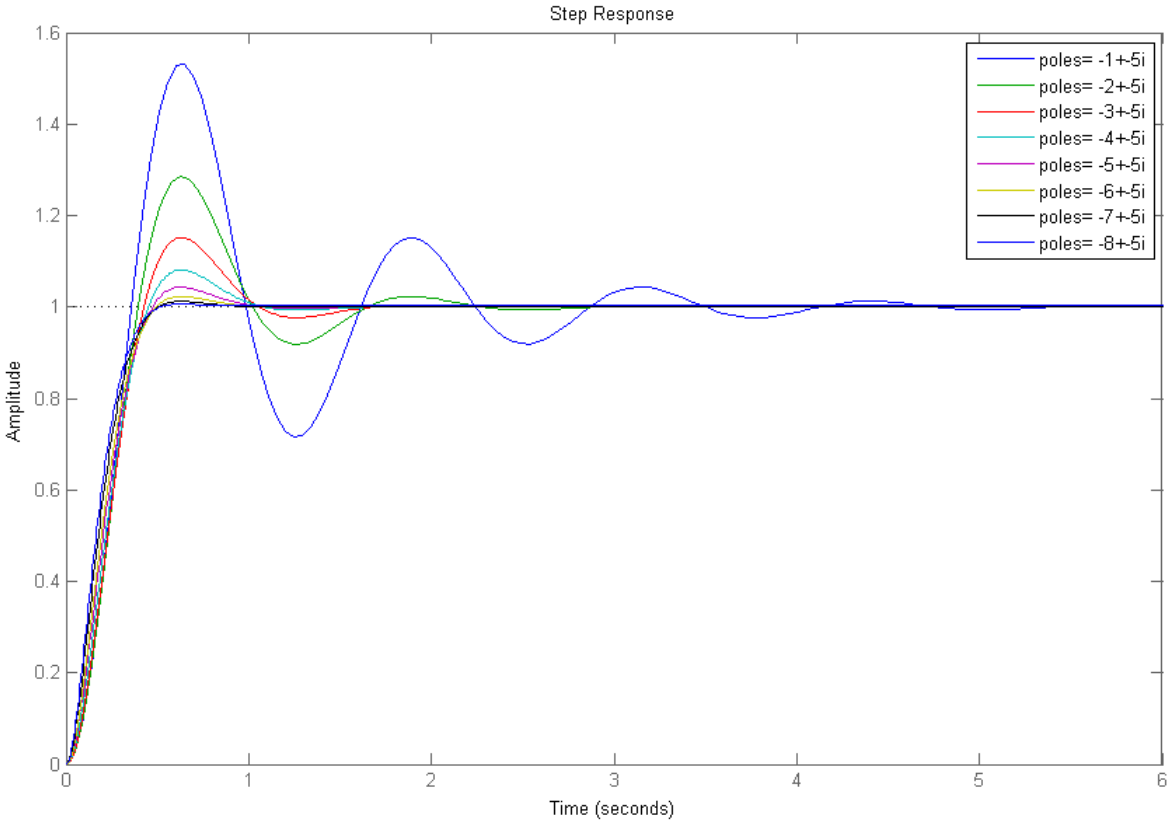
$$G(s) = \frac{|a+5i|^2}{(s+a+5i)(s+a-5i)}$$

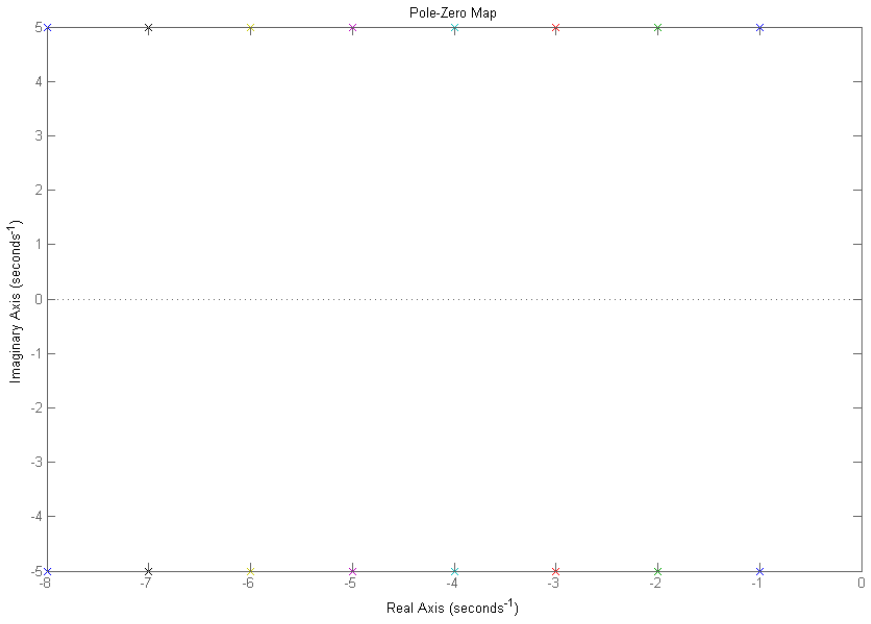
for $a = 1, ..., 8$ (fixed complex part to $\pm 5i$).

Solution.

```
leg=[];
for j=1:8
    sys=zpk([],[-j+5*i, -j-5*i],abs(-j+5*i)^2);
    figure(1)
    hold all
    step(sys)
    line=horzcat('poles= -',num2str(j), '+-5', 'i');
    leg=strvcat(leg,line);
    legend(leg)
    figure(2)
    hold all
    pzmap(sys)
```

```
    pause
end
% Keep pressing enter till you have the desirable result (1-8).
```





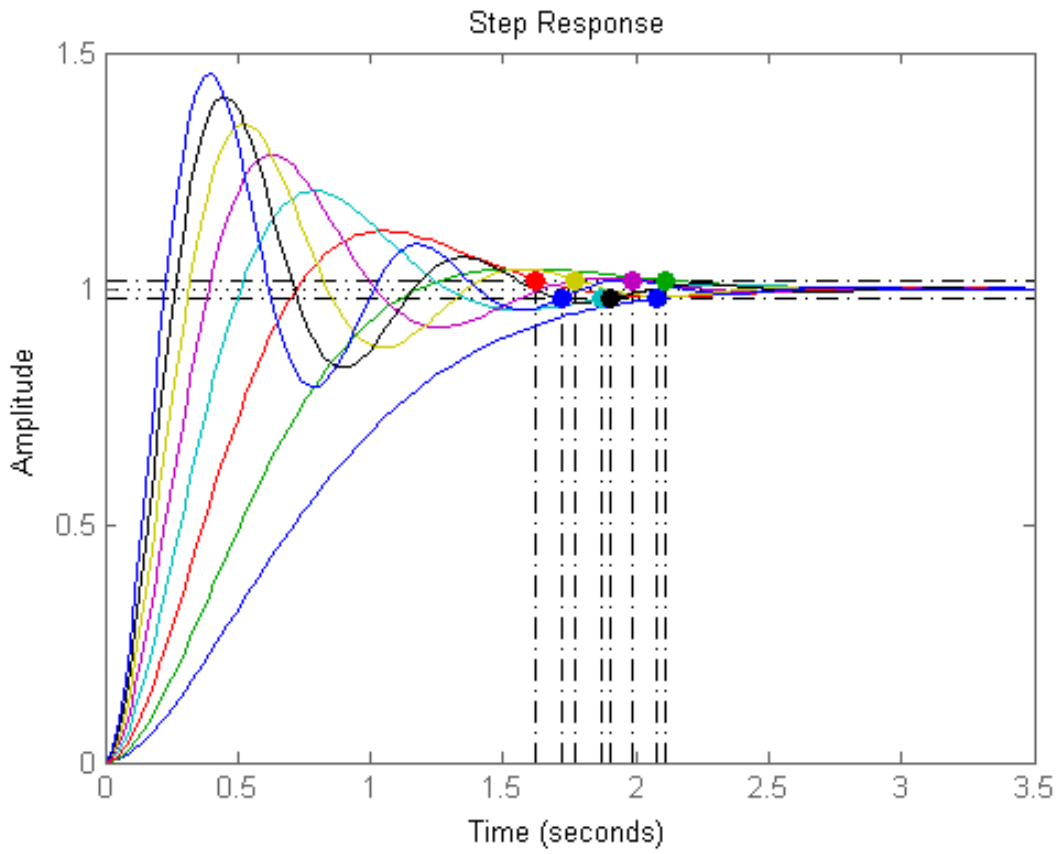
Example 4.2.3. Create a program that will plot the step response and pole-zero map of the following transfer function

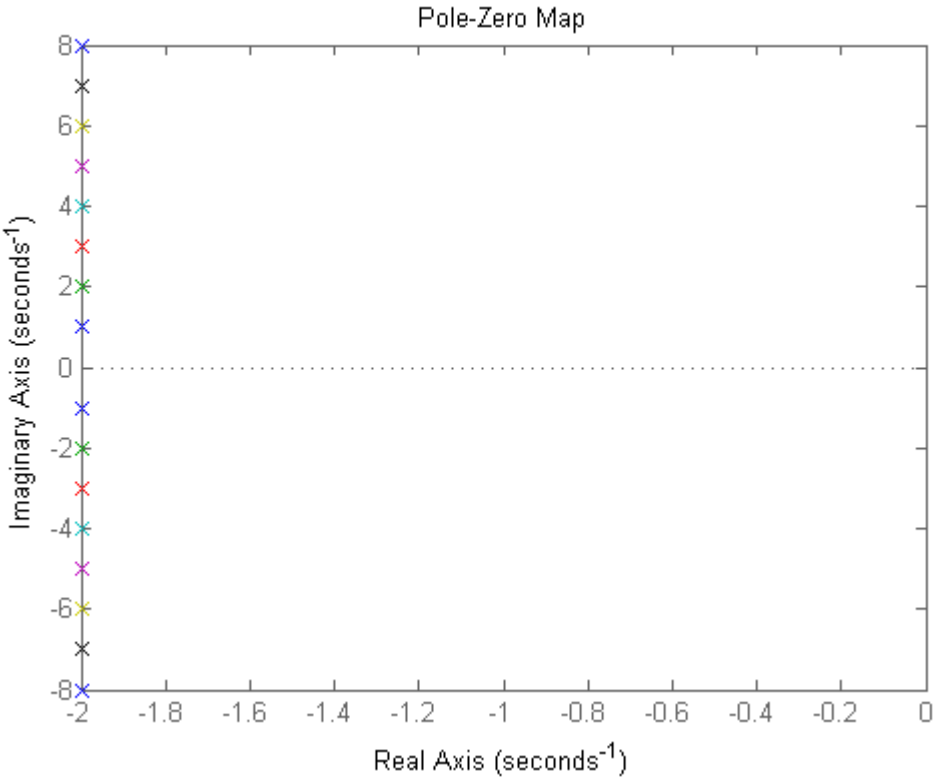
$$G(s) = \frac{|2 + wi|^2}{(s + 2 + wi)(s + 2 - wi)}$$

for $w = 1, ..., 8$ (constant real part to -2).

Solution.

```
for j=1:8
    sys=zpk([],[-2+j*i, -2-j*i],abs(-2+j*i)^2)
    figure(1)
    hold all
    step(sys)
    figure(2)
    hold all
    pzmap(sys)
    pause
end
```





Example 4.2.4. Create a program that will plot the step response and pole-zero map of the following transfer function

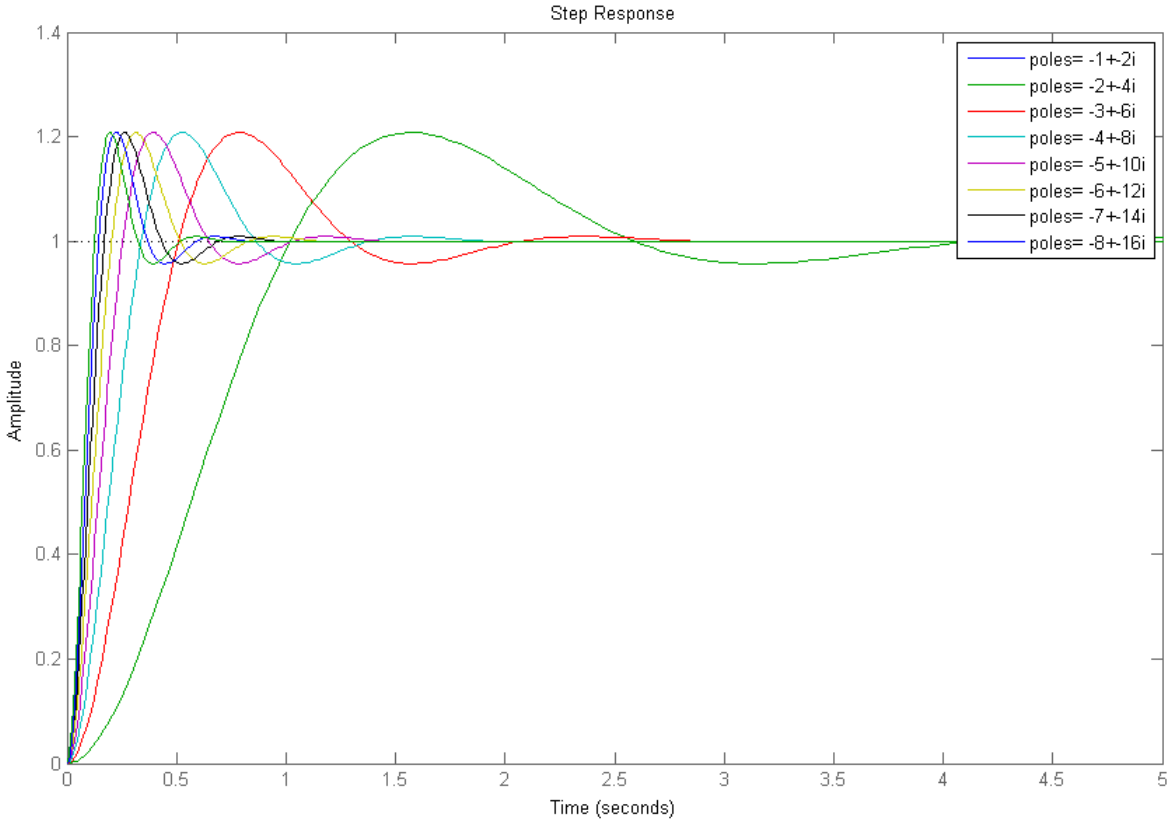
$$G(s) = \frac{|w + 2wi|^2}{(s + w + 2wi)(s + w - 2wi)}$$

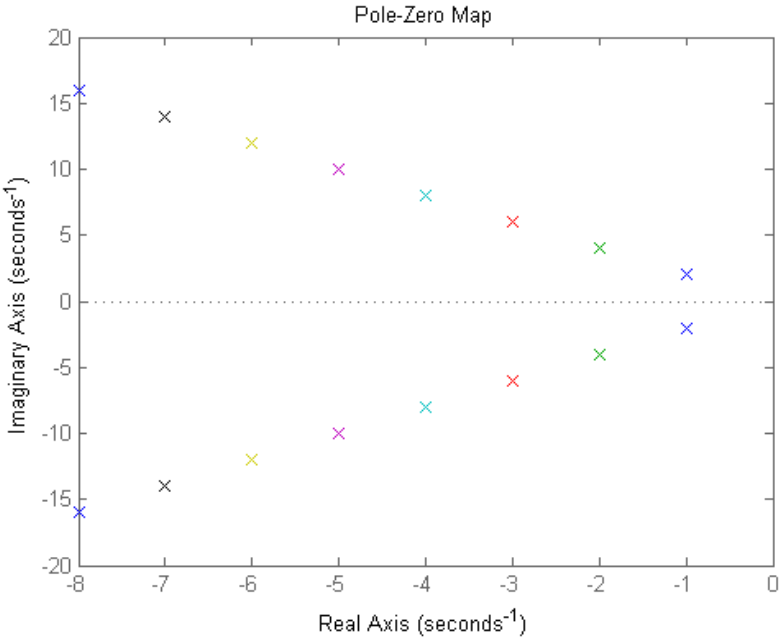
for $w = 1, ..., 8$.

Solution.

```
 leg=[];
 for j=1:8
sys=zpk([],[-j+2*j*i, -j-2*j*i],abs(j+2*j*i)^2);
figure(1)
hold all
step(sys)
% In order to create the legend string:
line=horzcat('poles= -',num2str(j), '+-', num2str(2*j), 'i');
leg=strvcat(leg,line);
legend(leg)
figure(2)
hold all
```

```
pzmap(sys)
pause
end
```





Chapter 5

Systems with Delay

5.1 Introduction

Real dynamical systems often show a time lag between the change of an input and the corresponding change of the output. There is a whole range of reasons that can cause this time lag. Yet for the needs of mathematical modeling, it is aggregated into a total phenomenon called time delay or dead time. Dead time can be defined as the time interval between the instant when the variation of an input variable is produced and the instant when the consequent variation of the output variable starts.

The meaning of delay systems can be understood, considering examples in real life. A faucet with a handle that controls the temperature of the water can be such an example. Even though the handle is immediately turned to achieve the desired temperature the water needs time to get to the desired point. Another example is a man crossing a road, when suddenly sees an incoming vehicle. There is a time interval between the moment the man sees the car and the moment he reacts to avoid getting hit.

5.2 Defining a Delay System in Matlab

There are two basic ways of defining a delay system to matlab, that will be described next along with the corresponding commands.

5.2.1 Defining a Delay system by its transfer function

The first way is to directly insert the transfer function. This can be done using the commands:

```
sys=zpk(z,p,k,'InputDelay',T)
sys=tf(num,den,'InputDelay',T)
```

The system defined by these commands shall be, when the delay is not 0, the transfer function of the system without delay defined by the same zeros, poles and gain or same numerator and denominator when using the first or the second command respectively, multiplied by e^{-Ts}. This is expected considering the following. Suppose a function u(t) is given and the corresponding Laplace transformation is U(s). When u(t) has dead time T is represented as u(t-T) and the corresponding Laplace transformation is $e^{-Ts}U(s)$.

5.2.2 Pade Approximation

The term e^{-Ts} can be approximated using either the Taylor method or the Pade method. Yet the Pade method is more accurate and so it is the most common for this cause.

It is known from basic analysis that the Taylor polynomial of degree n for e^{-Ts} is

$$1 - \frac{sT}{1!} + \frac{(sT)^2}{2!} - \cdots + (-1)^n \frac{(sT)^n}{n!} = \sum_{i=0}^{n} (-1)^i \frac{(sT)^i}{i!}$$

The Taylor approximation uses the fact that $e^{-Ts} = \frac{e^{-\frac{Ts}{2}}}{e^{\frac{Ts}{2}}}$. Using this and the Taylor series of e^{-Ts} one is able to get the approximations. The approximations for degrees 1,2 and 3 are given below

Degree n	Taylor Approximation
1	$\frac{2-Ts}{2+Ts}$
2	$\frac{8-4(Ts)+(Ts)^2}{8+4(Ts)+(Ts)^2}$
3	$\frac{48-24(Ts)+6(Ts)^2-(Ts)^3}{48+24(Ts)+6(Ts)^2+(Ts)^3}$

The Pade approximation uses the fact that

$$e^{-Ts} \approx \sum_{i=0}^{m+n} (-1)^i \frac{(Ts)^i}{i!} = \frac{\sum_{i=0}^{m} p_i (sT)^i}{\sum_{i=0}^{n} q_i (sT)^i}$$

where $p_i = (-1)^i \frac{(m+n-i)!m!}{(m+n)!i!(n-i)!}$ for $i = 0, 1, ..., m$ and $q_i = (-1)^i \frac{(m+n-i)!n!}{(m+n)!i!(m-i)!}$ for $i = 0, 1, ..., n$. When the degree of the numerator and denominator is the same, Pade approximation of e^{-Ts} for degree 1, 2 and 3 is given below

Degree n	Pade Approximation
1	$\frac{2-Ts}{2+Ts}$
2	$\frac{12-6(Ts)+(Ts)^2}{12+6(Ts)+(Ts)^2}$
3	$\frac{120-60(Ts)+12(Ts)^2-(Ts)^3}{120+60(Ts)+12(Ts)^2+(Ts)^3}$

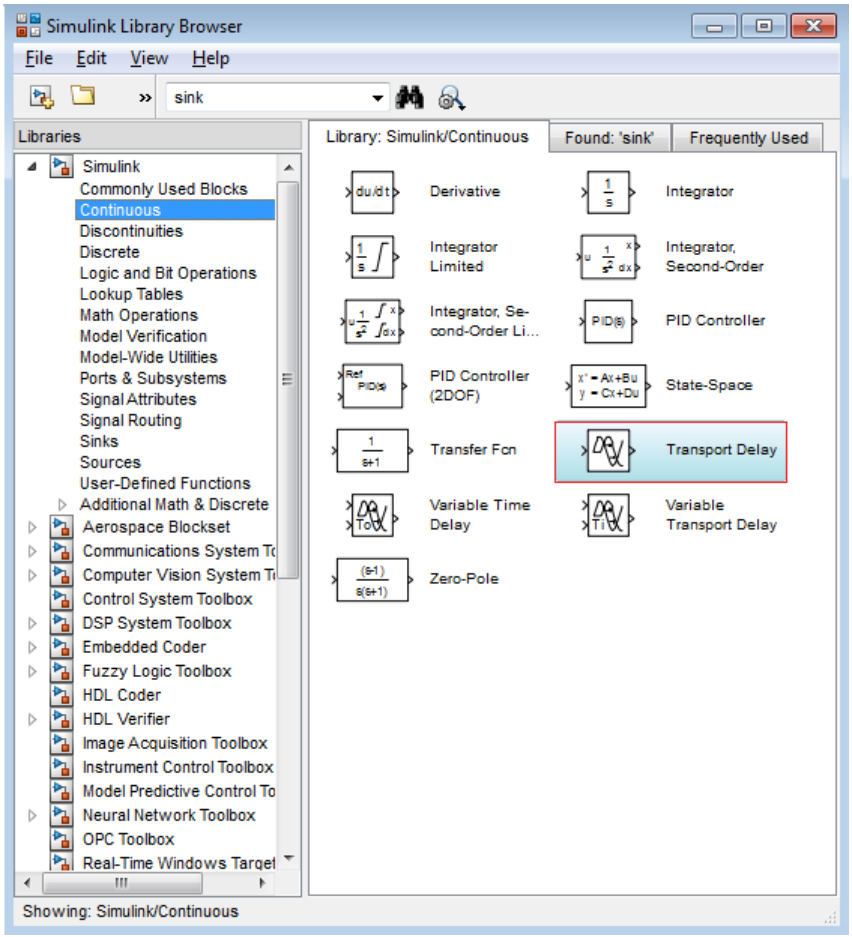
In Matlab, given a defined delay system, named sys, one can use the Pade approximation to define a new system , we will name it *sys_app*, where the exponential factor e^{-Ts}

is now replaced with its approximation, using the command: `sys_app=pade(sys,N)`, where N is the degree of the approximation. At this point it should be commented that this command gives an approximation where the degree of numerator and denominator are both equal to N.

Another really useful command is `[num,den]=pade(T,N)`. The output of this command is a vector with the numerator and denominator of the Pade approximation for e^{-Ts} of degree N.

5.2.3 Delay Systems in Simulink

Delay systems can be defined using Simulink. This can be done using the "Transport Delay" box found in the Simulink Library, in the "Continuous" Section as seen below



Adding this block and double-clicking it opens a window where the parameters regarding the delay can be adjusted. These are the dead time duration, the initial output, since it is not necessary to be always zero, and also the use of Pade approximation, by defining the desirable degree.

5.2.4 Example

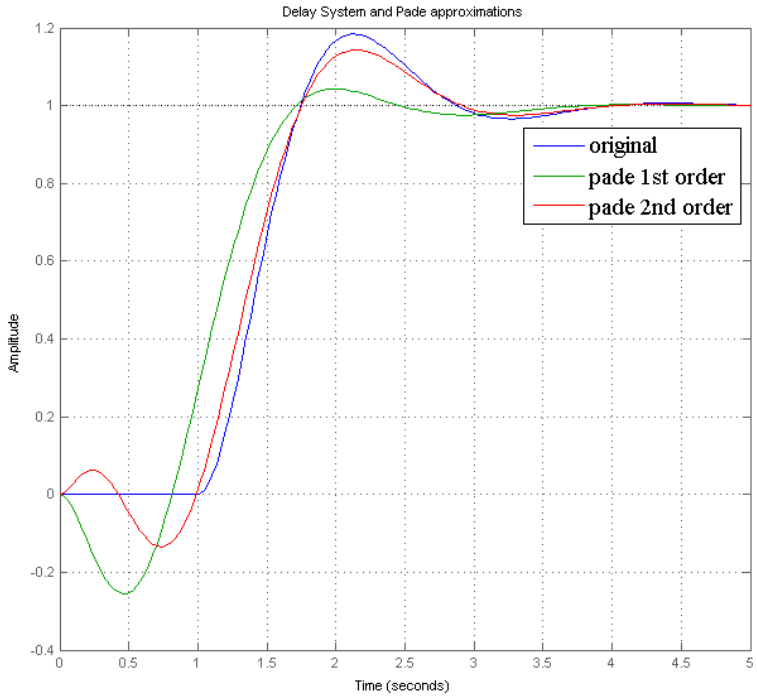
Example 5.2.1. Suppose a system with transfer function $G(s) = \frac{10}{s^2+3s+10}$ is given.

1. Plot the step response for the delay system and its Pade approximations of first and second degree.
2. Find the numerator and denominator of e^{-Ts} for the approximations used.
3. Plot the error for the different Pade approximations.

Solution.

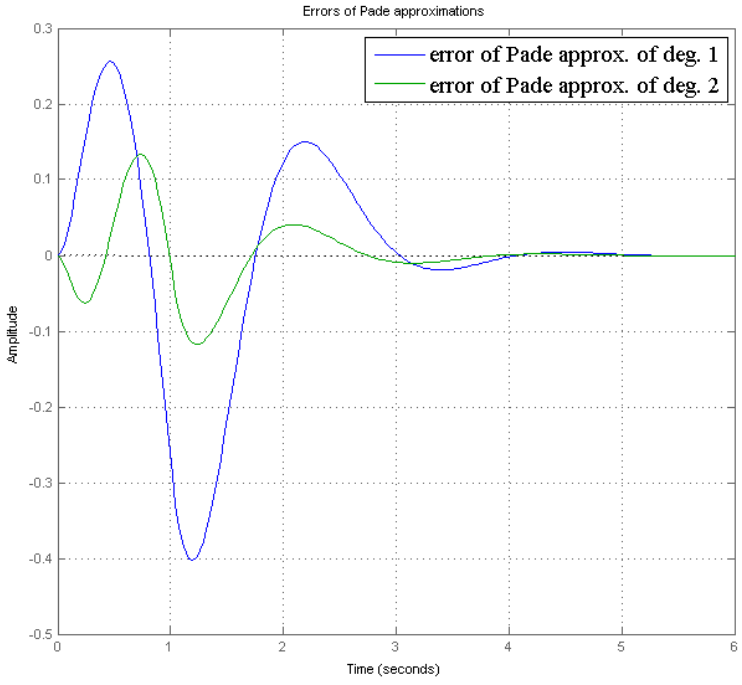
```
%defining the delay system
sys1=tf(10,[1 3 10],'InputDelay',1);
%Getting Pade approximations for degrees 1 and 2
sys2=pade(sys1,1);
sys3=pade(sys1,2);
%Getting the graph of all three approximations
figure;
step(sys1,sys2,sys3)
title('Delay System and Pade approximations')
legend('original','pade 1st order','pade 2nd order')
grid

%To see the terms of the approximation for delay 1 we use
[num1,den1]=pade(1,1);
[num2,den2]=pade(1,2);

%Define e_1 the error of the Pade approximation of degree 1 and
%e_2 the error of the Pade approximation of degree 2
figure(2);
e_1=sys1-sys2;
e_2=sys1-sys3;
step(e_1,e_2)
title('Errors of Pade approximations')
legend('error of Pade approx. of deg. 1','error of Pade approx. of deg. 2')
grid
```

This will produce the graph of the original system and the systems with Pade approximations of degree one and two as shown below



Also, removing the semi-colons from the `[num1,den1]=pade(1,1)` and `[num2,den2]=pade(1,2)` commands will give us the numerator and denominator of the Pade approximations for first and second degree respectively.

Finally the last part creates a second graph with the errors of the Pade approximations of degree one and two.



Chapter 6

System Interconnections

6.1 Subsystem Connection Types

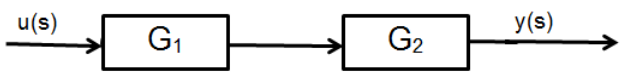
Up until this point we studied simple systems. The next step is to define complex systems since these are the most common in applications. Complex systems consist of simple subsystems connected to each other. There are three different types of subsystem connections:

1. Series
2. Parallel
3. Feedback

Of course a complex system can be defined by subsystems connected in a combination of the above interconnections.

We shall now observe how each connection type defines the transfer function of the complex system. To do so we shall consider two defined systems with transfer functions G_1 and G_2 and input u.

Series Connection: This type of connection is represented by the image below

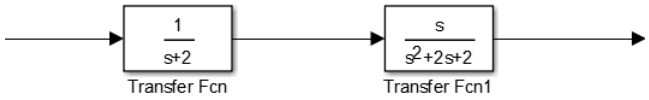


The transfer function of the system defined by this type of connection is $G(s) = G_1(s)G_2(s)$ The result remains the same regardless of the number of the systems connected. The transfer function of a system defined by a series connection of subsystems with transfer functions $G_1, G_2, \cdots, G_n$, $n \in \mathbb{N}$ is $G(s) = G_1(s)G_2(s)\cdots G_n(s)$. The commands used to define a system of this type in Matlab are

```
sys=series(G1,G2,. . . ,Gn)
sys=G1*G2*. . . *Gn
```

Example 6.1.1. Given two systems with transfer functions $G_1(s) = \frac{1}{s+2}$ and $G_2(s) = \frac{s}{s^2+2s+2}$ find the transfer function of the system defined by the two cascaded subsystems.

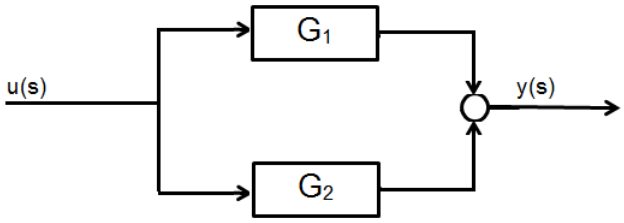
Solution. The system shall look like this




```
%define the systems given
sys1=tf(1,[1 2]);
sys2=tf([1 0],[1 2 2]);
%calculate the trasfer function of the system defined by the cascaded ...
    systems
sys3=series(sys1,sys2)
```

This output is the transfer function needed, which is $G(s) = \frac{s}{s^3+4s^2+6s+4}$.

Parallel Connection: This kind of connection is represented by the image bellow



and the transfer function of a system defined by the parallel connections of two systems is $H(s) = G_1(s) + G_2(s)$.
The commands used to define a system as a parallel connection of two subsystems in Matlab are

```
sys = parallel(G1,G2,. . .,Gn)
sys = G1+G2+. . . +Gn
```

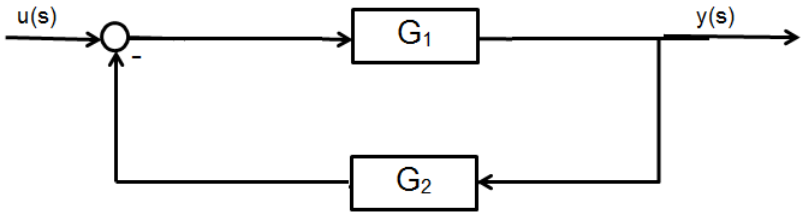
Example 6.1.2. Given two systems with transfer functions $G_1(s) = \frac{1}{s+2}$ and $G_2(s) = \frac{s}{s^2+2s+2}$ find the transfer function of the system defined by the two subsystems with parallel connection.

Solution. The procedure is about the same as in the example of series connection.

```
%define the systems given
sys1=tf(1,[1 2]);
sys2=tf([1 0],[1 2 2]);
%calculate the trasfer function of the system defined
%by the parallel connection of the subsystems
sys3=parallel(sys1,sys2)
```

This output is the transfer function needed, which is $G(s) = \frac{s^2+2s+2}{s^3+4s^2+7s+4}$.

Feedback Connection: Feedback connection is represented by the image below



and the transfer function of a system with feedback is $G(s) = \frac{G_1(s)}{1+G_1(s)G_2(s)}$.
The command to define a system with feedback in Matlab is

```
G = feedback(G_1,G_2)    % for negative feedback
G = feedback(G_1,G_2,1) % for positive feedback.
```

Example 6.1.3. Given two systems with transfer functions $G_1(s) = \frac{1}{s+2}$ and $G_2(s) = \frac{s}{s^2+2s+2}$ find the transfer function of the system with negative feedback.

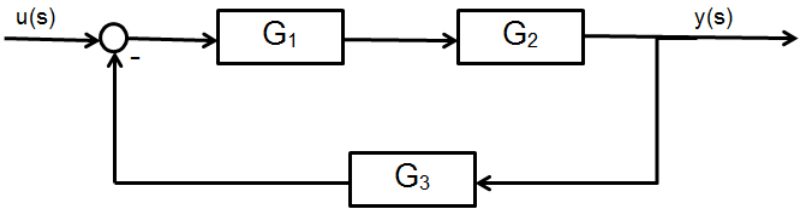
Solution.

```
%define the systems given
sys1=tf(1,[1 2]);
sys2=tf([1 0],[1 2 2]);
%calculate the trasfer function of the system with feedback
sys3=feedback(sys1,sys2)
```

This output of this process is the transfer function needed, which is $G(s) = \frac{s^2+2s+2}{s^3+4s^2+7s+4}$.

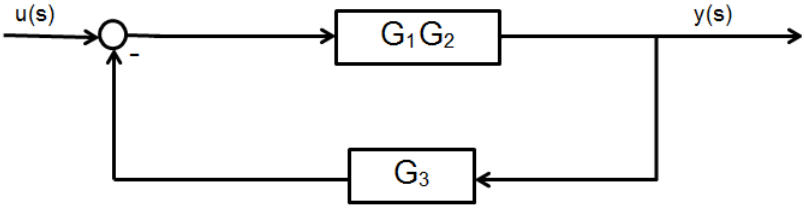
After these basic examples the one that follows is a combination of subsystems connection types mentioned above.

Example 6.1.4. Find the transfer function of the system



given that $G_1(s) = \frac{s+2}{s^2+4s+3}$, $G_2(s) = \frac{1}{s+1}$ and $G_3(s) = \frac{1}{s+2}$.

Solution. According to the picture one can observe that there are three subsystems with transfer functions $G_1,\ G_2,\ G_3$. We shall name these three systems 1,2,3 respectively. Systems 1 and 2 are connected in series. This means that the system given is equivalent to:



where G_1G_2 is the transfer function of the cascaded systems 1 and 2. Next we observe that this new system, namely 12, is connected through feedback to system 3.

After understanding the structure of the model, we shall use Matlab to solve this.

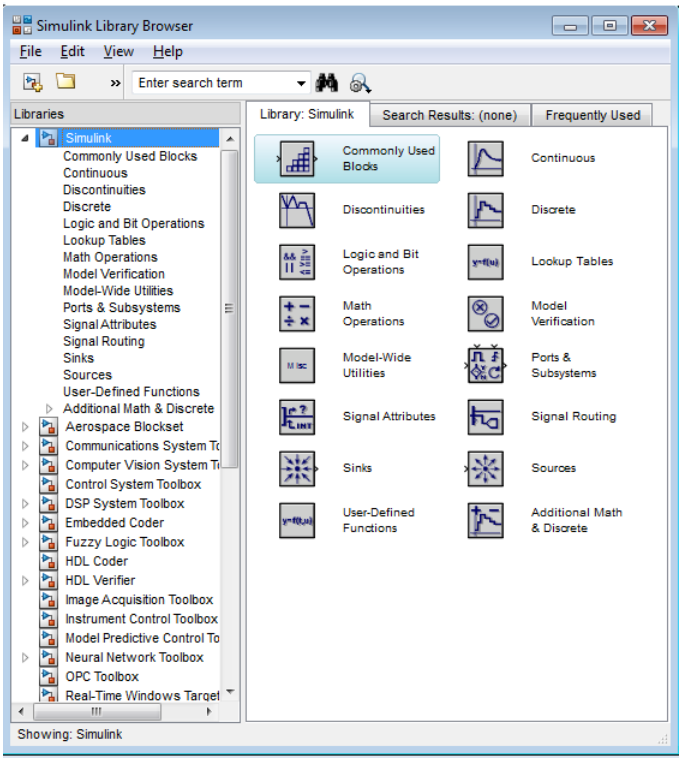
```
%Insert the transfer functions of the systems 1,2 and 3
G1=tf([1 2],[1 4 3]);
G2=tf(1,[1 1]);
G3=tf(1,[1 2]);
%Calculate the t.f. of the 12 system
G12=series(G1,G2);
%Calculate the t.f. of the final system
G_final=feedback(G12,G3)
```

6.2 Simulink

Although it seems easy to calculate the transfer function of a system using its subsystems and the types of connections between them, that is not always the case. This happens because in the methods presented until now the user had to understand the system and the connections that occur. The second thing is that the user should define all the basic systems in Matlab and after this use commands to calculate all the transfer functions

needed. This process is more time consuming and difficult depending on the complexity of the system.

The solution for this problem is Simulink. Simulink is a built-in Matlab tool that gives the user the ability to design the system using blocks, and then calculate its response to different inputs, its transfer function, steady state errors and more. To open Simulink the user can either click the "Simulink Library" button or type Simulink on the command window. This will open a window looking like this

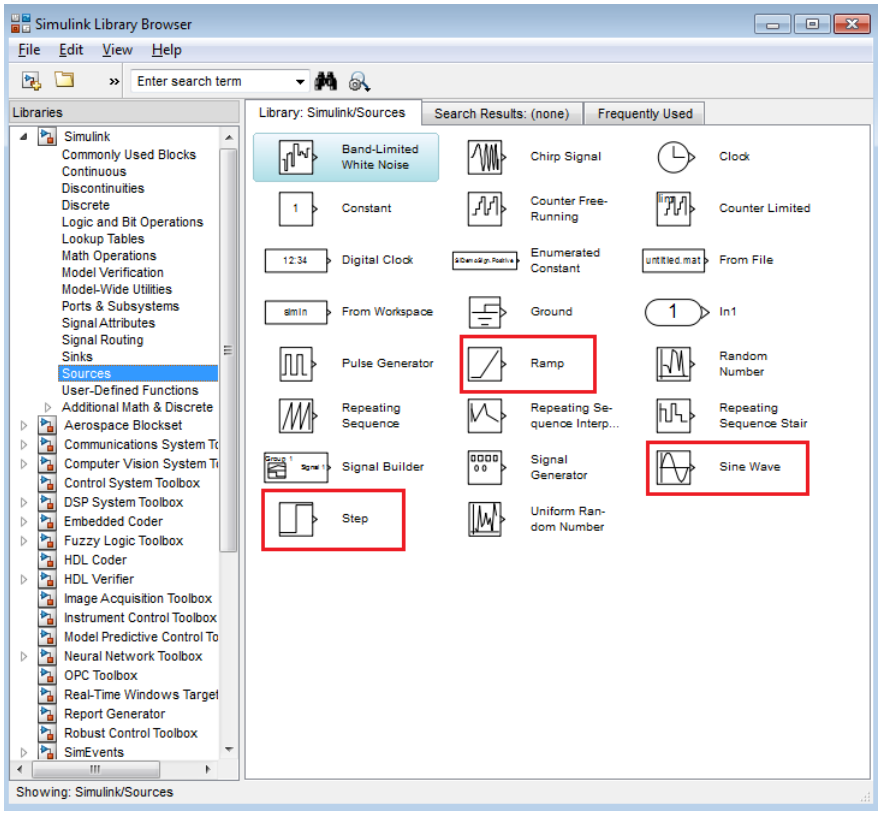


This window is split in two main parts. On the left part the user can see the categories of blocks, in order to find the necessary block faster. The right part consists of all the blocks in the selected category. Clicking "File → New → Model" opens a blank window, which is where the user adds the blocks. To add a block one shall locate it in the library and "drag and drop" it in the Model. After adding a block the user can double click it to change its properties. Each block has input and output ports, presented as "arrow heads". To connect two different blocks the user shall click and hold on the output of the first one. This creates an arrow that can be dragged to the input of the second block.

After the model is designed, the time duration for the simulation is chosen by changing the bar indicated on the tab of the window, which is set at 10 seconds by default. After the duration for the simulation is defined the user shall click the green "Run" button and the model is simulated.

Now, the basic blocks for designing a system in Simulink shall be presented.

Sources: This category consists of the blocks that are used as inputs for the system designed.



We shall focus on the blocks that will be of most use during this presentation and their properties.

"Step" Block: This represents a step input. The properties for this block include the moment when the value of the step function changes, the initial value and the final value.

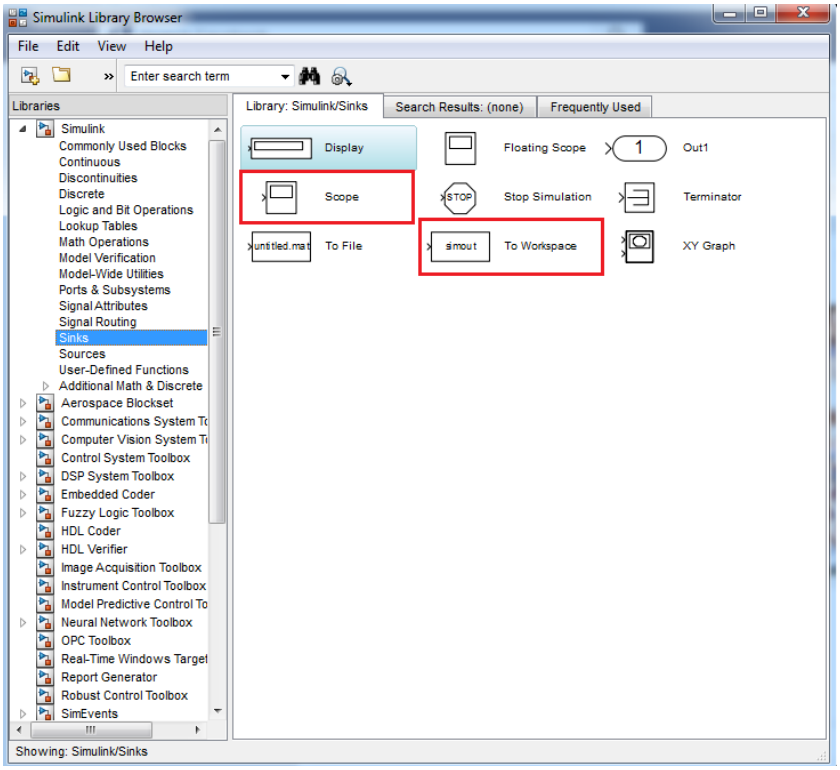
"Ramp" Block: This represents a ramp input. The properties for this block include the gradient of the signal and also the time the signal is produced.

"Sine" Wave: This represents a sinusoidal input. The properties for this block include the amplitude, frequency and phase for this input.

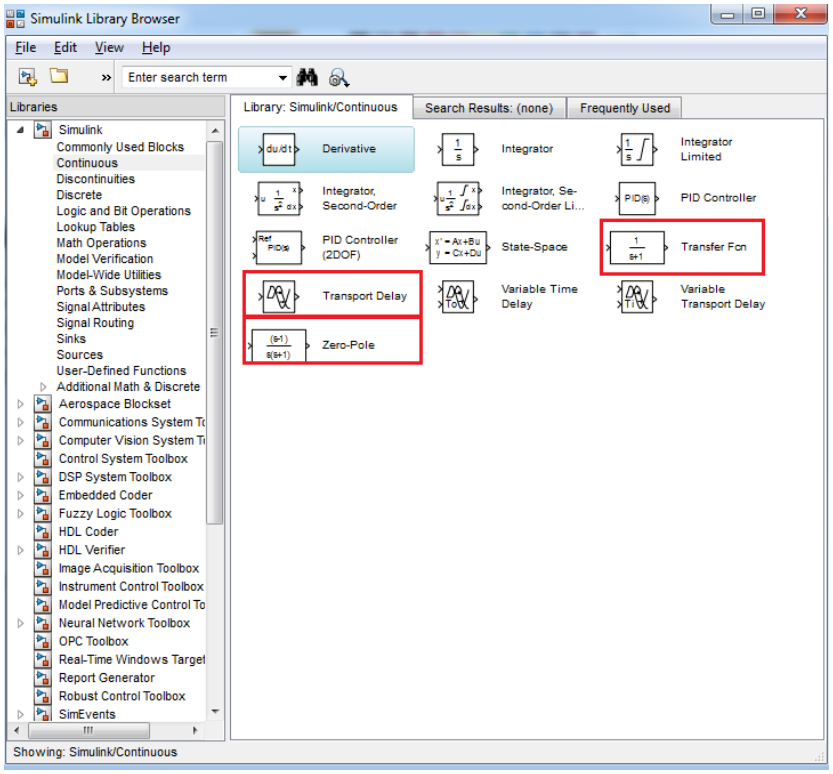
Sinks: This tab contains the output blocks. We shall focus on two main blocks in this category.

"Scope" Block: This block creates a diagram of the output signal against and time. To do so the user has to create the desired system and connect it to the Scope block.

"To Workspace" Block: This block allows the user to save the results of the simulation as variables in the workspace. In the options of this block the user can change the name of the variable and should also select "Array" in the "Save Format" block (it is "Timeseries" by default). Finally the user should open the "Simulation" tab in the model window and then go to *"Model Configuration Parameters"*. On the next window the user shall click the *"Data Import/Export"* tab and set the *"Format"* to *"Array"*. This should be done in order to save the time variable as an array in the workspace.



Continuous: This tab contains the blocks needed to insert transfer function subsystems.



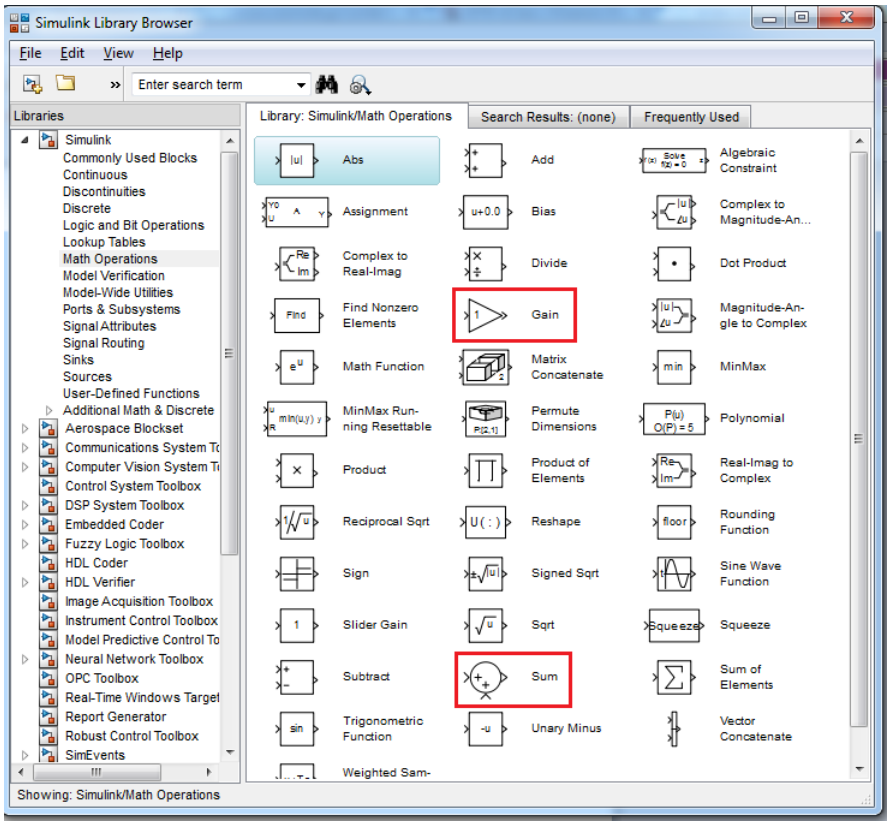
In this tab there are three basic blocks. The "Transfer Fcn" and the "Zero-Pole" that are used to define a transfer function and the "Transport Delay" that is used to define a "Delay System".

"Transfer Fcn" Block: This block allows the user to define the transfer function of a system using the numerator and denominator coefficients.

"Zero-Pole" Block: This block is also used to define a transfer function but this time the data needed are the zeros and poles of the system. In the options of this block the user can also adjust the gain.

"Transport Delay" Block: This block is used to define a delay system. The user shall place this block in series to a transfer function block.In the options the user can change the dead time and the initial value for the system.

Math Operators: Although this tab contains many useful blocks we shall focus on two of them, that are the most important for the purposes of this book. These are the "Gain" and "Sum" blocks.



"Gain" block: This block is used when the user needs to multiply with a real coefficient. This real number is defined in the options.

"Sum" block: This block is used to add different signals. It is used to create parallel connections of subsystems and also for feedback. In its options the user can select the shape of the block (it is a circle by default) and also the number of the signals added

along with their sign. In the "list of signs" section the user can use the "|" symbol to organize the space between the input ports.

6.3 Examples

Example 6.3.1. [Nise, 2013, Jiayu et al., 2009, Elarafi and Hisham, 2008] Mathematical modeling and control of pH processes are quite challenging since the processes are highly nonlinear, due to the logarithmic relationship between the concentration of hydrogen ions [H+] and pH level. The transfer function from input pH to output pH is

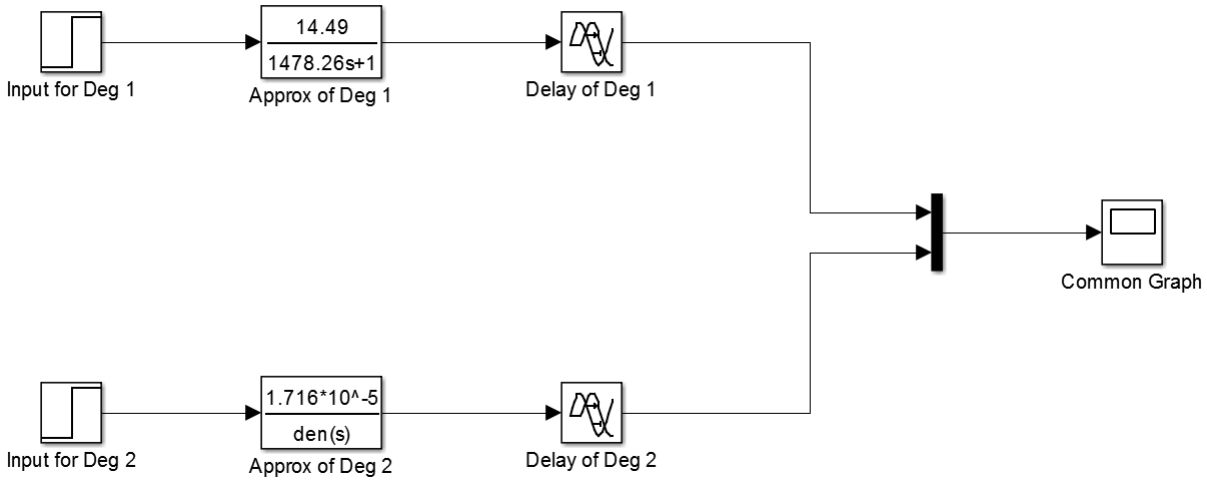
$$G_a(s) = \frac{14.49e^{-4s}}{1478.26s + 1}$$

$G_a(s)$ is a model for the anaerobic process in a wastewater treatment system in which methane bacteria need the pH to be maintained in its optimal range from 6.8 to 7.2 [Jiayu et al., 2009]. Similarly, [Elarafi and Hisham, 2008] used empirical techniques to model a pH neutralization plant as a second-order system with a pure delay, yielding the following transfer function relating output pH to input pH:

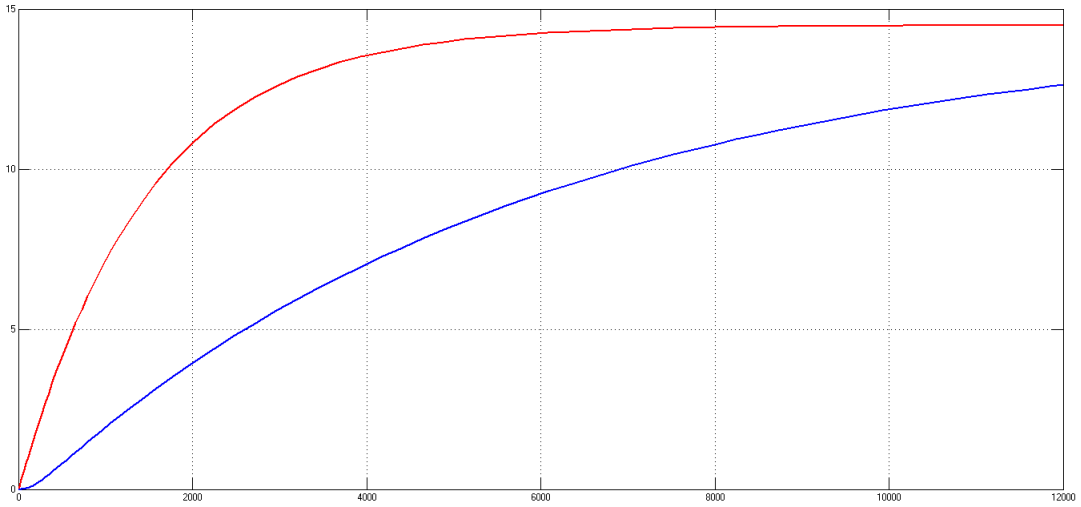
$$G_p(s) = \frac{1.716 * 10^{-5}e^{-30s}}{s^2 + 6.989 * 10^{-3}s + 1.185 * 10^{-6}}$$

Use Simulink to plot the the unit-step responses $y_a(t)$ and $y_p(t)$ for the two processes $G_a(s)$ and $G_p(s)$ on the same graph.

Solution. The first step to the solution of any example with Simulink is to decide the blocks needed. We observe that there are two systems defined by their transfer functions, both using the step function. This means we shall need two step blocks and two transfer function blocks. Next thing is to remember that the exponential terms that appear in both transfer functions refer to a delay system. So two Transport Delay blocks are needed too. Finally we shall plot the graph for the step responses in one graph so we need the Mux and Scope blocks. After placing those blocks on a new Model and connecting them the Model shall look like this



Next thing is to run the simulation and open the scope block to see the common graph for these two transfer functions. Running the simulation for 12,000 seconds and autoscale on the graph, the result is the following:



Chapter 7

State Space Systems

7.1 Introduction

The state space approach, also referred to as the modern, or time-domain approach, is a method for modelling, analysing and designing a wide range of both linear and non-linear systems. State space models use state variables to describe a system by a set of first-order differential equations. In this chapter, we will present methods for creating a state space model, ways to convert between state space and transfer function and show how to calculate various systems' responses.

7.2 State space representation

A system is represented in state space by the following equations:

$$x'(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where:

- $\boldsymbol{x(t)}$: State vector
- $\boldsymbol{u(t)}$: Input/control vector
- $\boldsymbol{y(t)}$: Output vector
- $\boldsymbol{A} \in \mathbb{R}^{n \times n}$: System matrix. It relates how the current state affects the state change x'
- $\boldsymbol{B} \in \mathbb{R}^{n \times m}$: Input/control matrix. It determines how the system input affects the state change
- $\boldsymbol{C} \in \mathbb{R}^{l \times n}$: Output matrix. Determines the relationship between the system state and the system output
- $\boldsymbol{D} \in \mathbb{R}^{l \times m}$: Feedthrough/feedforward matrix. Allows the system input to affect the system output directly

The first equation is called the *state equation* and the second the *output equation*.

In order to create a state space system, we need to know the matrices A, B, C, D. We can define a state space system by using the function

```
sys = ss(A,B,C,D)
```

Given that we have already defined our state space system in the workspace as we did with the above command, we can extract matrices A, B, C, D using

```
[A,B,C,D]=ssdata(sys) % Returns matrices A,B,C,D
```

We can convert from a transfer function to a state space system, in other words, if we have a transfer function, we can find the state space representation that corresponds to this transfer function. To achieve this use

```
[A,B,C,D] = tf2ss(num,den);
sys = ss(A,B,C,D)
```

Function `tf2ss(num,den)` returns matrices A, B, C, D. Note that, we have to assign 4 output variables, in this case A, B, C, D, otherwise Matlab will not return the complete result. Then, we create the state space model with the command `ss`.

An alternative way of converting to a state space model from a transfer function, is by using

```
sys_ss = ss(sys)
```

provided that, we have already stored the transfer function inside the variable *sys*. This method though has a disadvantage, since the matrices A, B, C, D are not saved as variables in the workspace. On the other hand, in order to convert from a state space model to a transfer function, we can use

```
[num,den] = ss2tf(A,B,C,D);
sys = tf(num,den)
```

Function `ss2tf(A,B,C,D)` returns the numerator and the denominator of the transfer function. Then, we create the transfer function with the command `tf`.

In case we have multiple inputs or outputs in our system and therefore the transfer function is a matrix, the function

```
[num,den] = ss2tf(A,B,C,D,n)
```

returns the numerator and denominator of the transfer function that results when the n-th input of our system is excited by a unit impulse.

Alternatively, if we have already defined our system in state space form inside the variable *sys* we can use

```
sys_tf = tf(sys)
```

In a similar way, one can use the following commands:

```
[A,B,C,D] = zp2ss(z,p,k);
sys = ss(A,B,C,D)
```

Converts a zero-pole-gain model, to a state space model.

```
[z,p,k] = ss2zp(A,B,C,D,i);
sys = zpk(z,p,k)
```

Converts a state space model, to a zero-pole-gain model, from the i-th input(using the i-th columns of B and D).

Generally, for systems with multiple inputs and outputs, the conversion from a state space model to a transfer function is easier when using the command `tf(sys)` or `zpk(sys)` than `ss2tf(A,B,C,D,ni)` or `ss2zp(A,B,C,D,i)`, due to the fact that such a system will be represented by more than one transfer functions and the above commands only produce the numerators and denominators of the transfer function. Therefore it's up to the user to extract the different combinations of numerators and denominators to create all the transfer functions that represent the system. On the other hand, commands like `tf(sys)` and `zpk(sys)`, return all the transfer functions that represent the system in a straightforward way.

7.2.1 Examples

Example 7.2.1. Consider the following transfer function

$$G(s) = \frac{s+1}{s^2+2s+1}$$

Find the state space model, that corresponds to this transfer function.

Solution.

```
num = [1 1];
den = [1 2 1];

[A,B,C,D] = tf2ss(num,den); % Convert from tf to ss.
sys = ss(A,B,C,D)
```

Example 7.2.2. Consider the following transfer function

$$K(s) = \frac{s}{2s^2 + 3s + 1}$$

Find the state space model, that corresponds to this transfer function.

Solution.

```
num = [1 0];
den = [2 3 1];
sys_tf = tf(num,den)

sys_ss = ss(sys_tf) % Convert from tf to ss

% Note, that in this way matrices A,B,C,D are not saved in the workspace.
```

Example 7.2.3. Consider the following transfer function in factored form

$$K(s) = \frac{2(s-2)(s-3)}{s(s-1)}$$

Find the state space model, that corresponds to this transfer function.

Solution.

```
z = [2 3];
p = [0 1];
k = 2;
[A,B,C,D] = zp2ss(z,p,k)
sys = ss(A,B,C,D)
```

Example 7.2.4. Consider the following state space model

$$x'(t) = \begin{pmatrix} 2 & 1 \\ 0 & -3 \end{pmatrix} x(t) + \begin{pmatrix} 2 \\ 1 \end{pmatrix} u(t)$$
$$y(t) = \begin{pmatrix} 1 & 1 \end{pmatrix} x(t)$$

Find the zero-pole-gain model, that corresponds to this state space model.

Solution.

```
A = [2 1;0 -3];
B = [2;1];
C = [1 1];
D = 0;
[z,p,k] = ss2zp(A,B,C,D)
% B and D have 1 column, this means ss2zp(A,B,C,D) is the same as ...
   ss2zp(A,B,C,D,1)
sys = zpk(z,p,k)
```

Example 7.2.5. [Nise, 2013, Cavallo et al., 1992] Given the F4-E military aircraft, normal acceleration a_n and pitch rate q are controlled by elevator deflection δ_e on the horizontal stabilizers and by canard deflection δ_c. A commanded deflection δ_{com}, is used to effect a change in both δ_e and δ_c. The state equations describing the effect of δ_{com} on a_n and q is given by

$$\begin{pmatrix} \dot{a_n} \\ \dot{q} \\ \dot{\delta_e} \end{pmatrix} = \begin{pmatrix} -1.702 & 50.72 & 263.38 \\ 0.22 & -1.418 & -31.99 \\ 0 & 0 & -14 \end{pmatrix} \begin{pmatrix} a_n \\ q \\ \delta_e \end{pmatrix} + \begin{pmatrix} -272.06 \\ 0 \\ 14 \end{pmatrix} \delta_{com}$$

$$\begin{pmatrix} a_n \\ q \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_n \\ q \\ \delta_e \end{pmatrix}$$

Find the transfer functions that correspond to this state space system.

Solution.

```
A = [-1.702 50.72 263.38;0.22 -1.418 -31.99;0 0 -14];
B = [-272.06;0;14];
C = [1 0 0;0 1 0];
D = [0;0];
sys_ss = ss(A,B,C,D);
sys_zpk = zpk(sys_ss)
```

Matlab returns the following transfer functions:

$$(1) : \frac{-272.06(s^2 + 1.865s + 84.13)}{(s + 4.903)(s + 14)(s - 1.783)}$$

$$(2) : \frac{-507.71(s + 1.554)}{(s + 4.903)(s + 14)(s - 1.783)}$$

We get this result because our system has 1 input and 2 outputs.

Transfer function (1) describes the relation between the input and the *first* output.

Transfer function (2) describes the relation between the input and the *second* output.

Example 7.2.6. [Nise, 2013, Liceaga-Castro and van der Molen, 1995] An autopilot is to be designed for a submarine to maintain a constant depth under severe wave disturbances. It has been shown that the system's linearised dynamics under neutral buoyancy and at a given constant speed are given by:

$$\begin{pmatrix} \dot{w} \\ \dot{q} \\ \dot{z} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} -0.038 & 0.896 & 0 & 0.0015 \\ 0.0017 & -0.092 & 0 & -0.0056 \\ 1 & 0 & 0 & -3.086 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} w \\ q \\ z \\ \theta \end{pmatrix} + \begin{pmatrix} -0.0075 & -0.023 \\ 0.0017 & -0.0022 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \delta_B \\ \delta_S \end{pmatrix}$$

$$\begin{pmatrix} z \\ \theta \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} w \\ q \\ z \\ \theta \end{pmatrix}$$

where,

- $\boldsymbol{w}$: The heave velocity
- $\boldsymbol{q}$: The pitch rate
- $\boldsymbol{z}$: The submarine depth
- $\boldsymbol{\theta}$: The pitch angle
- $\boldsymbol{\delta_B}$: The bow hydroplane angle
- $\boldsymbol{\delta_S}$: The stern hydroplane angle

(For an explanatory image, please check out the references). Find the transfer function that corresponds to this state space system.

Solution.

```
A = [-0.038 0.896 0 0.0015;0.0017 -0.092 0 -0.0056;1 0 0 -3.086;0 1 0 0];
B = [-0.0075 -0.023;0.0017 -0.0022;0 0;0 0];
C = [0 0 1 0;0 0 0 1];
D = zeros(2);
sys = ss(A,B,C,D);
sys_tf = tf(sys)
```

Matlab returns 4 transfer functions, because we have 2 inputs and 2 outputs in our system. These are the following:

```
  From input 1 to output...
          -0.0075 s^2 - 0.004413 s - 0.0001995
   1:  ---------------------------------------
       s^4 + 0.13 s^3 + 0.007573 s^2 + 0.0002103 s

                  0.0017 s + 5.185e-05
   2:  ------------------------------------
       s^3 + 0.13 s^2 + 0.007573 s + 0.0002103

  From input 2 to output...
           -0.023 s^2 + 0.002702 s + 0.0002466
   1:  ---------------------------------------
       s^4 + 0.13 s^3 + 0.007573 s^2 + 0.0002103 s

                  -0.0022 s - 0.0001227
   2:  ------------------------------------
       s^3 + 0.13 s^2 + 0.007573 s + 0.0002103

Continuous-time transfer function.
```

Example 7.2.7. [Nise, 2013, Tari et al., 2005] In the past, Type-1 diabetes patients had to inject themselves with insulin three to four times a day. New delayed-action insulin analogues such as insulin Glargine require a single daily dose. For a specific patient, state-space model matrices are given by

$$x'(t) = \begin{pmatrix} -0.435 & 0.209 & 0.02 \\ 0.268 & -0.394 & 0 \\ 0.227 & 0 & -0.02 \end{pmatrix} x(t) + \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} u(t)$$
$$y(t) = \begin{pmatrix} 0.0003 & 0 & 0 \end{pmatrix} x(t)$$

where the state variables, input and output are:

- $\boldsymbol{x_1}$: Insulin amount in plasma compartment
- $\boldsymbol{x_2}$: Insulin amount in liver compartment
- $\boldsymbol{x_3}$: Insulin amount in interstitial (in body tissue) compartment
- $\boldsymbol{u}$: External insulin flow
- $\boldsymbol{y}$: Plasma insulin concentration

Find the system's transfer function.

Solution.

```
A = [-0.435 0.209 0.02;0.268 -0.394 0;0.227 0 -0.02];
B = [1;0;0];
C = [0.0003 0 0];
D = 0;
[num,den] = ss2tf(A,B,C,D);
sys = tf(num,den)
```

The resulting transfer function is

$$G(s) = \frac{0.0003s^2 + 0.0001242s + 2.364e-06}{s^3 + 0.849s^2 + 0.1274s + 0.0005188}$$

Chapter 8

Pole Placement

8.1 Introduction

In the present Chapter, we will present examples of the Pole Placement technique. It is known that when the system is controllable, the poles of the closed loop system can be placed at any desired point using an appropriate feedback gain matrix. In the following, we present an analysis of the procedure and more importantly, the commands we are going to use, like `k=place(A,B,P)`.

8.2 Basic definitions and functions

Before we proceed to the pole placement technique, first we need to give some important definitions regarding the stability, controllability and stabilizability of a system.

Asymptotic Stability: A system is asymptotically stable if the eigenvalues of the matrix A (or the poles of the system) are located in the left-half of the complex plane.

Critical Stability: A system is critically stable, if at least one single eigenvalue of the matrix A (or a pole of the system) is located on the imaginary axis and no pole is located in the right-half plane of complex numbers. In addition, no poles with multiplicity more than one are located on the imaginary axis.

Instability: A system is unstable if at least one eigenvalue of the matrix A (or a pole of the system) is located in the right-half complex plane, or poles with multiplicity higher than one are located on the imaginary axis.

Controllability: A system is considered to be controllable, if for every initial condition $x(0) \neq 0$ and time $t_1 > 0$, there exists an input signal $u(t)$ and time t_1, such that the state of the system can be driven to the origin in finite time, i.e. $x(t_1) = 0$.

Stabilizability: A system is considered to be stabilizable, if for each initial state $\xi \in \mathbb{R}^n$, there exists a control input $u(t)$, such that the state response with initial condition

$x(0) = \xi$ satisfies, $\lim_{x\to\infty} x(t) = 0$

In other words, when a system is stabilizable, it means that we can make it stable. Additionaly, when a system is controllable, it is stabilizable. As you will see, in the following examples when we need to perform the pole placement technique in order to make a system stable or change it's response, we first examine whether the system is controllable or not. If it is controllable, it means that it's stabilizable and therefore we can perform the pole placement technique in order to make it stable or change it's response.

In order to examine whether our system is stable, we have to calculate the eigenvalues of matrix A. Given that we have already defined the matrix A in the workspace, we can achieve that by using the command

```
eigs = eig(A)
```

Alternatively, given that we have already defined our system in the workspace inside variable *sys* (and not just matrix A) we can use the command

```
isstable(sys)
```

Matlab then returns a logical value of whether 1 or 0, where 1 means "True" and 0 means "False".

A n-th order system is controllable, if the controllability matrix

$$Q = \begin{pmatrix} B & AB & \dots & A^{n-1}B \end{pmatrix}$$

has full rank, i.e. $rank(Q) = n$. We can examine the controllability of a system by using

```
Q = ctrb(A,B); % Returns matrix Q
rank(Q)
```

Function `ctrb(A,B)` returns the controllability matrix Q. Then, using the function `rank(Q)` we find the rank of the matrix and if it's in full rank, our system is controllable.

Alternatively, given that we have already defined our system in the workspace we can use

```
Q = ctrb(sys); % Returns matrix Q
rank(Q)
```

8.3 Pole placement through state feedback

Consider the following state space system

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t)$$

We define a new input $u(t)$ of the form, $u(t) = -Kx(t) + r(t)$, where

$$K = \begin{pmatrix} k_0 & k_1 & \dots & k_{n-1} \end{pmatrix} \in \mathbb{R}^{1\times n}$$

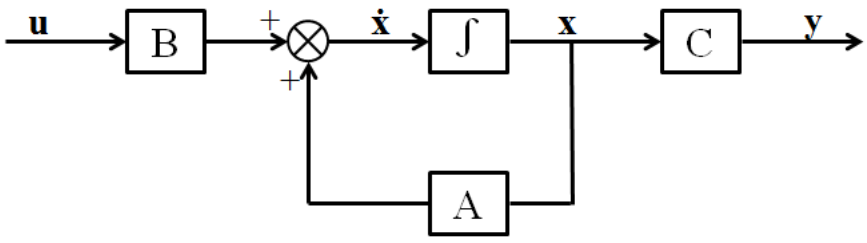
is the feedback gain matrix and $r(t)$ is a new input signal. Thus,

$$u(t) = -\begin{pmatrix} k_0 & k_1 & \dots & k_{n-1} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + r(t) \Rightarrow$$
$$u(t) = -k_0x_1 - k_1x_2 - \dots - k_{n-1}x_n + r(t)$$

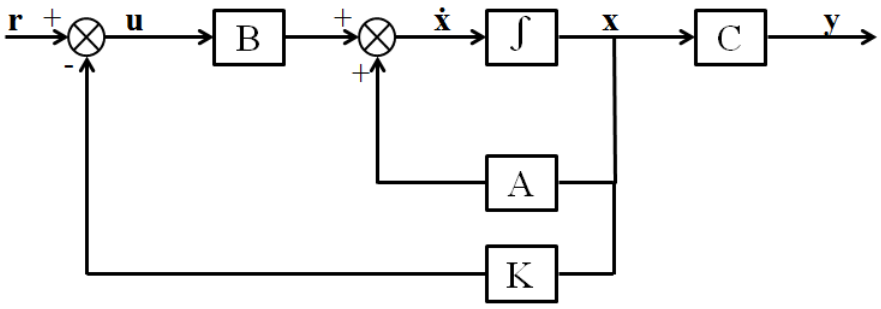
This is called state feedback. As you can see we feed back to the input $u(t)$ the states $x_1, \dots x_n$ multiplied by the elements of matrix K, which are called gains. By replacing $u(t)$ in our state space equation we get $\dot{x}(t) = Ax(t)+B[-Kx(t)+r(t)] = (A-BK)x(t)+Br(t)$. The state space system has the following form:

$$\dot{x}(t) = (A - BK)x(t) + Br(t)$$
$$y(t) = Cx(t)$$

Now, system stability depends on the location of the eigenvalues of matrix $A - BK$. Our purpose is to find the appropriate gains, in other words matrix K, in order to have the desired placement of the eigenvalues (of matrix $A - BK$). Of course this placement will take place in the left-half plane of the complex numbers, in order for our system to be stable.



State space system representation without state feedback



State space system representation with state feedback

If our system is controllable and stabilizable, pole placement can be achieved with the command

```
K = place(A,B,P)
```

By placing our desired eigenvalues in variable P, Matlab then calculates and returns matrix K, inside the variable K. Variables A and B contain matrices A and B respectively. Note, that command $place(A, B, P)$ can not perform pole placement for eigenvalues with multiplicity greater than the number of inputs of the system.

8.4 Examples

Example 8.4.1. Consider the following state space system

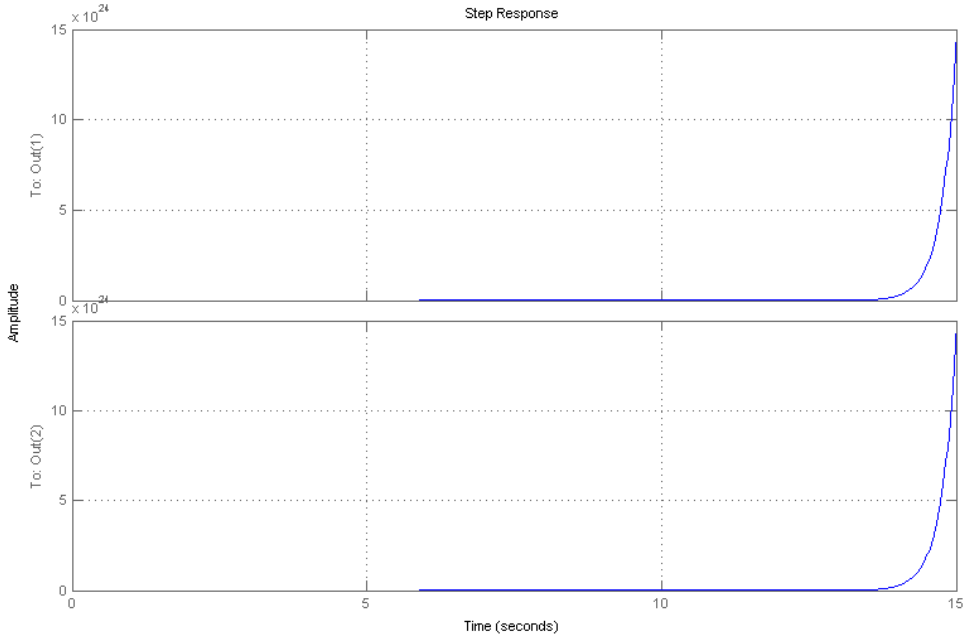
$$\begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}' = \begin{pmatrix} 1 & 3 \\ 3 & 1 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u(t)$$
$$y(t) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

1. Find the poles of the following system. Is it stable?
2. Is the system controllable?
3. Find a feedback matrix k such that the closed loop system has poles at $-1 \pm 2i$.
4. Find the new system and plot its step response.

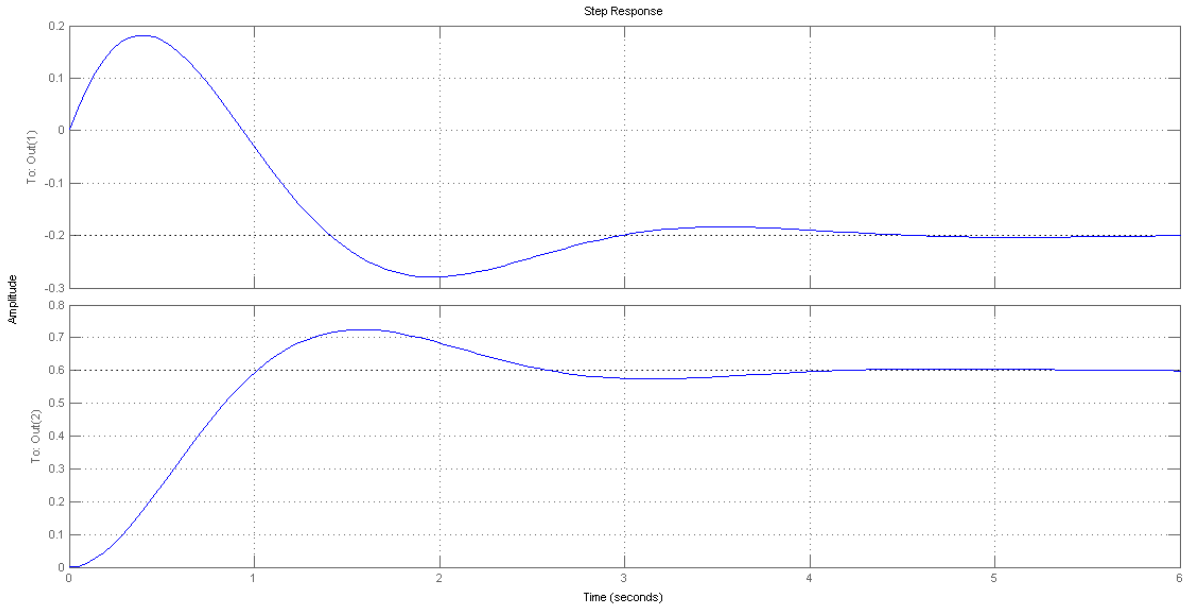
Solution.

```
% Begin by defining the system
a=[1, 3;3, 1];
b=[1;0];
```

```
c=[1,0;0,1];
d=[0;0];
sys=ss(a,b,c,d);
eig(sys)
% This results in -2,4 so the system is unstable

% Now we check the stability
rank(ctrb(sys))
% The matrix has full rank, so the system is controllable.

% We compute the feedback matrix
k=place(a,b,[-1+2i,-1-2i]);

% The new system is
sys_with_fb=ss(a-b*k,b,c-d*k,d)

step(sys)
grid
figure(2)
step(sys_with_fb)
grid
```



Step response without state feedback



Step response with state feedback

Example 8.4.2. [Nise, 2013, Liu et al., 2008] The use of feedback control to vary the pitch angle in the blades of a variable speed wind turbine, allows power generation optimization under variable wind conditions. At a specific operation point, it is possible to linearise turbine models. For example, the model of a three-blade turbine with a 15 m radius working in 12 m/s wind speed and generating 220 V can be expressed as:

$$\begin{pmatrix} \beta' \\ \xi' \\ \xi'' \\ \omega_g' \\ \omega_{gm}' \end{pmatrix} = \begin{pmatrix} -5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -10.5229 & -1066.67 & -3.38028 & 23.5107 & 0 \\ 0 & 993.804 & 3.125 & -23.5107 & 0 \\ 0 & 0 & 0 & 10 & -10 \end{pmatrix} \begin{pmatrix} \beta \\ \xi \\ \xi' \\ \omega_g \\ \omega_{gm} \end{pmatrix} + \begin{pmatrix} 5 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} u(t)$$

$$y(t) = \begin{pmatrix} 0 & 0 & 0 & 1.223 \cdot 10^5 & 0 \end{pmatrix} \begin{pmatrix} \beta \\ \xi \\ \xi' \\ \omega_g \\ \omega_{gm} \end{pmatrix}$$

where,

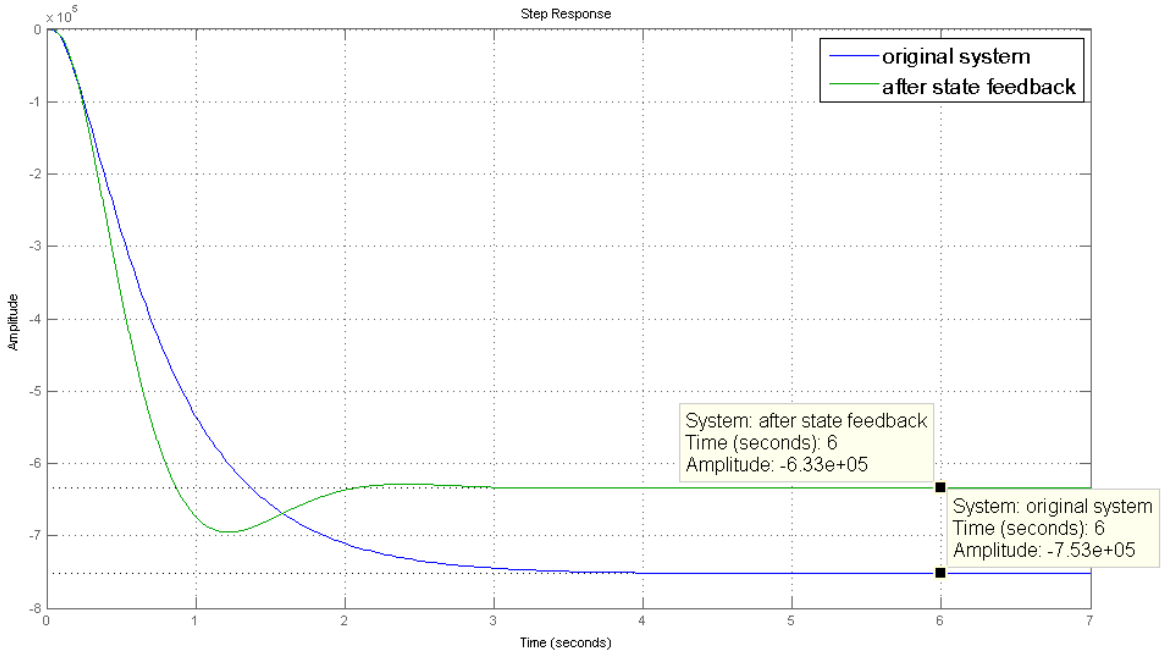
- $\boldsymbol{\beta}$: Pitch angle of the wind turbine blades
- $\boldsymbol{\xi}$: Relative angle of the secondary shaft
- $\boldsymbol{\omega_g}$: Generator speed
- $\boldsymbol{\omega_{gm}}$: Generator measurement speed
- $\boldsymbol{u(t)}$: Pitch angle reference
- $\boldsymbol{y(t)}$: Active power generated

1. Find the system's eigenvalues. Is it stable?
2. Examine if the system is controllable. Then, through state feedback, find a matrix K so that the system have 2 poles that approximate a second order system with characteristic polynomial $s^2 + 4s + 11$. Place the remaining 3 poles to a distance of at least 10 times in comparison to the other 2 poles, in the left half-plane of complex numbers.
3. Find the closed-loop system for matrix K.
4. Plot the step response of the open and closed-loop system in the same window.

Solution.

```
A = [-5 0 0 0 0;0 0 1 0 0;-10.5229 -1066.67 -3.38028 23.5107 0;0 ...
    993.804 3.125 -23.5107 0;0 0 0 10 -10];
B = [5;0;0;0;0];
C = [0 0 0 1.2331e5 0]; % 1.2331e5 means 1.2331*10^5
D = 0;
sys=ss(A,B,C,D)

eigs = eig(sys)
% The eigenvalues are
% -1.6620 + 0.0000i
% -12.6145 +29.5231i
% -12.6145 -29.5231i
% -5.0000 + 0.0000i
% -10.0000 + 0.0000i

% Our system is stable

sys_ctrb = rank(ctrb(A,B)) % We check for controllability

% Rank is 5, our system is controllable. This means that we can place ...
    our eigenvalues in desired locations

pol_roots = roots([1 4 11]) % Roots are -2.0000 + 2.6458i, -2.0000 - ...
    2.6458i

K = place(A,B,[-2+2.6458*i,-2-2.6458*i,-20,-21,-22])

% We choose the 2 eigenvalues to be the roots of the polynomial we want ...
    to approximate. We place the remaining 3 in a distance of at least ...
    10 times in the left half-plane. We choose -20,-21,-22

feedback_sys=ss(A-B*K,B,C,D); % Closed-loop system

step(sys)
hold all
```

```
step(feedback_sys)
legend('original system','after state feedback')
grid
```



Chapter 9

Observer Design

9.1 Introduction

Suppose a linear, time invariant siso system S is given. Let S be defined by

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t)$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times 1}$, $C \in \mathbb{R}^{1\times n}$.

For this system the input u(t) is considered a known signal with $u(t) = 0$ for $t \leq 0$. In addition, we usually also assume that the output and state vectors y(t),x(t) are known and can be measured for all $t \geq 0$. This makes it possible (as was seen in the previous chapter) to apply state feedback in order to change the characteristics of the system. This may not always be the case though, since in practice, not all states

$$x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{pmatrix}, \ t \geq 0$$

could be available for feedback. If that is the case, we need to find a way to estimate the unknown states.

In this case, an equivalent system is constructed, called an *observer*, with the purpose of estimating all the states of the system. The new observer system is characterised by the same equations, i.e.

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t)$$
$$\hat{y}(t) = C\hat{x}(t)$$

In order to ensure that the estimation is accurate, the observer is additionally fed with the error signal between the outputs, i.e. $y(t) - \hat{y}(t)$, multiplied by a gain matrix L. The

new system takes the form

$$\begin{gathered}\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L\big(y(t) - \hat{y}(t)\big) \Rightarrow \\ \dot{\hat{x}} = (A - LC)\hat{x}(t) + Ly(t) + Bu(t) \\ \hat{y}(t) = C\hat{x}(t)\end{gathered}$$

The use of the output feedback can be understood considering the approximation error. Let $\epsilon(t)$ denote the error of the approximation, i.e. $\epsilon(t) = x(t) - \hat{x}(t)$. Substituting the values of the state and estimated state, we end up with

$$\dot{\epsilon} = (A - LC)\epsilon(t)$$

The solution of this homogeneous linear diff. equation is

$$e(t) = e^{(A-LC)t}e(0)$$

So it should be obvious that if the feedback matrix L is chosen in order to have a stable system matrix (A-LC), the error $e(t) \to 0$ and the approximation is accurate. In addition, depending on the value of L, we may obtain quicker or slower approximations. For a thorough presentation of the procedure of observer design, one may refer to [Ogata, 2009].

9.2 Observer Construction Example

The method used for constructing an observer will be explained through the following example.

Example 9.2.1 (Observer Design)**.** Consider the following system

$$\begin{pmatrix} x_1'(t) \\ x_2'(t) \end{pmatrix} = \begin{pmatrix} -20.6 & 2 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} + \begin{pmatrix} 5 \\ 1 \end{pmatrix} u(t)$$

$$y(t) = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

With initial conditions

$$\begin{pmatrix} x_1(0) \\ x_2(0) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

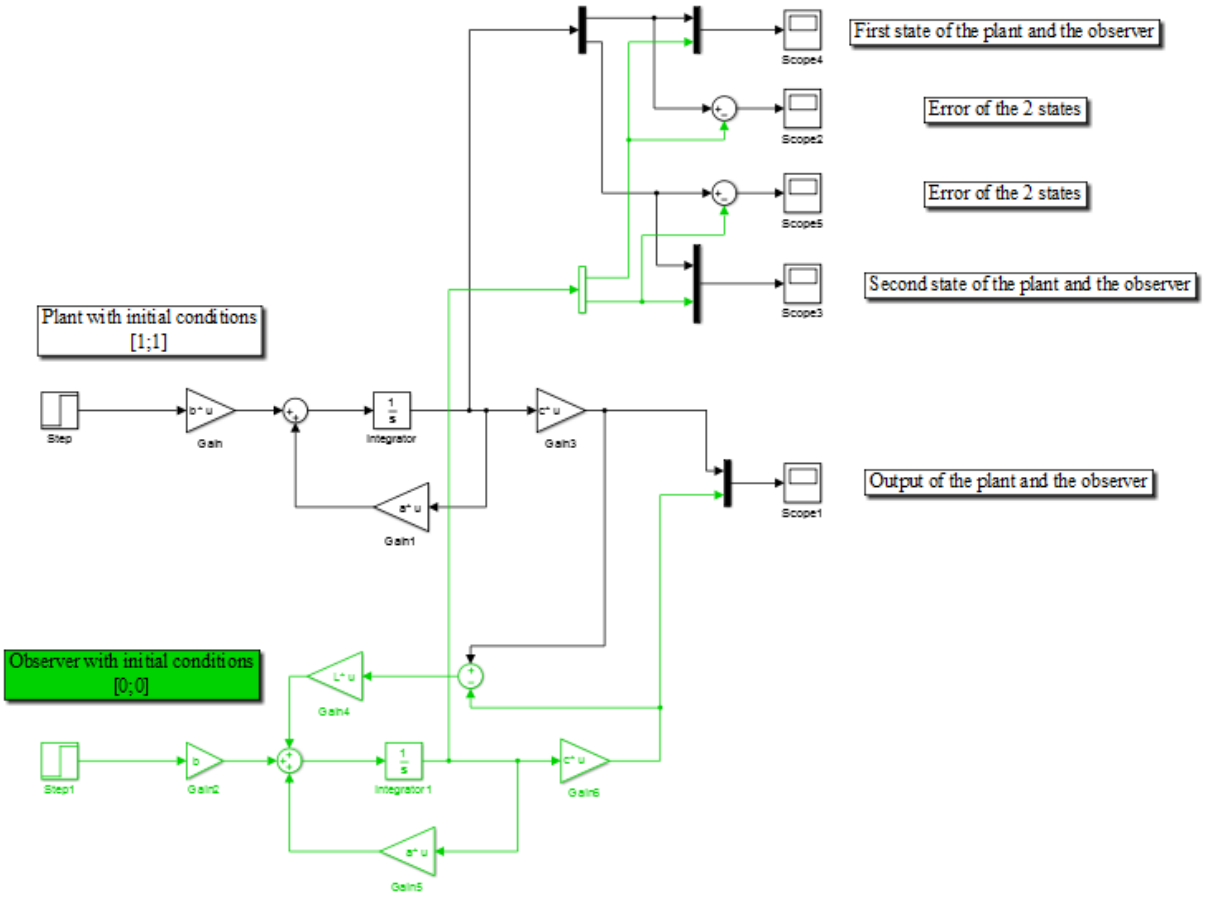
1. Is the system stable?
2. Is the system observable?
3. If the system is observable, create an observer with poles at -10, -9.
4. Assuming different initial conditions for the observer, compare the outputs of the two systems, their states and the error of the observer approximation, for different kinds of inputs and verify the estimation of the observer.

5. Would the approximation be the same if the observer had poles near the imaginary axis, e.g. at -1,-2?

Solution. The first two questions can be answered through Matlab workspace directly. As a reminder, stability is not necessary to create an observer.

```
%% Defining the Plant
a=[-20.6 0;0 -1];
b=[0;1];
c=[1 1];
d=0;
sys=ss(a,b,c,d);
% Now we check the eigenvalues of the system for stability
eig(sys); %the eigenvalues are both negative so its stable
%then we check the observability
rank(obsv(sys));
%the system is observable so we can construct an observer.
```

Since the system is observable, the observer construction is possible. So moving on to Simulink, we create the following system



The above model constitutes of 3 basic parts. The first is the original plant of the system with initial conditions [1,1]. The second part is the observer, created above the original plant and designed with green colour. The error $y(t)-\hat{y}(t)$ is fed to the observer, multiplied by L. For the observer we choose as initial conditions [0,0]. The third part consists of all the scopes for the inputs, the outputs and the errors between real and estimated signals.

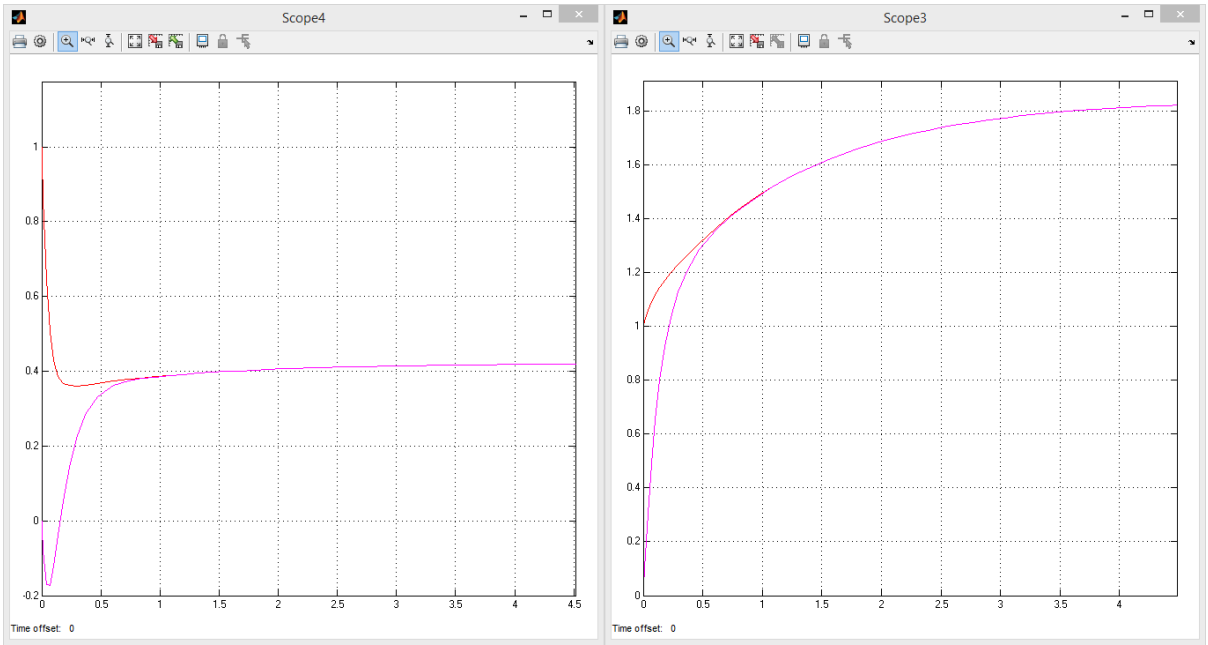
Before we can simulate the system, we must choose the values for L. This is done through the following commands.

```
%% Observer pole placement at  -10 and -9
% This observer will lead to a faster approximation of the two states
L_T=place(a',c',[-10,-9])
L=L_T'
% (the symmetric system is used for pole placement)
```

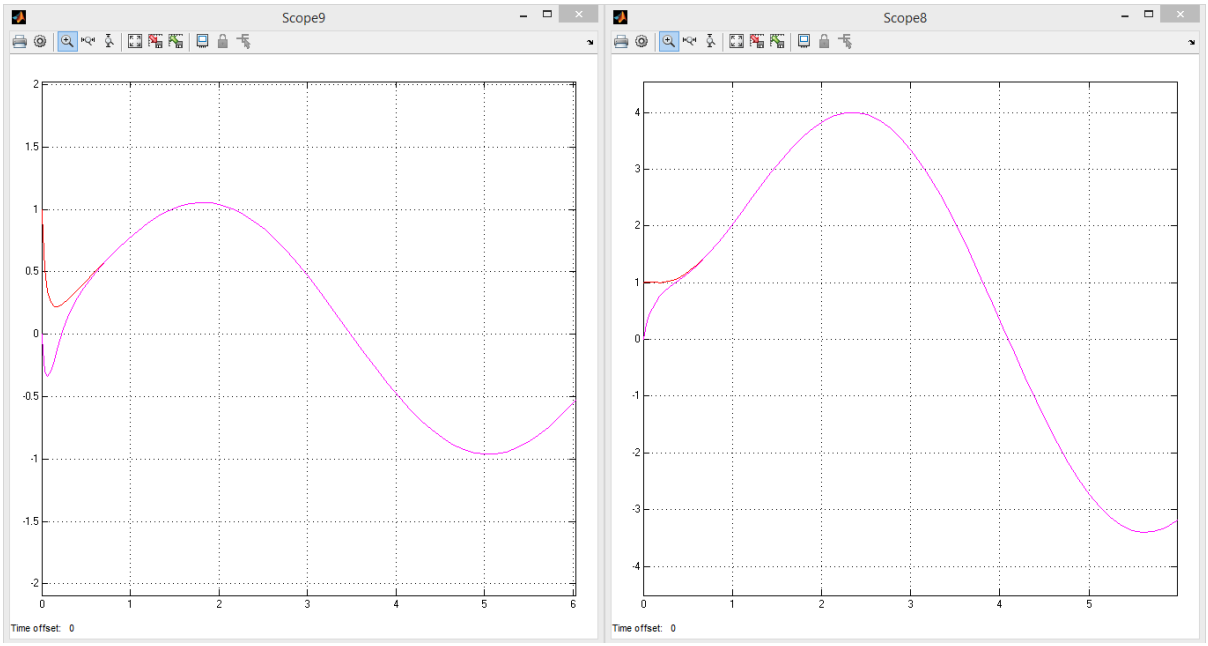
The matrix L is equal to $L = \begin{pmatrix}-6.7429 & 4.1429\end{pmatrix}^T$. Now we are ready to start the simulation.

It should also be noted that we designed the state space systems using only one differentiator block. This was done by changing the way the signals are interpreted. This is possible by choosing 'Matrix Multiplication' at the Gain blocks and then defining all the parameters as matrices in the workspace.

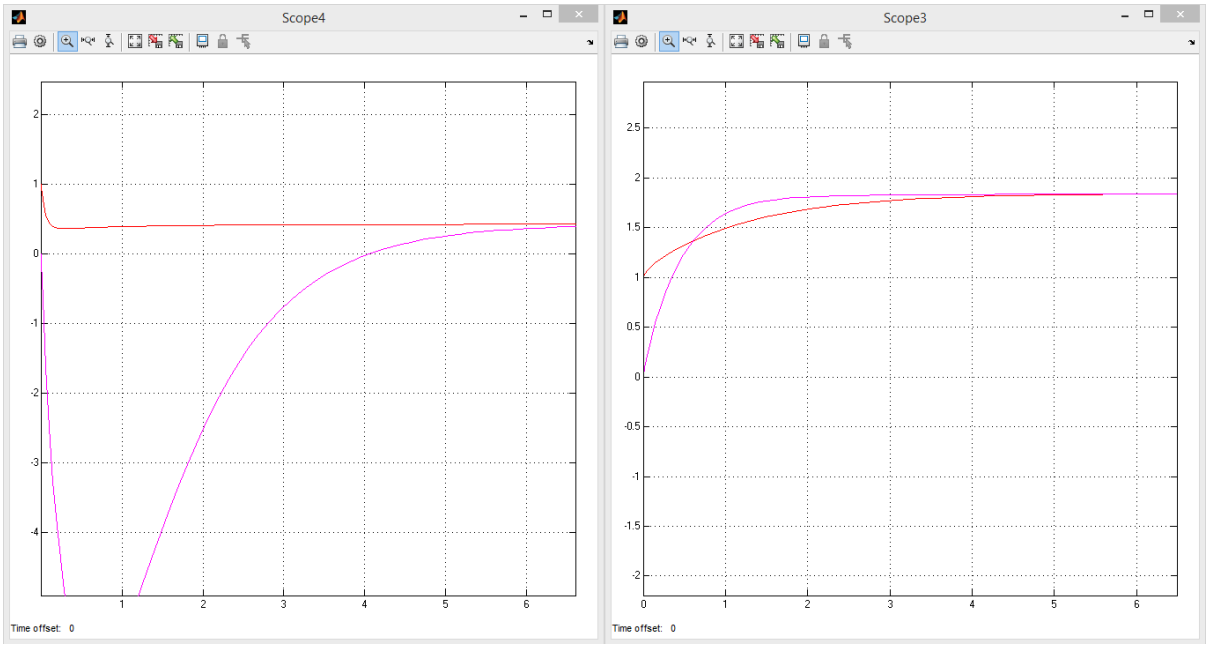
The simulation results for the two states and a step input are the following



From these figures we can observe that the estimation becomes accurate for both states in less that 1 second, which is a satisfying result. This result is independent of the type of input that we use, see for example for $u(t) = 3sin(t)$:



The swiftness of the approximation is due to the fact that the poles of the observer are chosen to be far left on the imaginary axis, making the system's response fast. If we chose poles that would still be stable, but closer on the imaginary axis, then the approximation would take longer time to reach its accuracy. For example if we chose the poles at -1,-2, the result would be the following



and we can see that it take about 4.5 seconds to get the desired result.

Chapter 10

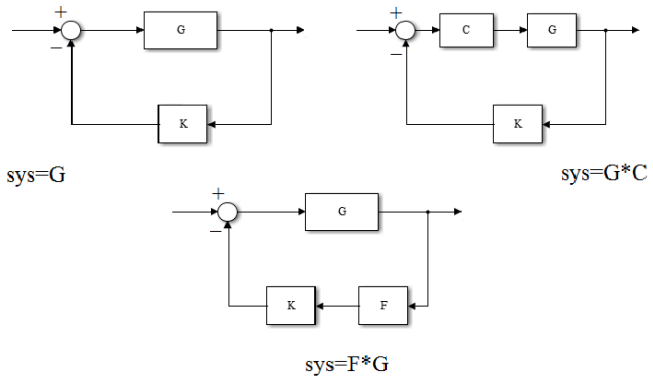
Root Locus Analysis

10.1 Introduction

The Root locus method covered in this chapter, is a very useful tool in system analysis, used to describe qualitatively the performance of a system with feedback when a system parameter, usually the gain, is changed. For example, we can see the effect of varying gain upon percent overshoot, settling time and peak time. It can be used as a graphical design technique, in order to find an appropriate gain value to meet some performance requirements. (For example a specific value of damping ratio.) If the gain adjustment cannot give us the desired result, an addition of a compensator to the system will be necessary.

10.2 Root Locus method

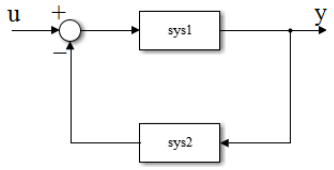
Root locus is a graphical presentation of the system's closed-loop poles, as the gain varies between zero and infinity. The MATLAB command that produces the root locus diagram is `rlocus`.

`rlocus(sys)`	Calculates the root locus diagram of the open-loop SISO model `sys`. This function can be applied to any of the following negative feedback loops by setting `sys` appropriately: 

Some other useful commands are the following:

`sgrid(z,w)`	Plots a grid of constant damping factor and natural frequency lines. The grid is drawn over the root locus diagram.
`sgrid(z,[ ])` `sgrid([ ],w)`	Plots a grid of constant damping factor lines, or a grid of natural frequency lines respectively.
`[R,K]=rlocus(sys)`	Returns the vector K of selected gains and the closed-loop pole locations(vector R) for these gains.
`sys=feedback(sys1,sys2)`	Returns a model object `sys` for the negative feedback interconnection of model objects `sys1` and `sys2`.

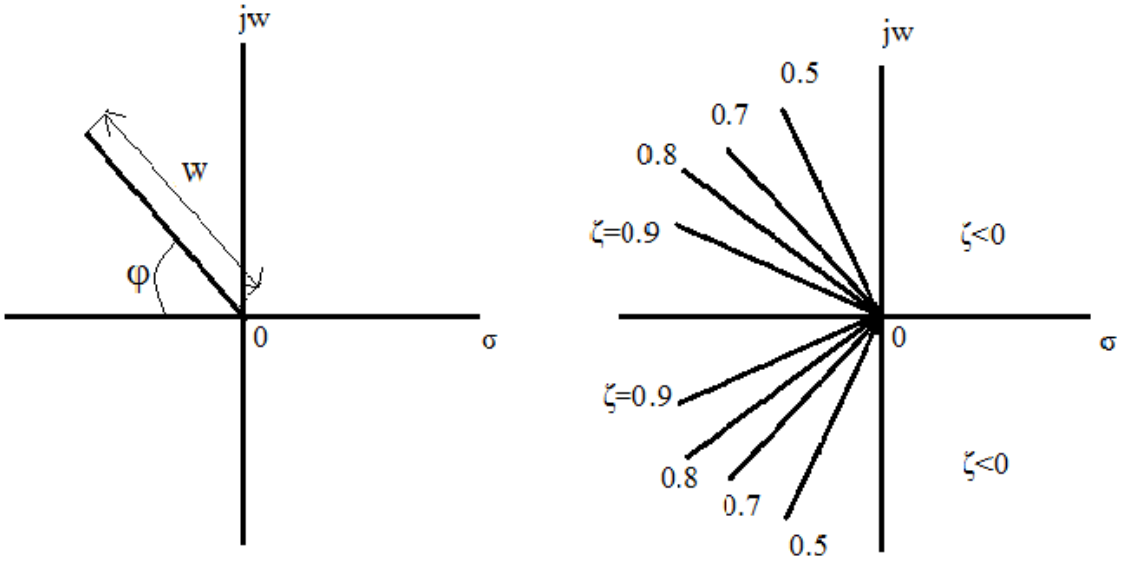
The last command forms the following negative feedback loop:



So if `sys2` is a positive number k, we get the transfer function of the closed-loop system when gain value is equal to k.

Lines of constant damping ratio ζ ($\zeta = cos\phi$) are radial lines passing through the origin as shown below. For example, a damping ratio of 0.5 requires that the complex-conjugate poles lie on the lines drawn through the origin making angles of $\pm 60^{\circ}$ with the negative real axis. (If the real part of a pair of complex-conjugate poles is positive, which

means that the system is unstable, the corresponding ζ is negative.) The damping ratio determines the angular location of the poles, while the natural frequency ω determines the distance of the pole from the origin. (with circles)



Since the root locus actually showcases the locations of all possible closed-loop poles, from the root locus we can select a gain such that our closed-loop system will perform the way we want. If any of the selected poles are on the right half plane, the closed-loop system will be unstable.

As the gain of the open-loop transfer function varies, the characteristic equation changes. So the closed-loop poles, which are the roots of the characteristic equation, change too. Let's see a simple example:

Example 10.2.1. We have the transfer function

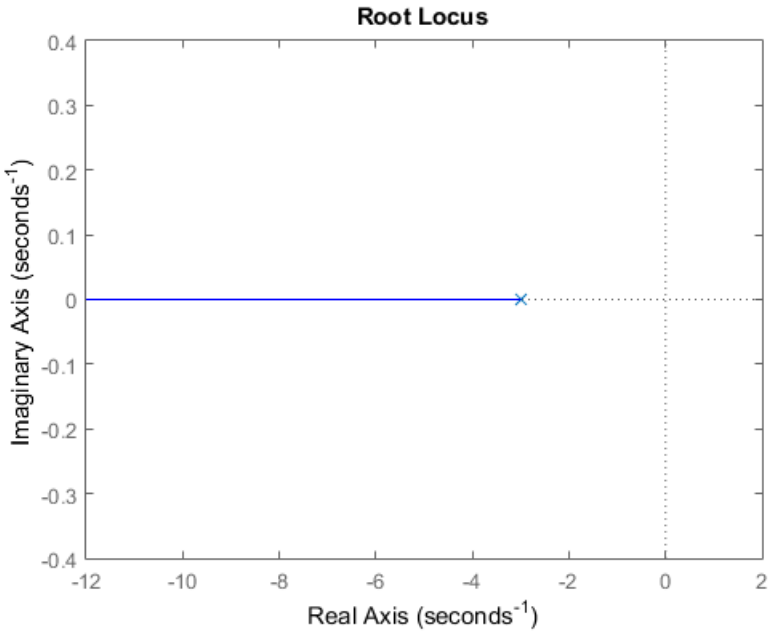
$$G(s) = \frac{K}{s+3}$$

The closed-loop transfer function is:

$$\frac{G(s)}{1+KG(s)} = \frac{K}{s+K+3}$$

So the characteristic equation is: $s + K + 3 = 0$ (1+KG(s)=0).
If we put K=0 we get a pole at -3, if we put K=1 we get a pole at -4, if we put K=9 we get a pole at -12 e.t.c.
We see that as the value of K is increased, the pole moves to the left, and this can be easily seen with MATLAB. So it is clear that the root locus begins from the closed-loop pole at -3 (with zero gain) and goes to minus infinity. (Note that MATLAB does not show the direction of the movement of the poles. It should be understood that the movement starts from the poles and ends to the zeros).

```
sys=tf([1],[1 3]);
[R,K]=rlocus(sys) % Returns the closed-loop pole locations for varied ...
    gain values
r=rlocus(sys,[0,3,9]) % Returns the closed-loop pole locations for gain ...
    values 0,3,9 respectively
rlocus(sys)  % Returns the root locus diagram
```



Example 10.2.2. Find the root locus plot for the transfer function:

$$G(s) = \frac{K}{(s+1)(s+3)}$$

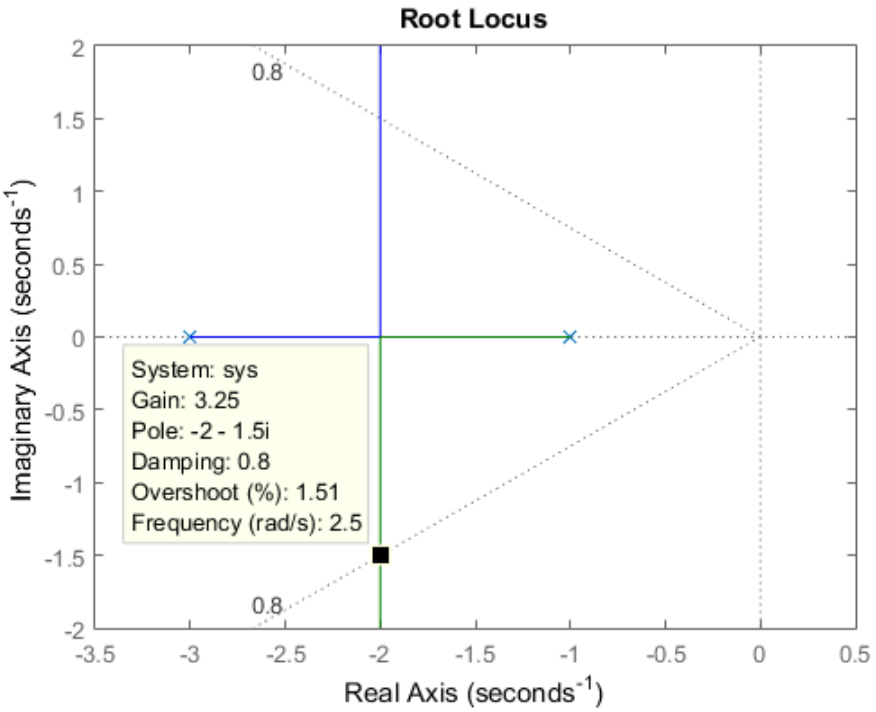
Find the gain value in order to have damping ratio=0.8, and find the closed-loop transfer function for that value.

Solution.

```
sys=tf([1],[1 4 3])
rlocus(sys)
sgrid(0.8,[]) % Plots a grid of constant damping factor lines for z=0.8
sys_cl=feedback(sys,3.25) % Returns the closed-loop tranfer function ...
    when gain is equal to 3.25
```

The solution can be found from the intersection points of the root locus and the damping ratio lines. Clicking at these points on the root locus, we obtain the corresponding gain,

pole locations, overshoot and frequency. We find that for a damping ratio of 0.8, the gain value should be 3,25.



Now we will see some basic rules in order to have a better understanding of the root locus plot:

•The number of branches of the root locus equals the number of closed-loop poles. At the previous example there are two branches, as there are two closed-loop poles {-1,-3}

•Root locus exists on the real axis to the left of an odd number of poles plus zeros. As we can see above, real-axis locus is between (-1) and (-3).

•The root locus is symmetrical about the real axis (because roots are either real or complex conjugate).

•The root locus departs (zero gain) from open-loop poles and arrives (infinite gain) at open-loop zeros. Because at the previous example there are no real axis poles, branches start at (-1) and (-3) and go to infinity.

•Root locus approaches straight lines as asymptotes as the locus approaches infinity. As we can see, at the previous example there is an asymptote at -2, which it is also a breakaway point.

10.3 Examples

Example 10.3.1. Given a negative unity feedback system with the following transfer function:

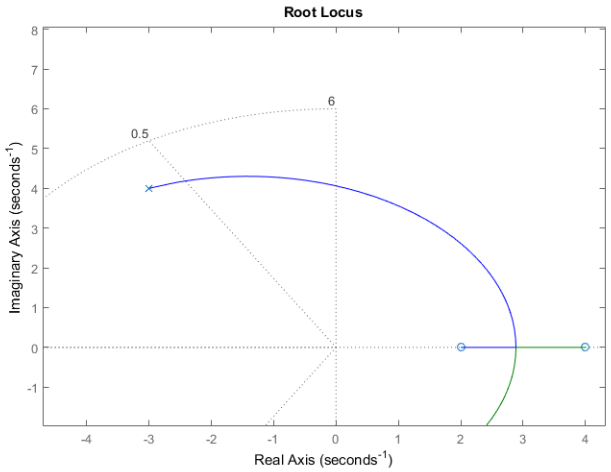
$$G(s) = \frac{K(s-2)(s-4)}{s^2+6s+25}$$

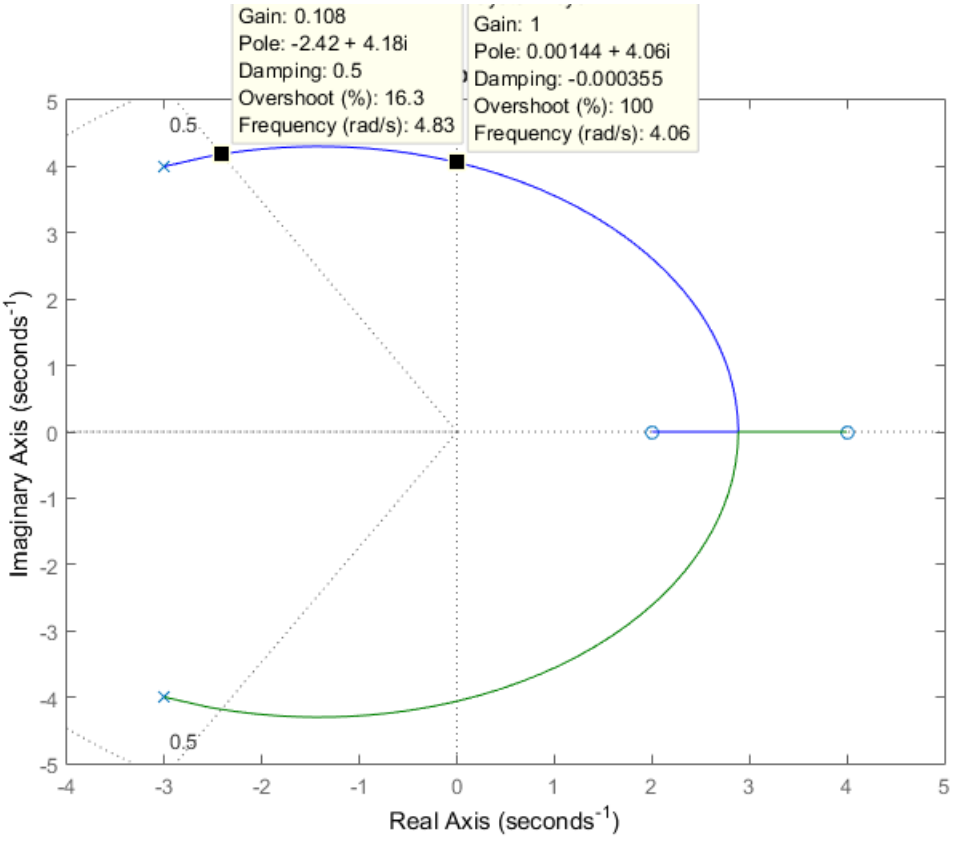
a. Find the root locus diagram.
b. For which gain value the closed-loop system becomes stable?
c. For which gain value the closed-loop system has poles with damping ratio 0.5? Find the closed-loop transfer function.
d. Find the gain value in order to get frequency value equal to 6.

Solution.

```
sys=zpk([2 4],[-3+4i -3-4i],1)
rlocus(sys)
sgrid(0.5,6) % Plots a grid of constant damping ratio lines(z=0.5) and ...
    frequency lines(w=6)
sys_cl=feedback(sys,0.108) % Returns the closed-loop transfer function ...
    when the gain value is 0.108
```

b. As we can see from the root locus diagram, the gain value for which the closed-loop system becomes stable is 1. (We click at the intersection points of the root locus and the imaginary axis.)
c. Now we click at the intersection points of the loot locus and the damping ratio lines, and we find that the gain value is 0,108.
d. There are no gain values which give frequency equal to 6. The reason is that there are no intersection points of the root locus and the frequency lines. This can be clearly seen from the pictures below.





Example 10.3.2. Given a negative unity feedback system with the following transfer function:

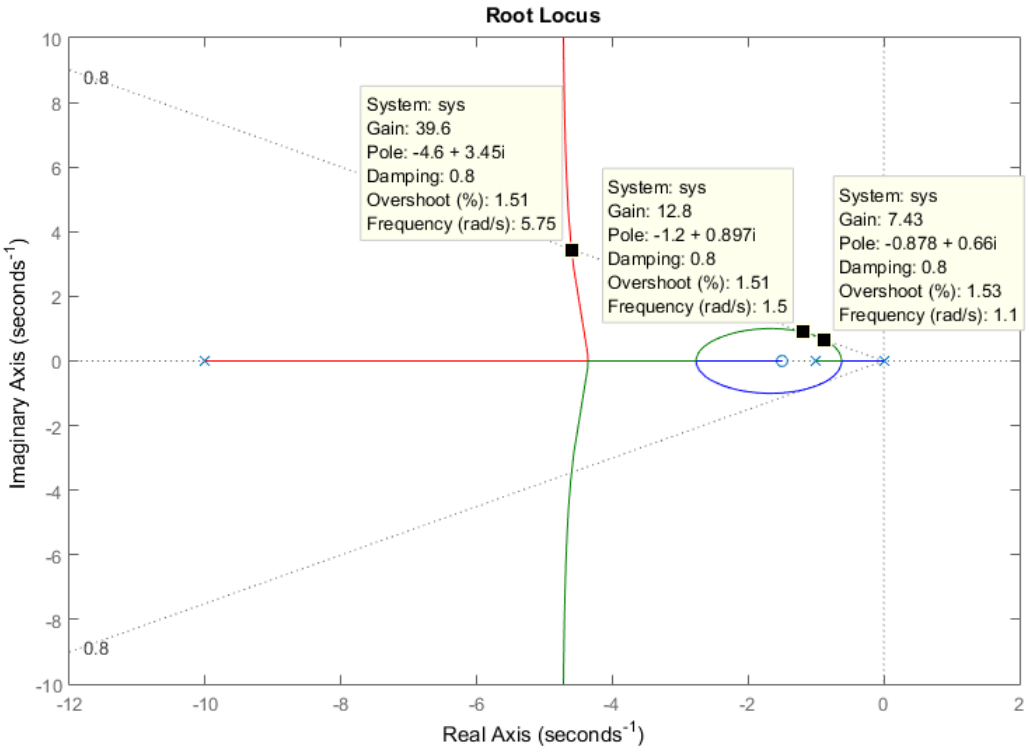
$$G(s) = \frac{K(s+1.5)}{s(s+1)(s+10)}$$

a. Find the root locus diagram.
b. Find the value of K for which the closed-loop system is stable.
c. Find the value of K for which the closed-loop system has overshoot 1.52%.

Solution.

```
sys=zpk([-1.5],[-1 -10 0],1)
rlocus(sys)
% Now we will calculate the damping ratio value, when overshoot is 1.52%
z=-log(1.52/100)/(sqrt(pi^2 +log(1.52/100)^2)) % We get that z=0.7998
sgrid(0.7998,[]) % Plots a grid of constant damping ratio lines(z=0.7998)
```

b. Root locus plot is at the left half plane, so the system is stable for all values of K.
c. Overshoot is 1.52% when damping ratio is $0,7998 \simeq 0,8$. We get the damping ratio lines and as we can see below, the gain values that we want are: 7,32 , 12,7 and 39,6.



Example 10.3.3. [Nise, 2013] A negative unity feedback system has the following transfer function:

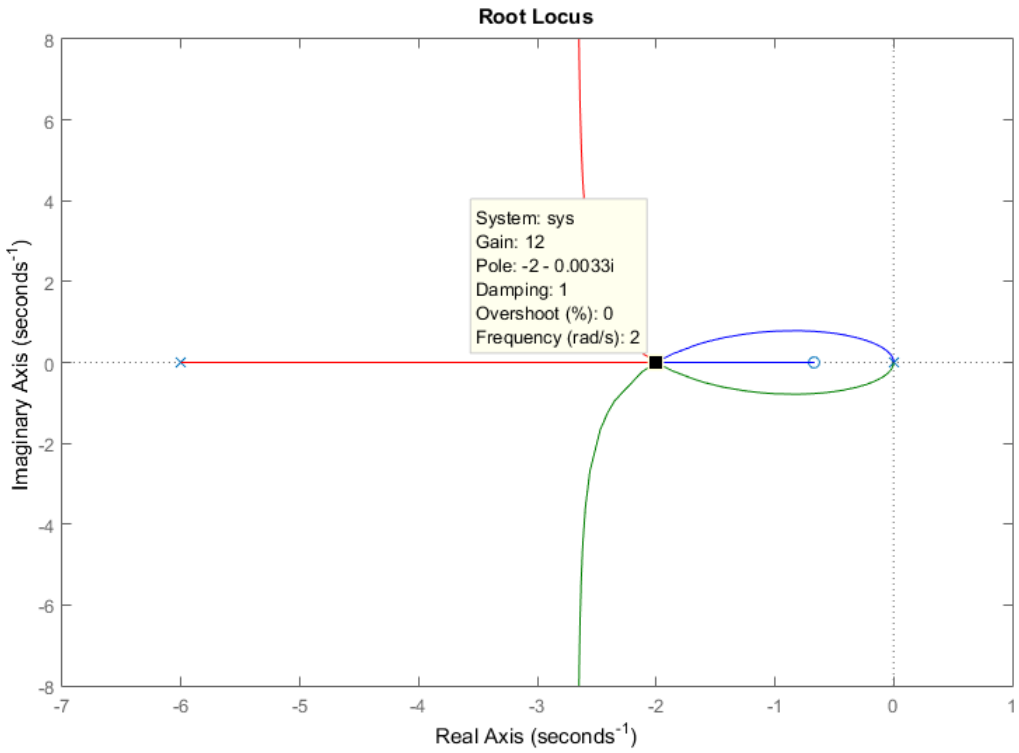
$$G(s) = \frac{K(s + \frac{2}{3})}{s^2(s + 6)}$$

a. Plot the root locus.
b. Find an expression for the closed-loop transfer function at the point where the three closed-loop poles meet.

Solution.

```
sys=zpk([-2/3],[0 0 -6],1)
rlocus(sys)
sys_cl=feedback(sys,12) % Returns the closed-loop transfer function ...
    when the gain value is 12
```

As we can see from the root locus diagram, the three closed-loop poles meet at the point where the gain value is 12.



Example 10.3.4. [Fadali, 2015] Find the root locus of the unity feedback system with transfer function:

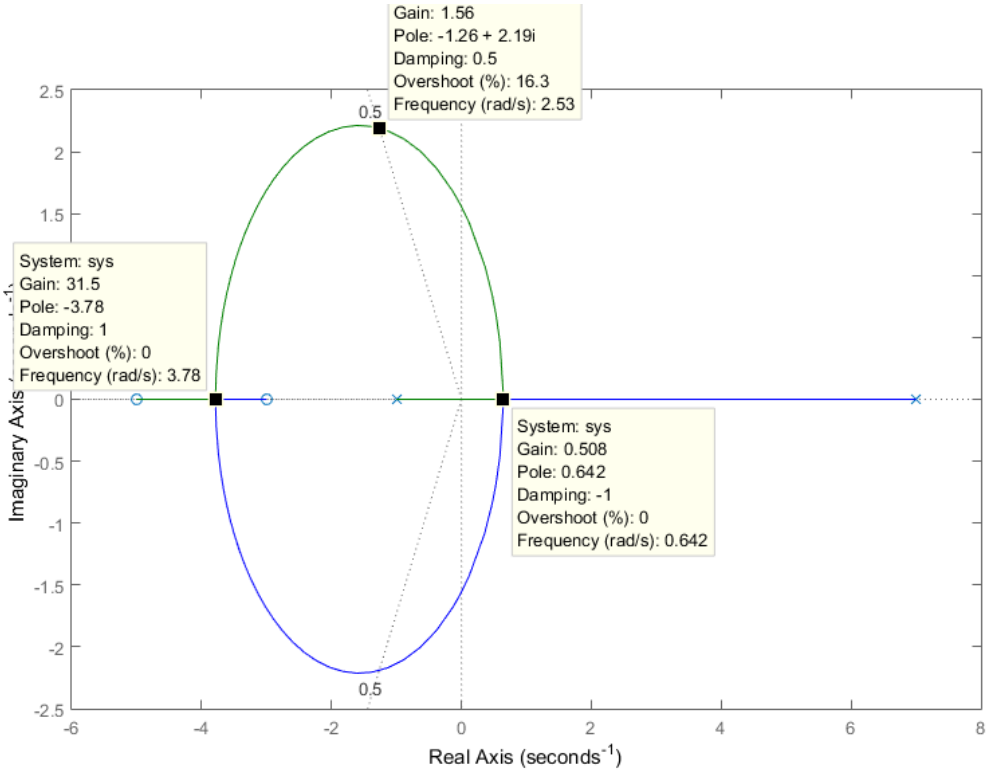
$$G(s) = \frac{K(s+3)(s+5)}{(s+1)(s-7)}$$

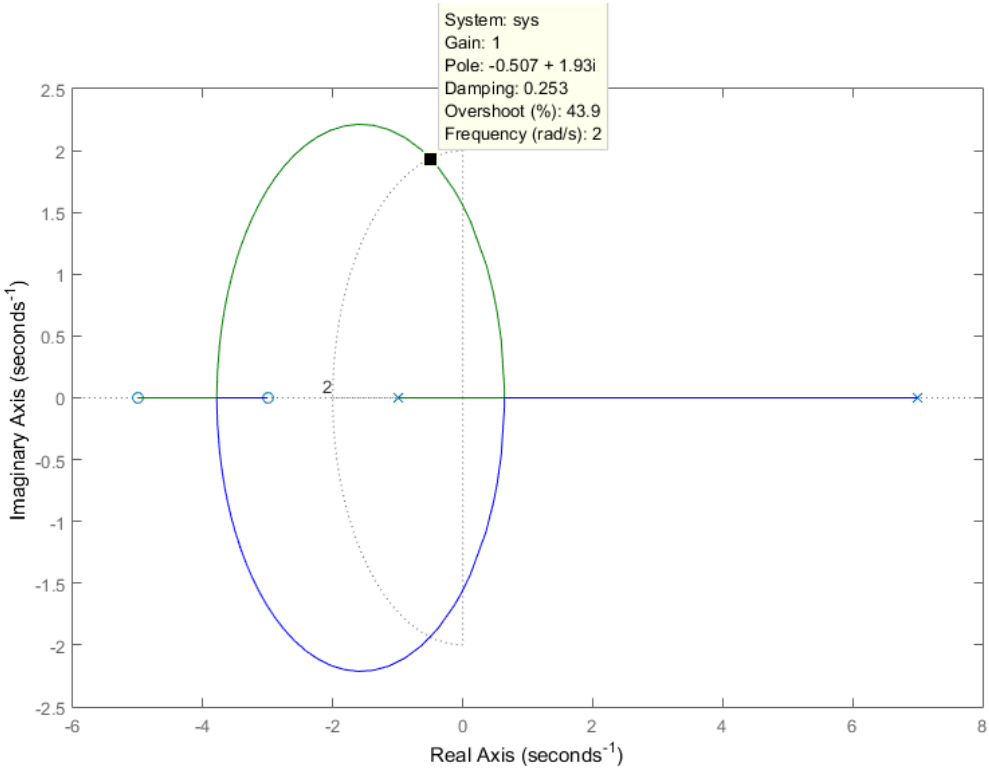
a. Find the gain at break-in and breakaway points.
b. Find the closed-loop transfer function when damping ratio value is 0.5
c. Find the closed-loop transfer function when frequency is equal to 2.

Solution.

```
sys=zpk([-3 -5],[-1 7],1)
rlocus(sys)
sgrid(0.5,[])
sys_cl=feedback(sys,1.56) % Returns the closed-loop transfer function ...
    when the gain value is 1.56
sgrid([],2)
sys_cl=feedback(sys,1)
```

Root locus starts from the two poles {-1,7} and ends at the two zeros {-3,-5}. So as we can see below, the gain at the breakaway point is 0,508 and at the break-in point is 31,5. Also we can see that when damping ratio is 0,5 the gain value is 1,56 and when frequency is 2, the gain value is 1.


Gain: 1.56
Pole: -1.26 + 2.19i
Damping: 0.5
Overshoot (%): 16.3
Frequency (rad/s): 2.53
System: sys
Gain: 31.5
Pole: -3.78
Damping: 1
Overshoot (%): 0
Frequency (rad/s): 3.78
System: sys
Gain: 0.508
Pole: 0.642
Damping: -1
Overshoot (%): 0
Frequency (rad/s): 0.642
0.5
0.5
Imaginary Axis
Real Axis (seconds-1)


System: sys
Gain: 1
Pole: -0.507 + 1.93i
Damping: 0.253
Overshoot (%): 43.9
Frequency (rad/s): 2
2
Imaginary Axis (seconds-1)
Real Axis (seconds-1)

Chapter 11

Nyquist Plot

11.1 Introduction

The Nyquist stability criterion, created by Swedish-American electrical engineer Harry Nyquist at Bell Telephone Laboratories in 1932, is a graphical technique for determining the stability of a dynamical system. This criterion relates the stability of a closed system to the open-loop frequency response and open loop pole location. It can tell us how many closed-loop poles are in the right half-plane, thus helping determine whether a system is stable or not.

11.2 Nyquist Criterion

A Nyquist diagram is a polar plot of the frequency response function $G(i\omega)$ in the complex plane, for all real values of ω. It's a plot of the transfer function, G(s) with $s = i\omega$. The complex number $G(i\omega)$, depends upon frequency, so frequency will be the free parameter when plotting the imaginary part of $G(i\omega)$ against the real part of $G(i\omega)$. We can easily get the Nyquist plot in Matlab, using the command `nyquist`.

`nyquist(sys)`	Creates the Nyquist plot of the dynamic system `sys`, as the frequency takes values from zero to infinity. The model can be continuous or discrete.
`nyquist(sys,w)`	Specifies the frequency range or frequency points to be used for the plot. To focus on a particular frequency interval, set $w = [w_{min}, w_{max}]$. To use particular frequency points, set `w` to the vector of desired frequencies.
`nyquist(sys1,sys2,..,sysk)`	Plots the Nyquist diagrams of several models on a single figure. All systems must have the same number of inputs and outputs, but may otherwise be a mix of continuous and discrete-time systems. We can also specify a distinctive color, linestyle, or marker for each system plot with the syntax `nyquist(sys1,'PlotStyle1',..)`.
`[re,im]=nyquist(sys,w)`	Returns the real and imaginary parts of the frequency response, at the specified frequency `w`.

An example of a Nyquist plot will illustrate what a Nyquist plot is. We will take a very simple system:

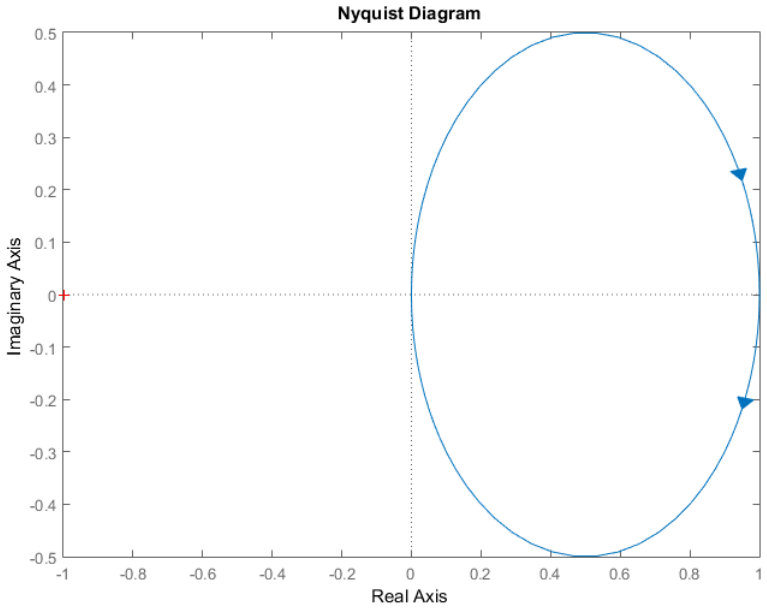
$$G(s) = \frac{1}{s+1}.$$

If we substitute s = iw, we get $G(iw) = \frac{1}{iw+1}$, and that's a complex number. We will compute the real and imaginary parts of G(iw) by converting the denominator to a real number: $G(iw) = \frac{1}{(iw+1)}\frac{1-iw}{(1-iw)} = \frac{1-iw}{1+w^2}$.
The real and imaginary parts of the frequency response function are:

$$Re(G(iw)) = \frac{1}{1+w^2} \qquad Im(G(iw)) = \frac{-w}{1+w^2}$$

The Nyquist diagram starts at the point (1,0) at an angle of 0 degrees. As w varies between zero and infinity, we get the Nyquist plot as shown below, in the direction of increasing w.

```
sys=tf([1],[1 1])
nyquist(sys)% Plots the Nyquist diagram
[re,im]=nyquist(sys,2) % Returns the real and imaginary parts of the ...
   frequency response, when w=2. The real part is equal to 0.2 and the ...
   imaginary part is -0.4
```



Consider now the following negative feedback system:

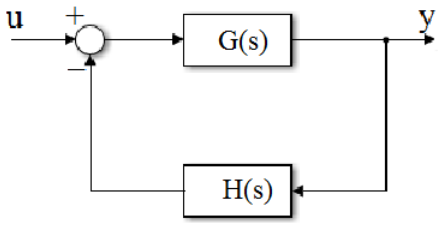
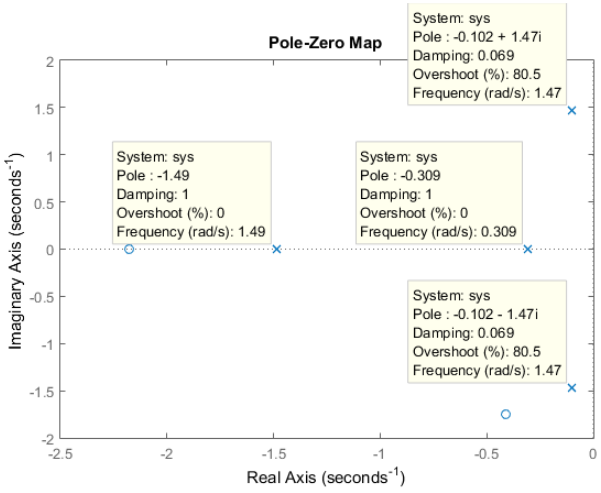


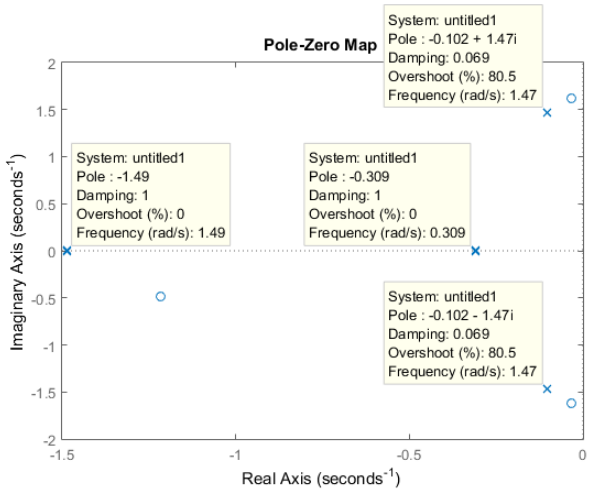
Figure 1

We know that the poles of the open-loop system G(s)H(s) are also the poles of 1+G(s)H(s). We can take a random system and easily see that in MATLAB.

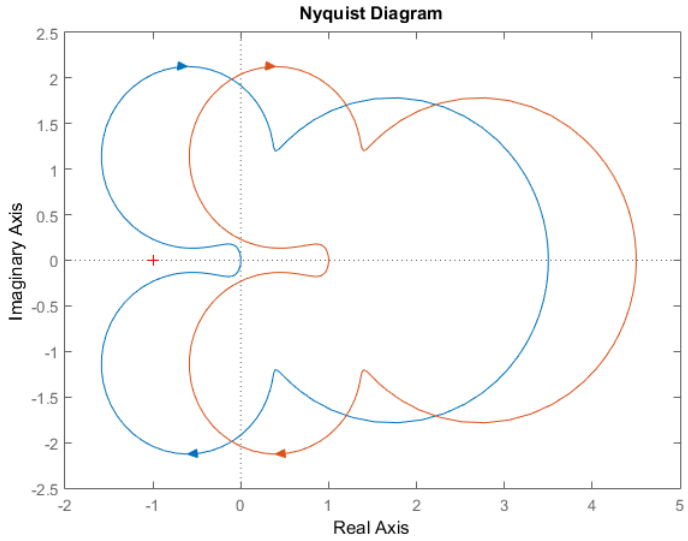
```
sys=tf([1 3 5 7],[2 4 6 8 2]); % We take a random open-loop transfer ...
    function
pzmap(sys) % Map the poles and zeros for the OL (open-loop) system
figure; pzmap(sys+1) % Map the poles and zeros for the 1+OL system
close all
nyquist(sys,sys+1) % Plots the Nyquist diagram for both systems
```



poles and zeros for the OL system

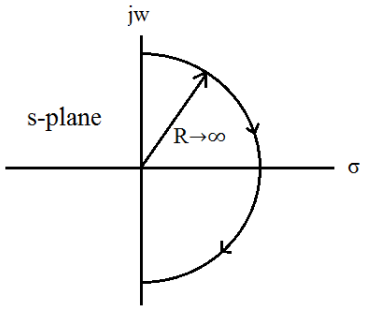


poles and zeros for the 1+OL system



The Cauchy criterion (from complex analysis) states that when taking a closed contour in the complex plane s, and mapping it through a complex function F(s), the number of times that the plot of F(s) encircles the origin in the (Re{F(s)},Im{F(s)})-plane, is given by: $N = Z - P$, where Z and P stand for the number of zeros and poles of the function F(s) inside the contour. Encirclements of the origin are counted as positive if they are in the same direction as the original closed contour or negative if they are in the opposite direction.

The Nyquist stability test is obtained by applying the Cauchy criterion to the complex function $1 + G(s)H(s)$. If we extend the contour to include the entire right half-plane, as shown below, we can count the number of right half plane, closed-loop poles and determine a system's stability.



The poles of $1+G(s)H(s)$ are the same as the poles of the open-loop system $G(s)H(s)$ which are known, and the zeros of $1 + G(s)H(s)$ are also the poles of the closed-loop system $\dfrac{G(s)}{1 + G(s)H(s)}$ which are not known. Since all of the poles and zeros of $G(s)H(s)$ are known, what if we map through $G(s)H(s)$ instead of $1 + G(s)H(s)$? The resulting contour is the same as a mapping through $1 + G(s)H(s)$, except that it is translated one unit to the left. (We can see the Nyquist plots that we got above, with MATLAB). So we count rotations about -1 instead of rotations about the origin.

Hence, the **Nyquist criterion** states that: $Z = P + N$, where:
P = the number of open-loop poles of G(s)H(s) in the right half-plane,
N = the number of times the Nyquist diagram encircles -1. Clockwise (counter-clockwise) encirclements of -1 count as positive (negative) encirclements.
Z = the number of poles of the closed-loop system in the right half-plane.
So to have a stable system, we should get Z=0.

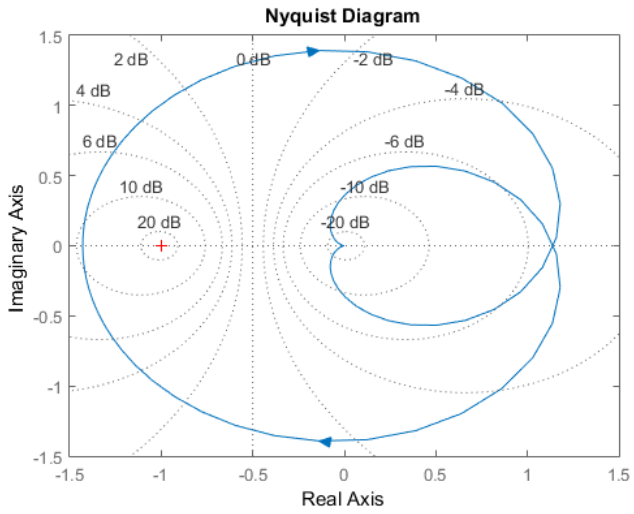
Example 11.2.1. Apply the Nyquist criterion to determine the stability of the following unit-feedback system:

$$G(s) = \frac{500(s-2)}{(s+2)(s+7)(s+50)}$$

Solution.

```
sys=zpk([2],[-2 -7 -50],500)
nyquist(sys)
grid on
```

We have P = 0 (no open-loop poles in the right-half plane), but N=1, so the system is unstable with one closed loop pole in the right half plane. (Z=P+N=1)



If the open-loop system contains a variable gain K, we set K=1 and sketch the Nyquist diagram. We consider the critical point to be at $-\frac{1}{K}$ rather than at -1, and then we adjust the value of K to yield stability, based upon the Nyquist criterion.

Using MATLAB is very easier to determine stability. Using the Nyquist diagram, we define two quantitative measures of how stable a system is. These quantities are called gain margin and phase margin. The gain margin is the change in open-loop gain, expressed in decibels (dB), required at 180 degrees of phase shift to make the closed-loop system unstable. The phase margin is the change in open-loop phase shift required at unity gain to make the closed-loop system unstable. In MATLAB we can find their values using the command `allmargin`. We can also find them from the Nyquist diagram. Right-click in the graph area, select Characteristics and then select All Stability Margins. Let the mouse rest on the margin points to read the gain and phase margins. Gain margin is expressed in decibels, so we convert it using `db2mag`.

`allmargin(sys)`	Computes the gain margin, phase margin, delay margin and the corresponding crossover frequencies of the SISO open-loop model `sys`.
`db2mag(ydb)`	Returns the corresponding magnitude `y` for a given decibel (dB) value `ydb`. The relationship between magnitude and decibels is $\texttt{ydb} = 20 \cdot log_{10}(\texttt{y})$.

11.3 Examples

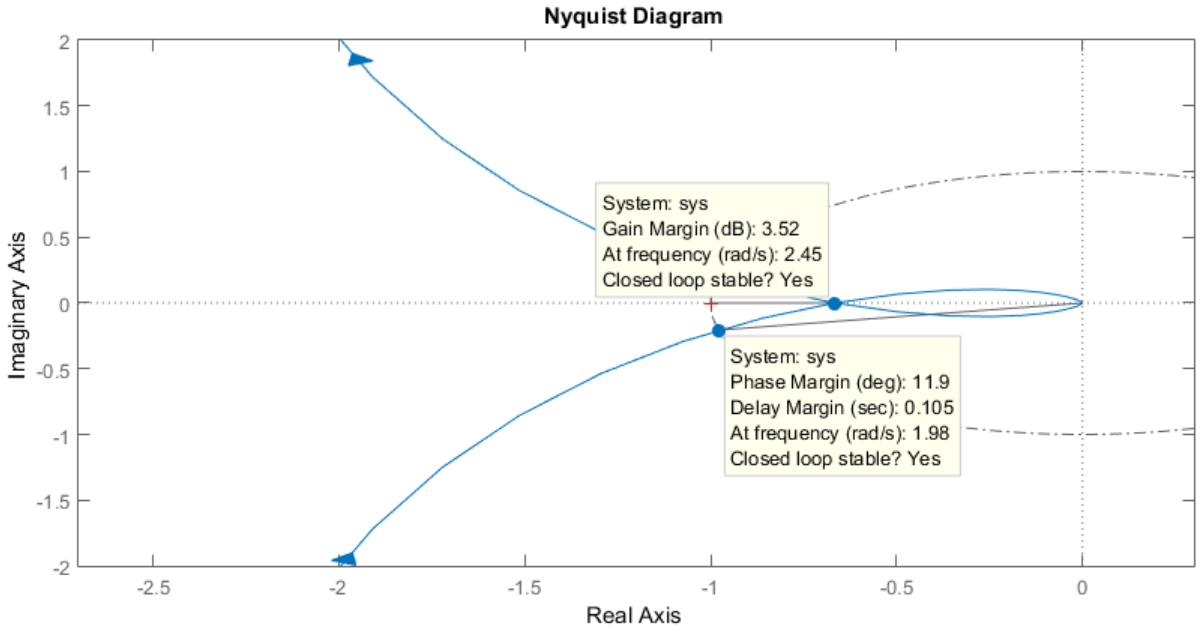
Example 11.3.1. Check the stability of the following unit-feedback system:

$$L(s) = \frac{20}{s(s+3)(s+2)}$$

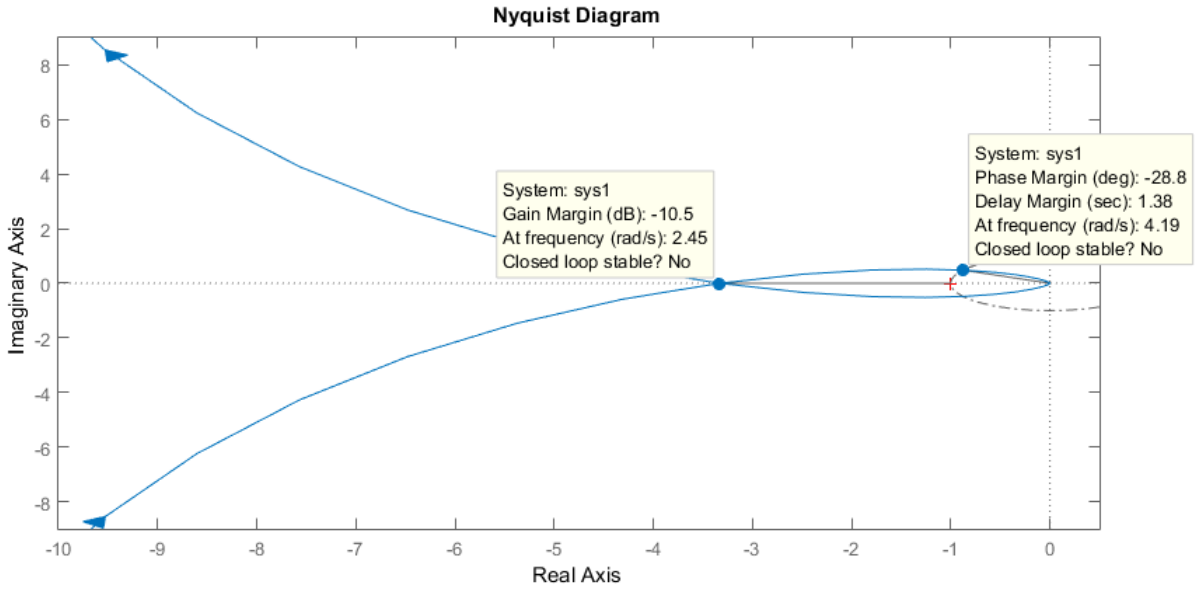
Solution.

```
sys=zpk([],[0 -3 -2],[20])
nyquist(sys)
db2mag(3.52) % Converts the gain margin from decibels, we find that ...
    it's 1.5
sys1=zpk([],[0 -3 -2],[100])
nyquist(sys1)
```

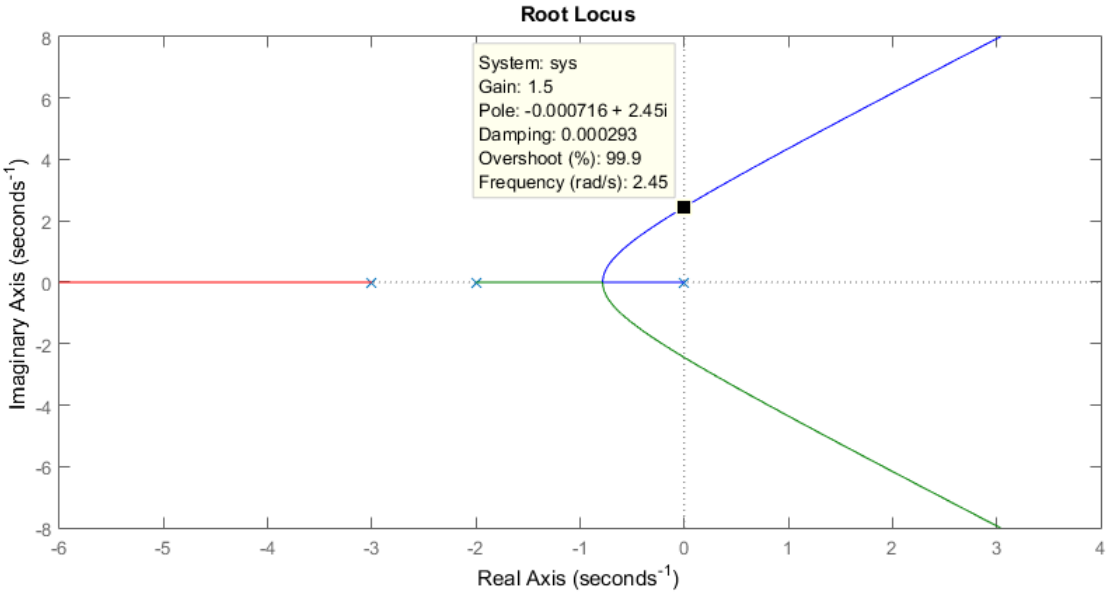
This Nyquist Diagram is a little hard to decipher, so we change the axes limits in order to zoom in. The -1+i0 point is not encircled, so N=0. L(s) has no poles in the right half plane, so P=0. Since Z=P+N=0, the closed loop system has no poles in the right half plane, so the system is stable.



Since the gain margin is 3.52dB (=1.5), this tells us that we could increase the gain by up to a factor of 1.5 before the system goes unstable. Let's test this. If we multiply L(s) by 5, we get: $L_1(s) = \dfrac{100}{s(s+3)(s+2)}$.
We get the Nyquist plot shown below, which has negative gain and phase margins. Now there is one clockwise encirclement of -1, so N=1. Z=0+1, so the system is indeed unstable.



We can verify the result by checking the root locus diagram:

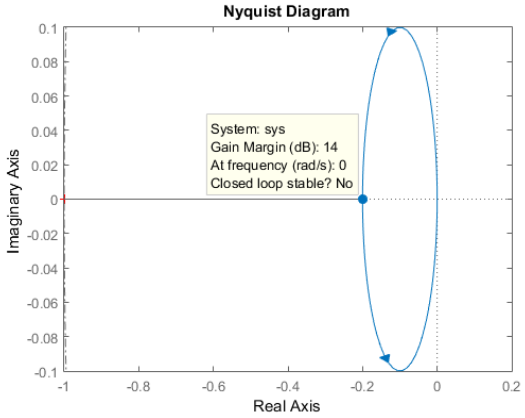


Example 11.3.2. Find the range of K for stability of the following unit-feedback system:

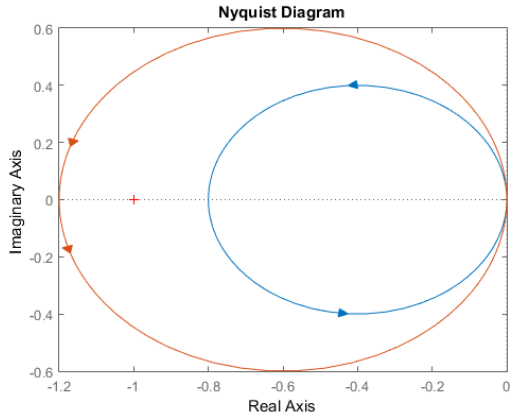
$$G(s) = \frac{K}{s-5}.$$

Solution.

```
sys=zpk([],[5],1)
nyquist(sys)
allmargin(sys) % We get that the gain margin is 5
db2mag(14) % or if we convert the gain margin seen on the diagram from ...
    decibels, we can see that it's 5
sys1=zpk([],[5],4)
sys2=zpk([],[5],6)
nyquist(sys1,sys2)
```



The gain margin is 5, so for $0 < K < 5$ the system is unstable, and for marginal stability, K=5. We can see this by setting K=4 and K=6. When K=4 (blue diagram), the Nyquist plot does not encircle the critical point -1+i0, so N=0. The open loop transfer function has one unstable pole, so P = 1. Z=P+N=1, so the closed loop system has one unstable pole. When K=6 (red diagram), the Nyquist plot makes one circle around the critical point $-1 + j0$ in the anti-clockwise direction (N=-1). Thus, the closed loop system is unstable for K = 4, and stable for K=6.



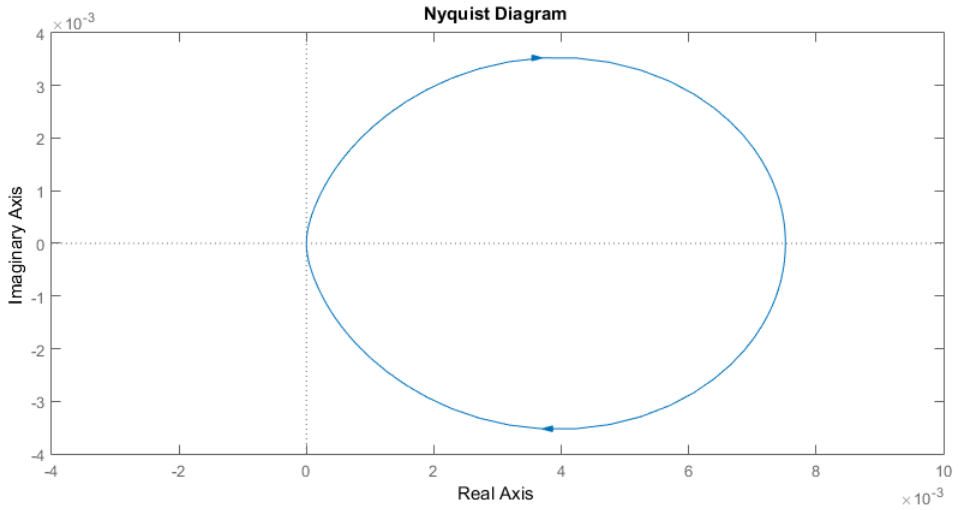
Example 11.3.3. [Nise, 2013, Thomas et al., 2005] A room's temperature can be controlled by varying the radiator power. In a specific room, the transfer function from indoor radiator power, $\dot{Q}$, to room temperature, T in ℃, is

$$P(s) = \frac{T(s)}{Q(s)} = \frac{(1 \times 10^{-6})s^2 + (1.314 \times 10^{-9})s + (2.66 \times 10^{-13})}{s^3 + 0.00163s^2 + (5.272 \times 10^{-7})s + (3.538 \times 10^{-11})}$$

The system is controlled in a negative feedback closed-loop configuration with G(s)=KP(s).
a. Draw the corresponding Nyquist diagram for K=1.
b. Obtain the gain and phase margins.
c. Find the range of K for the closed-loop stability.

Solution.

```
sys=tf([1*10^(-6) 1.314*10^(-9) 2.66*10^(-13)],[1 0.00163 5.272*10^(-7) ...
    3.538*10^(-11)])
nyquist(sys)% Plots the Nyquist diagram
```



The gain margin is infinite as the plot never crosses the 180^o line. The phase margin is undefined since $|G(jw)| < 1$ at all frequencies, and the system is closed loop stable for all $0 < K < \infty$.

Example 11.3.4. [Nise, 2013, Kim et al., 2009] An experimental holographic media storage system uses a flexible photopolymer disk. During rotation, the disk tilts, making information retrieval difficult. A system that compensates for the tilt has been developed. For this, a laser beam is focused on the disk surface and disk variations are measured

through reflection. A mirror is in turn adjusted to align with the disk and makes information retrieval possible. The system can be represented by a unity feedback system in which a controller with transfer function:

$$G_C(s) = \frac{78.575(s+436)^2}{(s+132)(s+8030)}$$

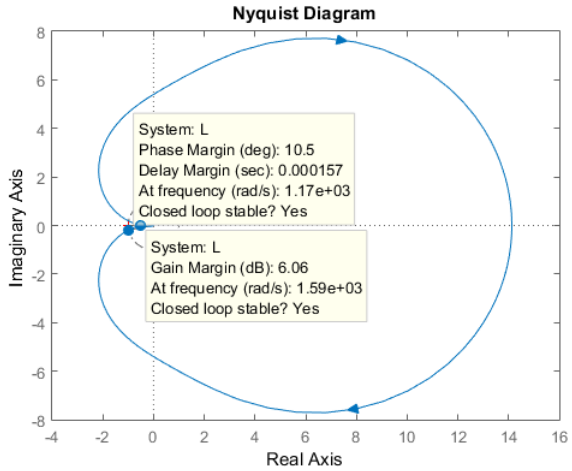
and a plant

$$P(s) = \frac{1.163 \times 10^8}{s^3 + 962.5s^2 + 5.958 \times 10^5 s + 1.16 \times 10^8}$$

from an open loop transmission $L(s) = G_C(s)P(s)$.
a. Use MATLAB to obtain the system's Nyquist diagram. Find out if the system is stable.
b. Find the system's phase and gain margin.

Solution.

```
s=tf('s');
P=1.163e8/(s^3+962.5*s^2+5.958e5*s+1.16e8);
G=78.575*(s+436)^2/(s+132)/(s+8030);
L=G*P;
nyquist(L)
```



The system is closed-loop stable and it can be seen from the plot that the phase margin is 10.5 degrees. Also the gain margin is 6.06dB.

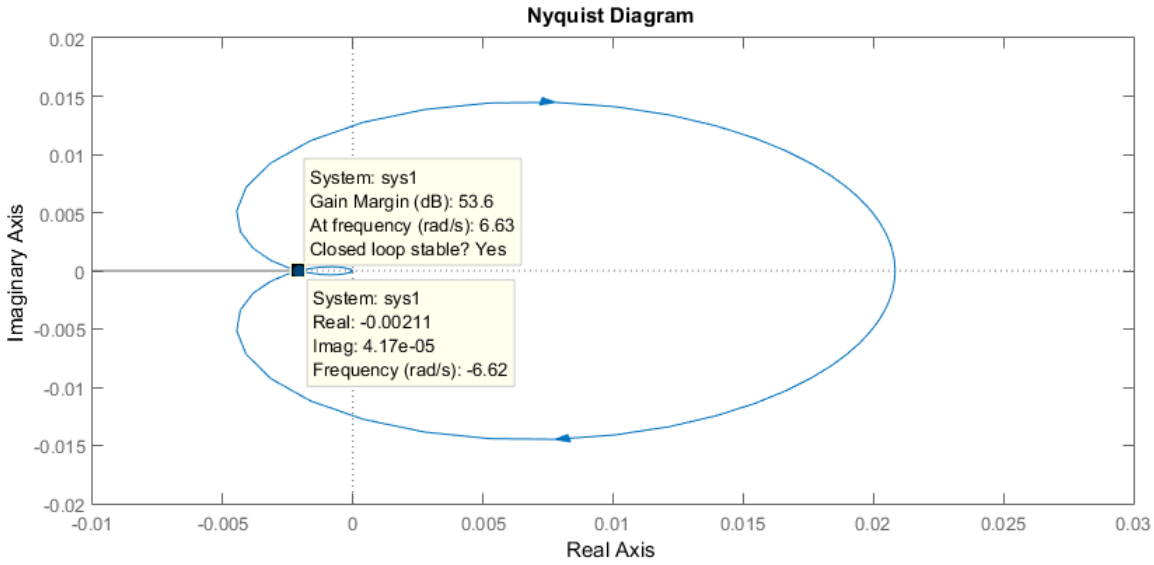
Example 11.3.5. [Nise, 2013] Find the range of K for stability, for each of the following systems that are controlled in a feedback closed-loop configuration with:

1. $G(s) = \dfrac{K}{(s+2)}$ and feedback gain $H(s) = \dfrac{1}{(s+4)(s+6)}$.

2. $G(s) = \dfrac{K(s^2 - 4s + 13)}{(s+2)(s+4)}$, and feedback gain $H(s) = \dfrac{1}{s}$.

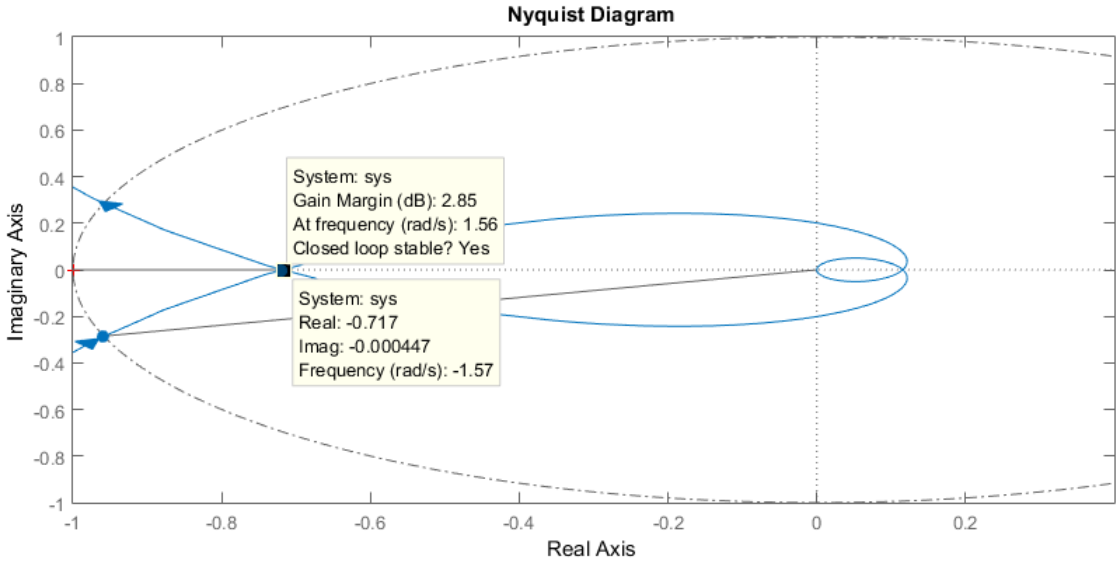
Solution.

```
sys1=zpk([],[-2 -4 -6],1)
nyquist(sys1)
db2mag(53.6) % Converts the gain margin from decibels, we find that ...
    it's 478.63
```



The Nyquist diagram intersects the real axis at -0.0021. The open-loop transfer function has no unstable poles, so P=0. Hence, Z=P+N, Z=0+0 if $K < 478.63$, and Z=0+2 if $K > 478.63$. Therefore, we have stability if $0 < K < 478.63$.
b.

```
p=tf([1 -4 13],[1 0]);
q=zpk([],[-2 -4],1);
sys=p*q
nyquist(sys)
db2mag(2.85) % Converts the gain margin from decibels, we can see that ...
    it's 1.39
```



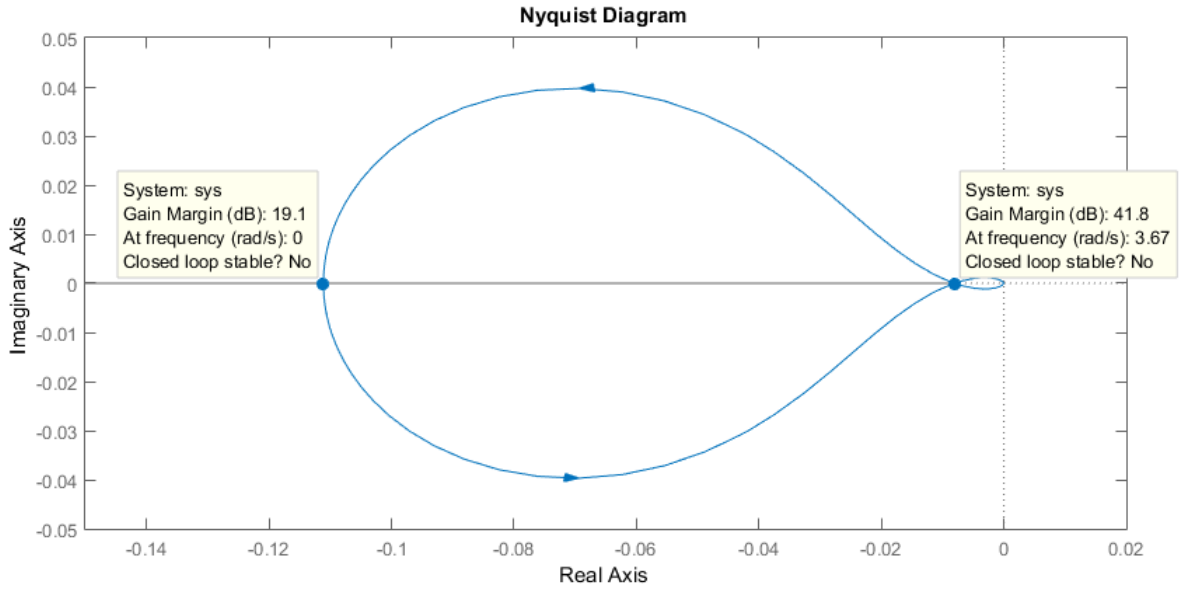
The Nyquist diagram intersects the real axis at -0.72 and P = 0. Hence, Z = P + N, Z=0+0 if $K < 1.39$, and Z=0+2 if $K > 1.39$. Therefore, stability if $0 < K < 1.39$.

Example 11.3.6. Find the range of K for stability of the following unit-feedback system:

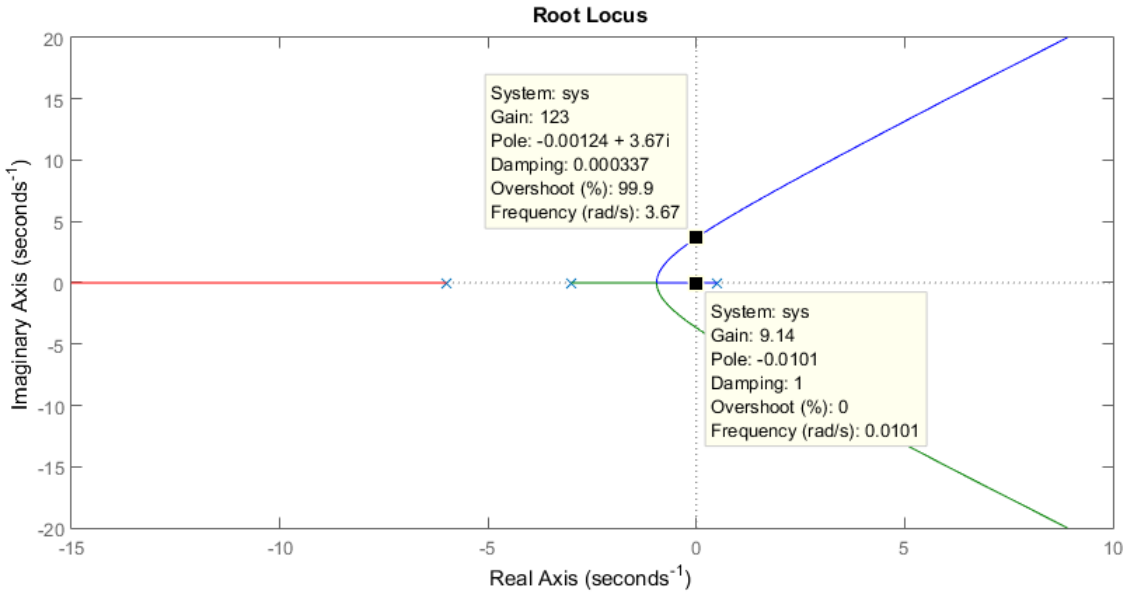
$$\frac{K}{(s-0.5)(s+3)(s+6)}$$

Solution.

```
sys=zpk([],[0.5 -3 -6],1)
nyquist(sys)
allmargin(sys) % The gain margin is the interval [9 123.7198]
```



The gain margin here is an interval, because as we can see from the root locus plot, the system from unstable becomes stable at gain 9, and then it becomes unstable again.



Chapter 12

Control Systems Toolbox

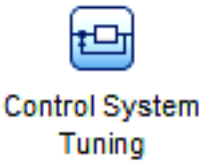
12.1 Introduction

In this chapter a useful Matlab® application called the Control Systems Toolbox® will be presented. This Toolbox allows us to define complicated systems (as in Simulink®) and by interactively modifying the systems parameters, observe the changes made in the system's response, root locus, Nyquist and Bode diagrams.

So when using this Toolbox, we can input our desired plant, either in a transfer function or a state space form, choose the systems structure from a predefined set of available options and observe the systems behavior in the time (step/ramp responses, root locus) or frequency (Nyquist and Bode plots) domain. These functionalities will be explicitly presented in the following sections.

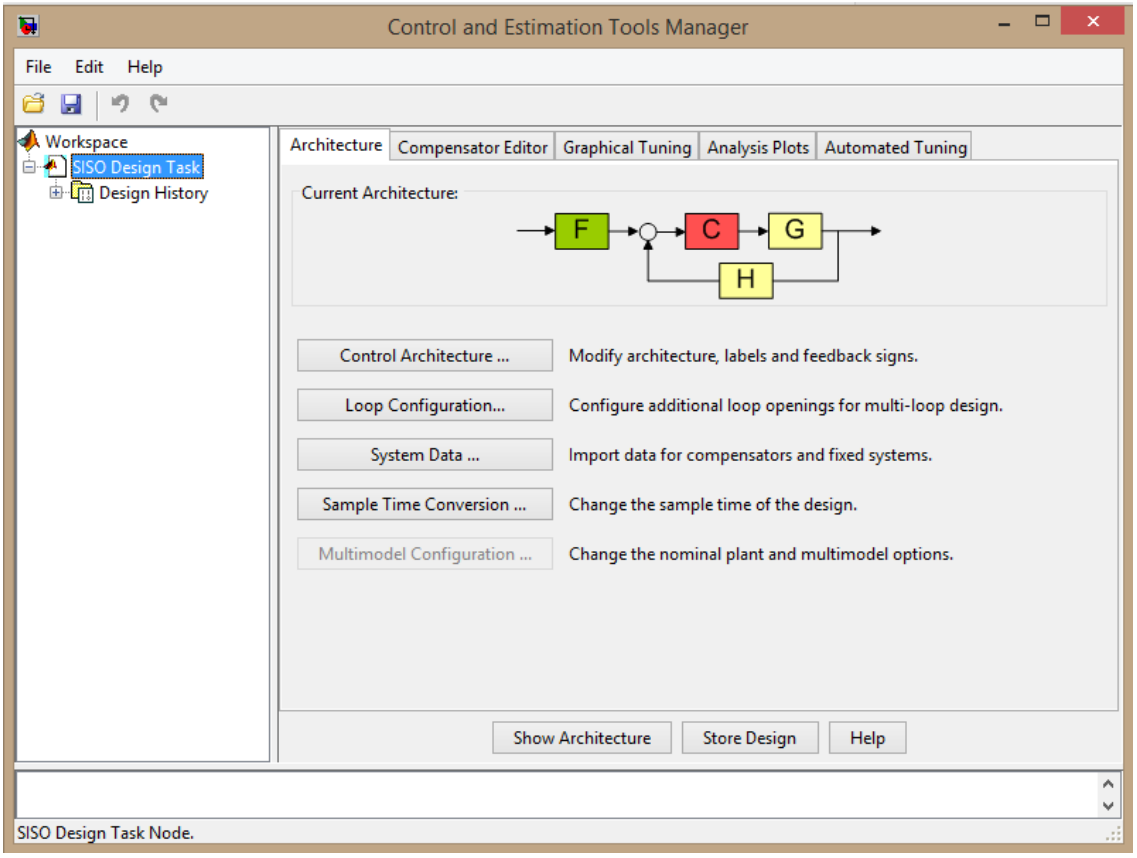
12.2 Designing the system

First of all, in order to open up the Control Systems Toolbox®, type `sisotool` in the command window. Alternatively, you can find the Toolbox from Matlab's® *APPS* tab.



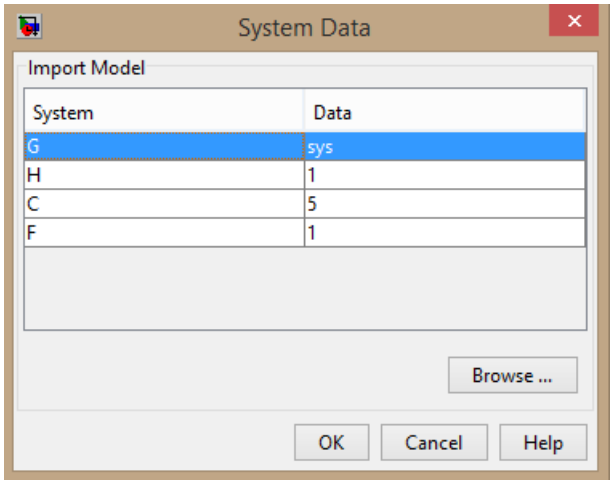
A window will pop up looking like this: [1]

[1] An additional window with blank plots may appear too. Ignore this for now.



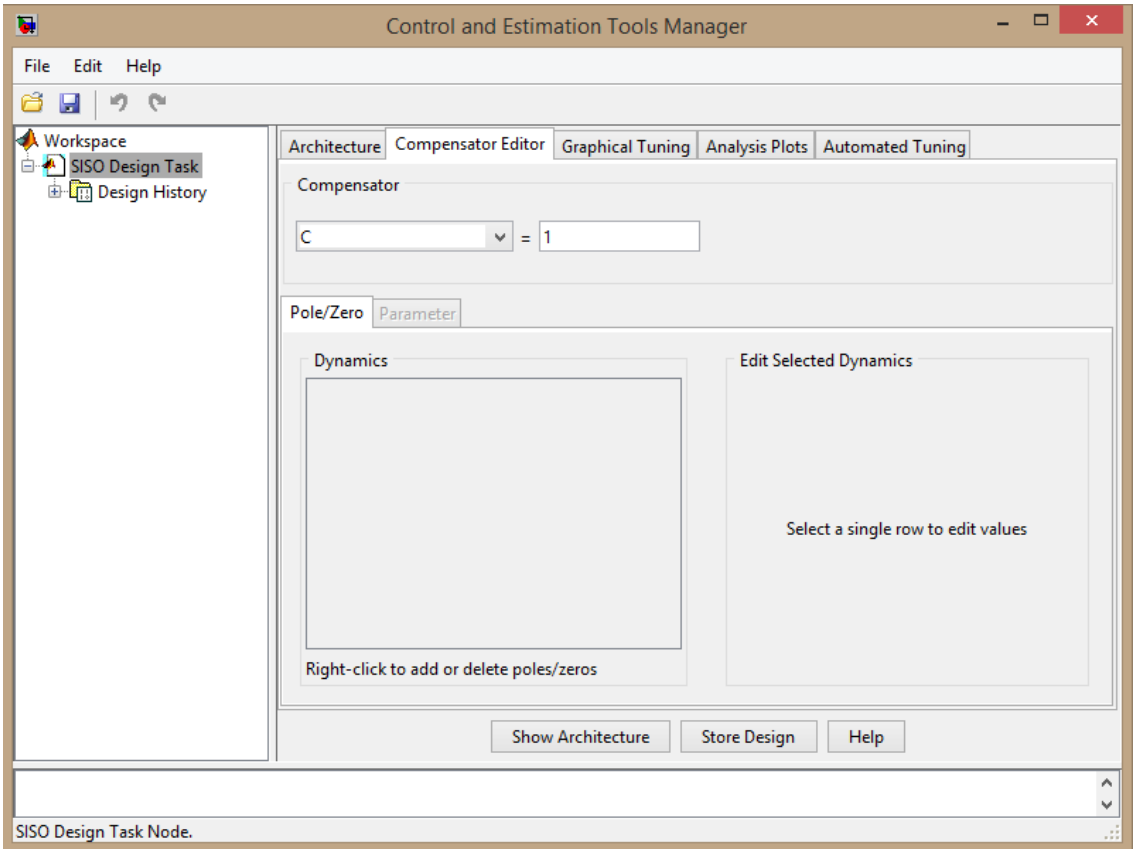
This is the main interface of the Toolbox. It contains five tabs, namely *Architecture, Compensator Editor, Graphical Tuning, Analysis Plots* and *Automated Tuning*. We will focus on the main aspects of each tab.

On Architecture, one can choose the basic structure of the system and import its plant. On *Control Architecture* one can choose between six different system structures. On *System Data*, one can choose the values of each block F, C, G, H. It is here that we can import the plant for the system. By clicking on this option, a new window pops up



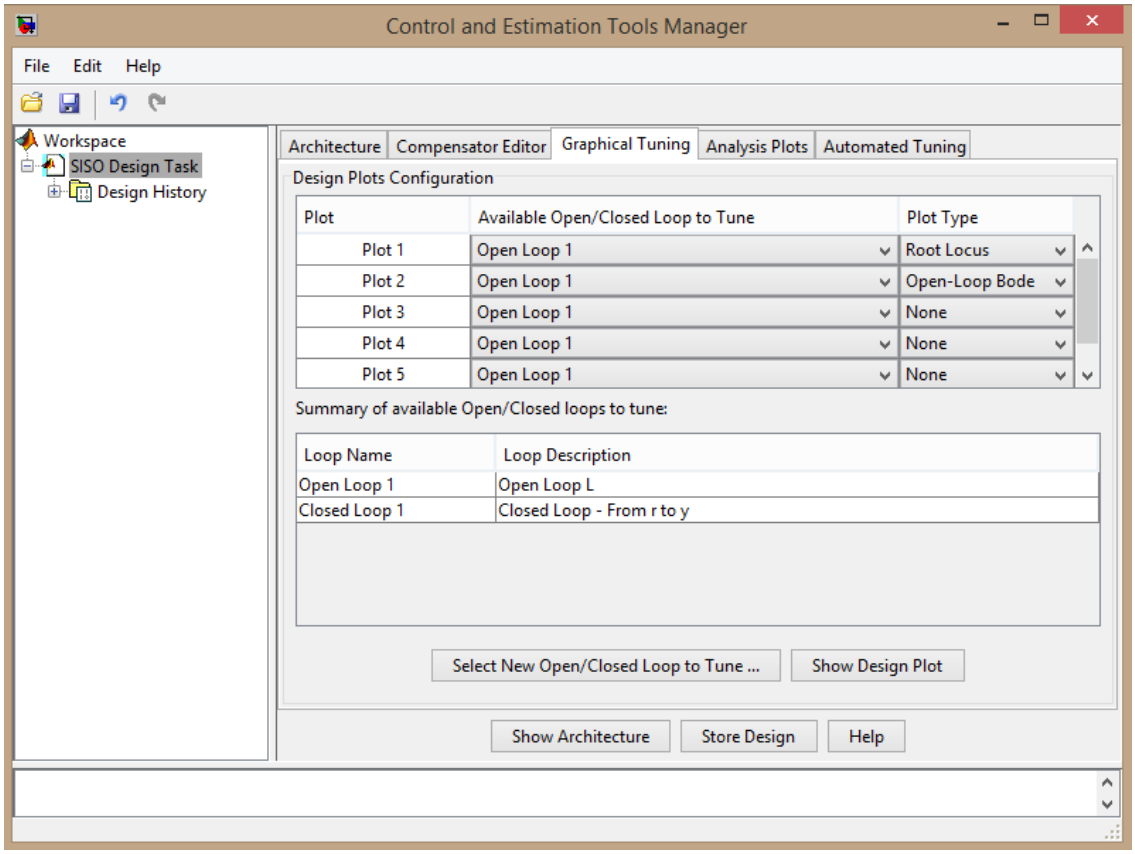
Here we can import the desired plant that we should have previously defined in our workspace. So here it should be understandable that there exists a system variable called sys in the workspace. One can also directly input numerical values for the blocks, but it is preferable to do so from the Compensator tab.

On Compensator Editor, one can edit the compensator blocks of the system. You can change the gain value and add or remove poles and zeros by right clicking on the blank space above *Pole/Zero*. The values of these parameters though can also change interactively by "moving along"the root locus, as we shall see later.



On Graphical Tuning, one can choose among three different plots: Root locus, open loop Bode plot, and Nichols plot. Once the plots have been chosen, a new window will pop up displaying the user's choices. Root locus and Bode are the most usual tools in system analysis.

Here lies the strongest advantage of the Control Systems Toolbox®, which we will explain in the following examples. This is the fact that we can interactively change the value of the compensator gain C by placing the poles on the desired positions in the root locus, or by dragging the magnitude curve of the Bode plot up or down. In addition to this, one can input design constraints in these plots, which help in the system design.

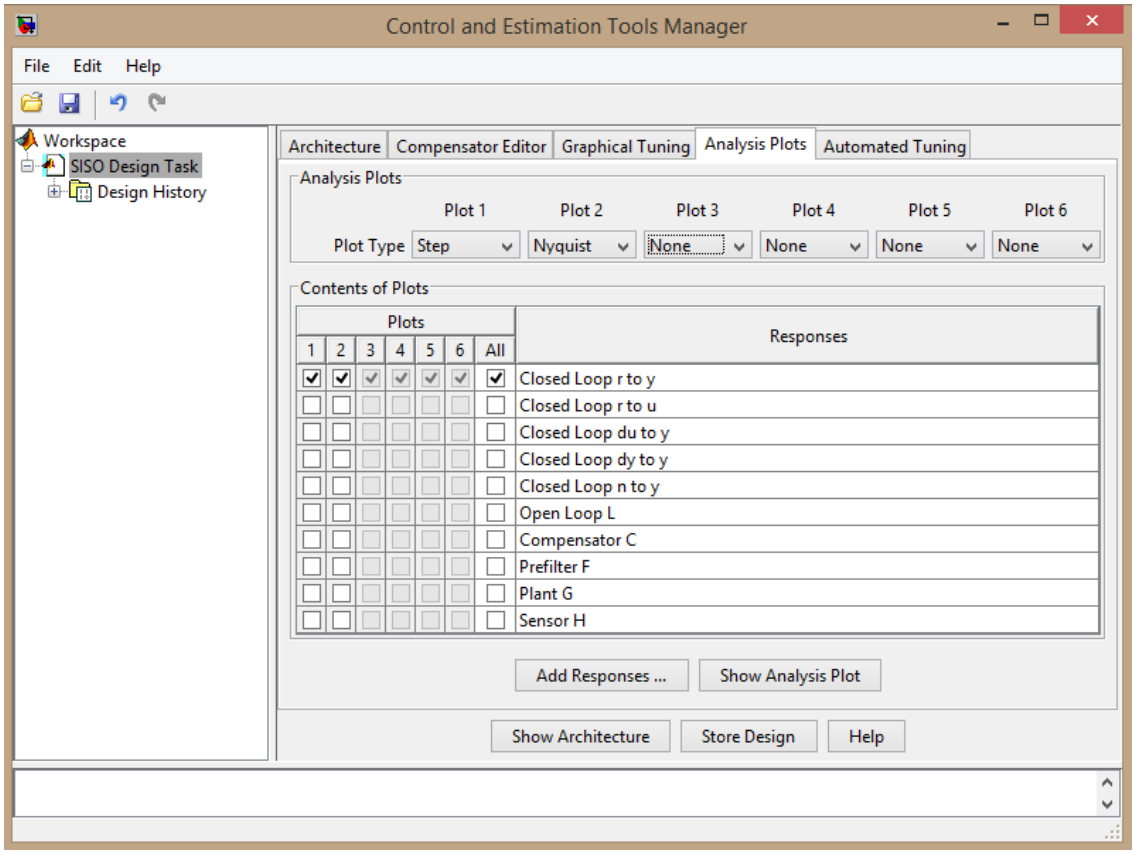


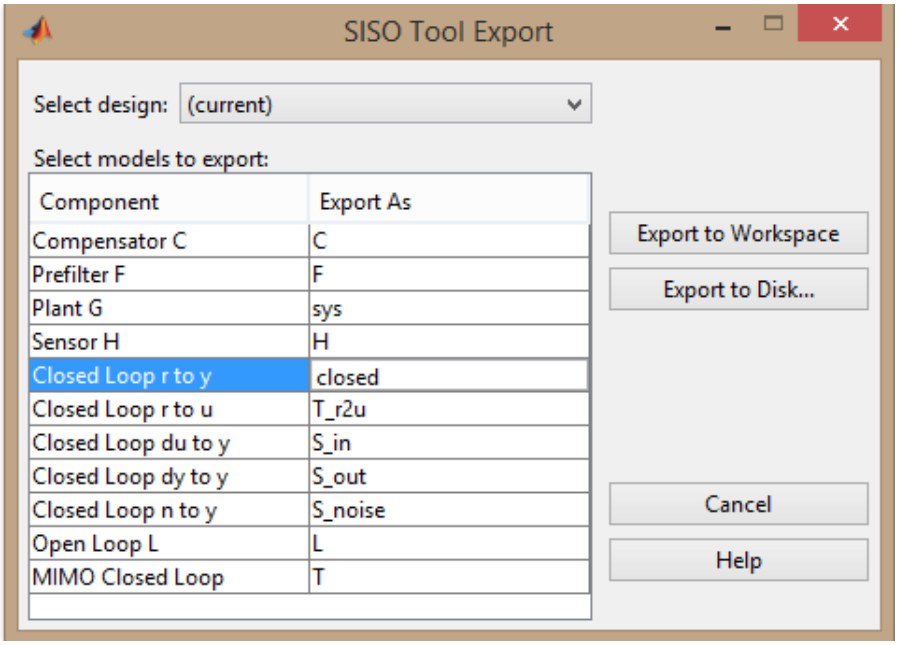
On Analysis Plots, time and frequency domain plots are available, like step and impulse responses, and Nyquist plots. The user may choose to which part of the system the plot will correspond to. For the complete system, one must choose the *closed loop r to y*, but at times it will be useful to examine parts of the system separately, like the plant G or the compensator C.

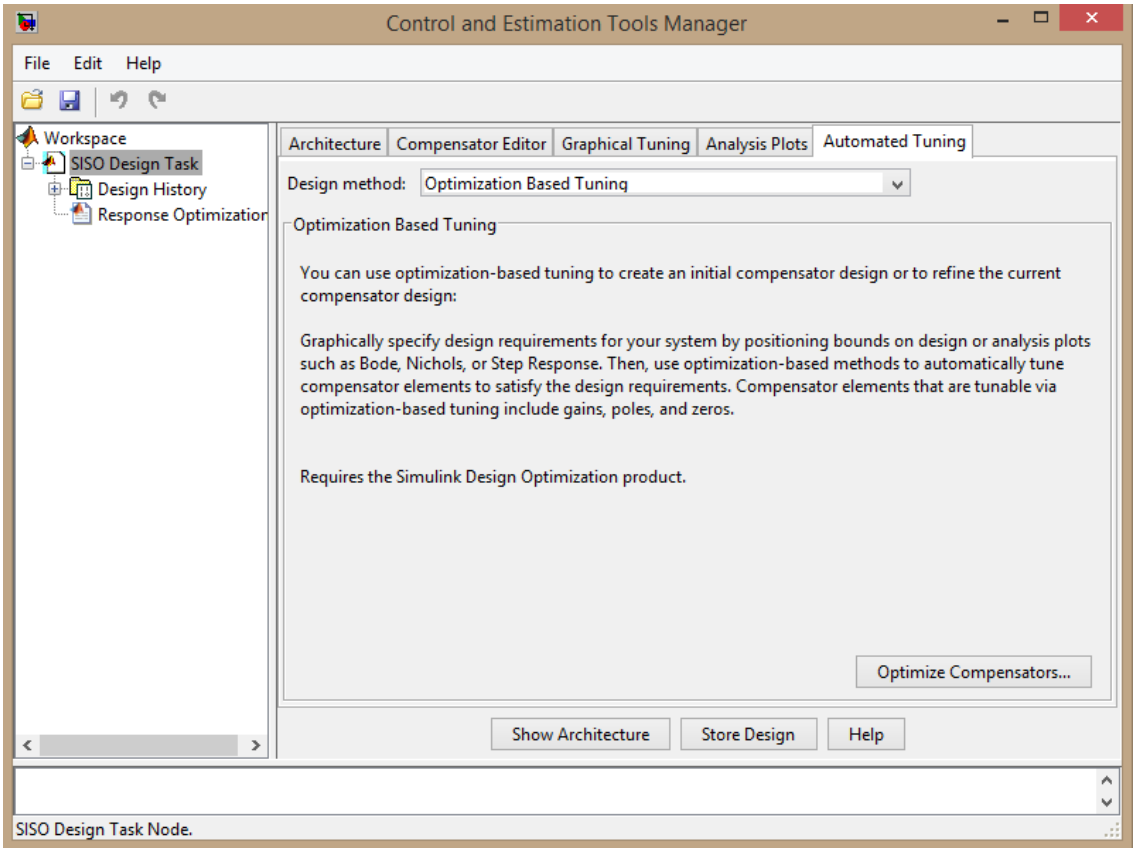
Additionally, the same plot type can be chosen for different subsystems and displayed on the same graph, for the convenience of the user.

Lastly, on Automated Tuning the user may automatically design a compensator for the system, by specifying requirements and constraints on the analysis plots (Step, Bode, Pole-Zero).

Now, one last feature that needs to me mentioned is the ability to export the design data as state space models to the workspace. So, if for example we want to aquire the closed loop state space system that we have designed, from the Control Systems Toolbox® menu, choose File → Export. A new window will pop up. On this window, one can choose the desired part (gain, compensator, complete system etc) of the design and export it in the workplace as a state space variable.

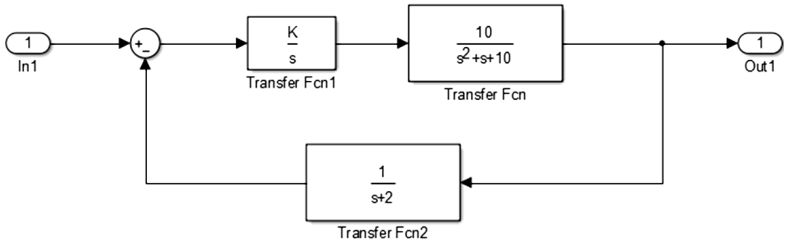

Control and Estimation Tools Manager
File Edit Help
Workspace
SISO Design Task
Design History
Architecture Compensator Editor Graphical Tuning Analysis Plots Automated Tuning
Analysis Plots
Plot 1 Plot 2 Plot 3 Plot 4 Plot 5 Plot 6
Plot Type Step Nyquist None None None None
Contents of Plots
Plots 1 2 3 4 5 6 All
Responses
Closed Loop r to y
Closed Loop r to u
Closed Loop du to y
Closed Loop dy to y
Closed Loop n to y
Open Loop L
Compensator C
Prefilter F
Plant G
Sensor H
Add Responses ...
Show Analysis Plot
Show Architecture
Store Design
Help


SISO Tool Export
Select design: (current)
Select models to export:
Component
Export As
Compensator C C
Prefilter F F
Plant G sys
Sensor H H
Closed Loop r to y closed
Closed Loop r to u T_r2u
Closed Loop du to y S_in
Closed Loop dy to y S_out
Closed Loop n to y S_noise
Open Loop L L
MIMO Closed Loop T
Export to Workspace
Export to Disk...
Cancel
Help



12.3 Examples

Example 12.3.1. Consider the following system



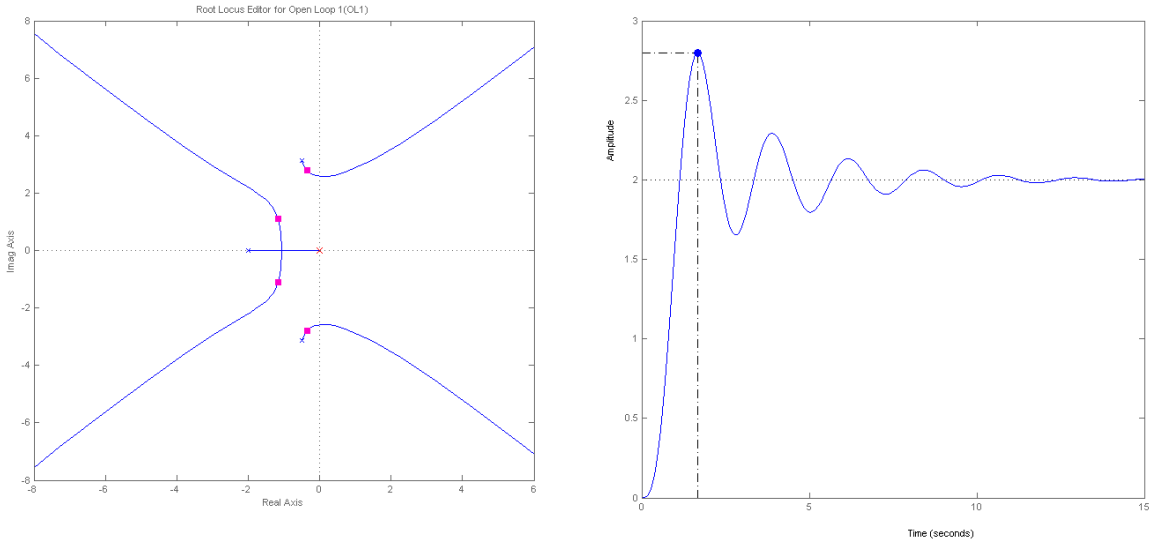
After you import this system in sisotool:

1. Plot system's step response for K=2.
2. Plot the system's root locus.
3. Find the closed loop transfer function.

Solution. First, we begin by defining the different blocks in Matlab. Then we open up sisotool and import these blocks to the system

```
sys_fw=tf(10, [1 1 10]); % The open loop block
sys_bw=tf(1,[1 2]); % The feedback block
sisotool %opens sisotool
```

After we import the above transfer function blocks in SISOtool, we choose the correct value for K from Compensator Editor, and also add an additional zero at 0. The root locus and step response of the system are the following



Now, in order to find the closed loop transfer function, we export the *closed loop r to y* system , and using the transfer function command, we get

$$Closed(s) = \frac{20s + 40}{s^4 + 3s^3 + 12s^2 + 20s + 20}$$

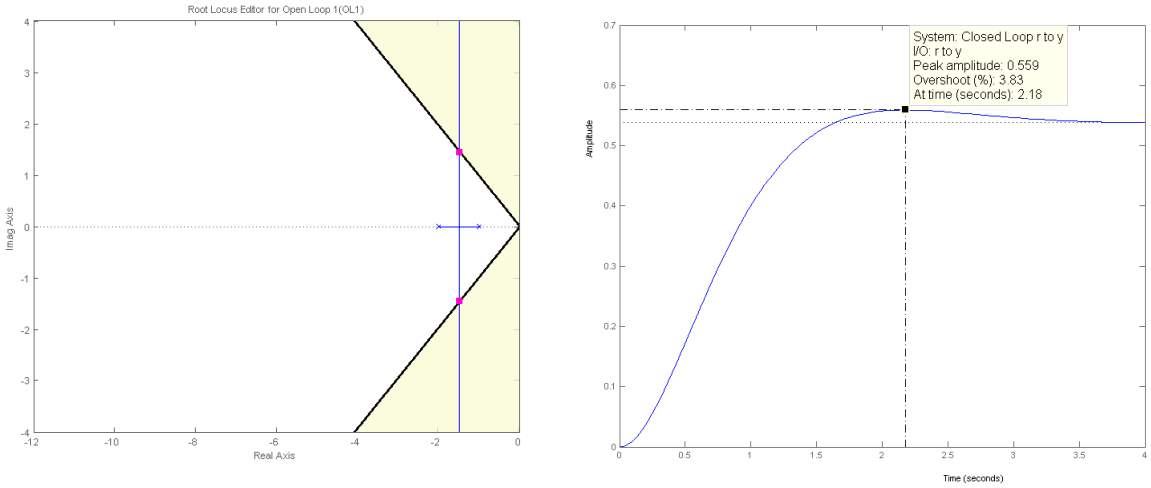
Example 12.3.2. Consider the closed loop system with open loop transfer function

$$G(s) = \frac{K}{(s+1)(s+2)}$$

After you import this system in sisotool:

1. Using the root locus, find the gain matrix K such that the system has an Overshoot of 4%.
2. Plot the system's step response.
3. Find the closed loop transfer function.

Solution. First, we begin by defining the transfer function in the command window. Then, after we import the above transfer function in SISOtool, we open up the root locus plot and input the desired constraints. Then, we interactively change the gain in order to place the system's poles on the intersections of the root locus and the constraint lines, just like the following image shows.



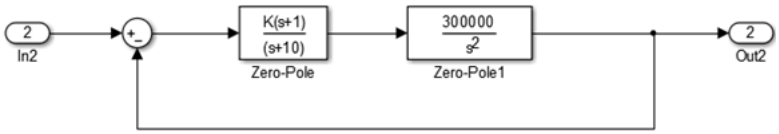
Now, in order to find the closed loop transfer function, we export the *closed loop r to y* system , and using the transfer function command, we get

$$Closed(s) = \frac{2.335}{s^2 + 3s + 4.335}$$

Example 12.3.3. [Nise, 2013, Yan and Lin, 2003] The read/write head assembly arm of a computer hard disk drive (HDD) can be modeled as a rigid rotating body with inertia I_b. Its dynamics can be described with the transfer function

$$P(s) = \frac{X(s)}{F(s)} = \frac{1}{I_b s^2}$$

where X(s) is the displacement of the read/write head and F(s) is the applied force (Yan, 2003). Assume the arm has an inertia of $3 \cdot 10^{-5} kg \cdot m^2$ and that a lead controller G_c is placed in series to yield the following system



with open loop transfer function

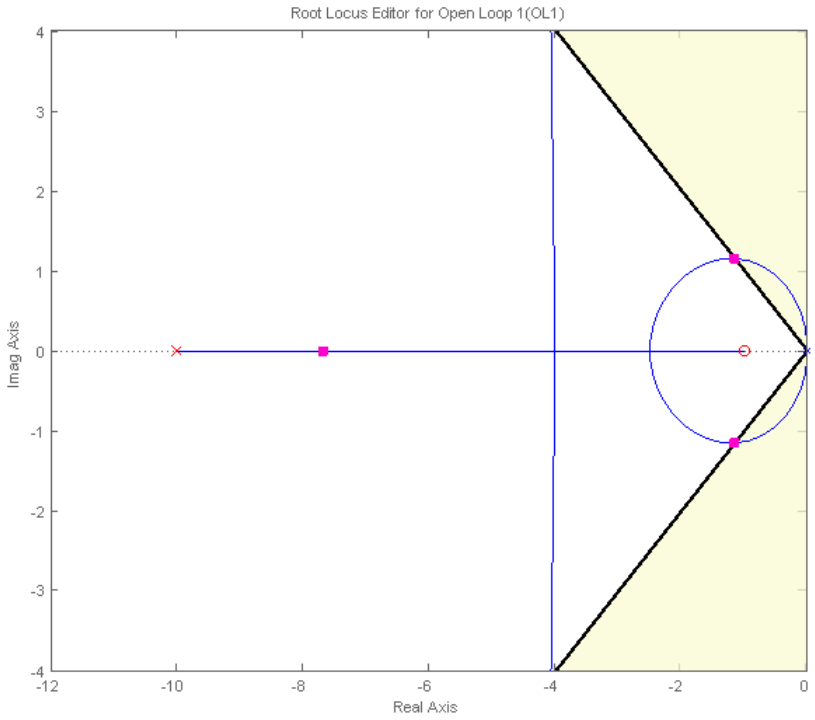
$$G(s) = G_c(s)P(s) = \frac{K}{I_b s^2}\frac{(s+1)}{(s+10)}$$

After you import this system in sisotool:

1. Find the value of K that will result in dominant complex conjugate poles with a $\zeta = 0.707$ damping factor.

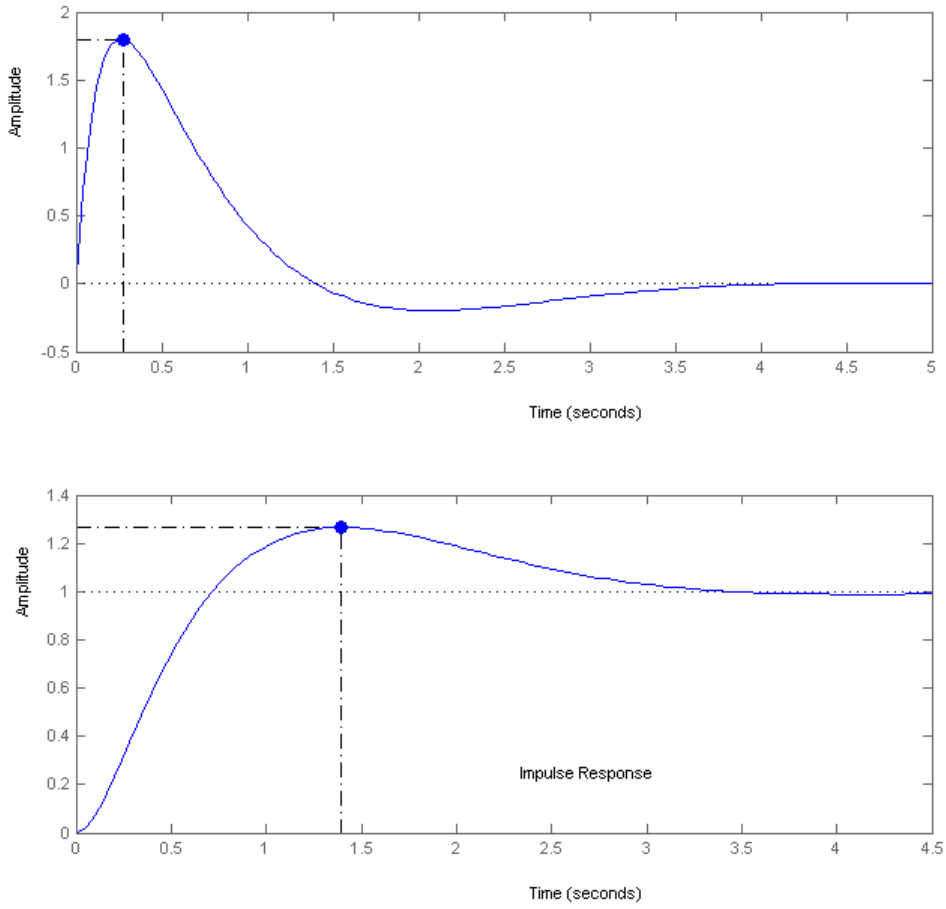
2. Plot the system's impulse and step responses.

Solution. First, we begin by defining the different blocks in Matlab. Then we open up sisotool and import these blocks to the system. The next step is to import the desired constraint of $\zeta = 0.707$ on the root locus. Then, we interactively change the gain in order to place the system's poles on the intersections of the root locus and the constraint lines, just like the following image shows



This corresponds to a gain value of $K = 6.8138 \cdot 10^{-6}$ and closed loop poles $s = -7.6866,\ -1.1567 \pm 1.1495i$. It is clear that we have two complex conjugate poles that constitute the dominant poles of the system and one more real pole that is far on the left and thus much less important in the dynamic behavior of the closed loop system.

The step and impulse responses are the following


Amplitude
Time (seconds)
Impulse Response
Amplitude
Time (seconds)

Chapter 13

Designing Controllers Using the Control Systems Toolbox

13.1 Introduction

Using SISOtool, one can interactively design controllers for the system, in order to change its characteristics. The observer design can have two different aims. Either change the transient response characteristics of the system (rise time, settling time, % overshoot) or change its steady state characteristics (steady state). Of course, more advanced controllers, like PID controllers, can change both transient and steady state characteristics of a system.

The basic idea behind controller design is this. Change the root locus of the system, in order to make it pass through specific points that satisfy the design requirements. How can this be achieved though?

As is well known, for any point s_0 in the root locus, the angular contribution of the poles and zeros of the system satisfies

$$\angle G(s)|_{s=s_0} = (2k+1)\pi$$

So the main idea of the design is to add poles and zeros to the system, so that the angular contribution at the desired points in the Complex Plane satisfies the above relation.

13.2 Improving Transient Response

To improve the transient response of the system, PD and Lead compensators are used. Their characteristics are shown in the following table

Ideal Proportional Derivative (PD)	$C(s) = K(s + z_c)$	A gain plus a differentiator.
Lead	$C(s) = K\frac{s+a}{s+r\cdot a}$, $1 \leq r \leq 10$	The pole is further on the left, so that the angular contribution is positive $\angle(s+a) - \angle(s+r\cdot a) > 0$.

Lead compensators are actually an approximation of a PD. The change these compensators bring about on the root locus will be made clear in the following examples.

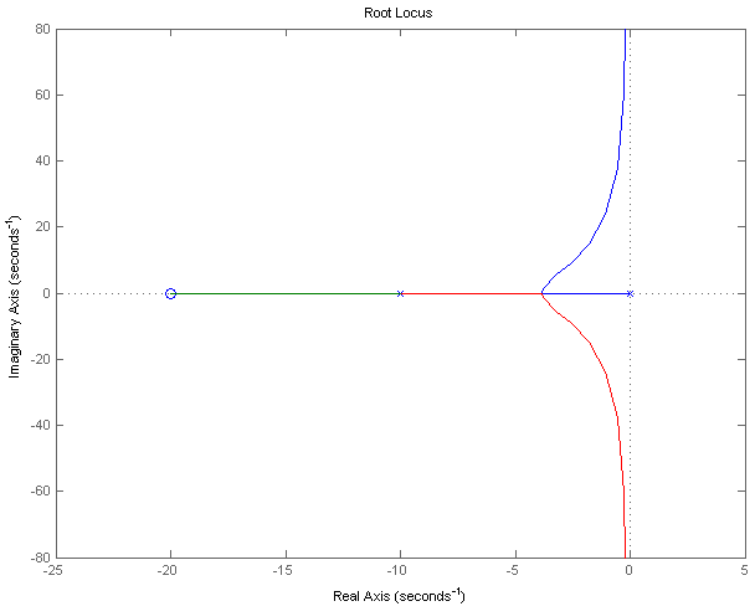
13.2.1 Examples

Example 13.2.1. [Dorf and Bishop, 2009] Designers have developed small, fast, vertical-takeoff fighter aircraft that are invisible to radar (stealth aircraft). This aircraft concept uses quickly turning jet nozzles to steer the airplane [22]. The control system for the heading or direction control is a unity feedback system with the Aircraft dynamics described by the transfer function

$$G_{plant} = \frac{s + 20}{s(s + 10)^2} \tag{13.1}$$

along with a Proportional Controller. Plot the root locus and determine the maximum gain K that ensures stability.

Solution. If we plot the root locus of the above system, we can see that the locus lies completely on the negative part of the complex plane. That means that the system is stable for all $K > 0$. So the usage of the Proportional gain K lies in the tuning of the response characteristics of the system.



Example 13.2.2. [Nise, 2013, Yan and Lin, 2003] Going back to Example 12.3.3, we have the read/write head assembly arm of a computer hard disk drive (HDD), whose dynamics are described by the open loop transfer function

$$P(s) = \frac{X(s)}{F(s)} = \frac{3 \cdot 10^5}{s^2}$$

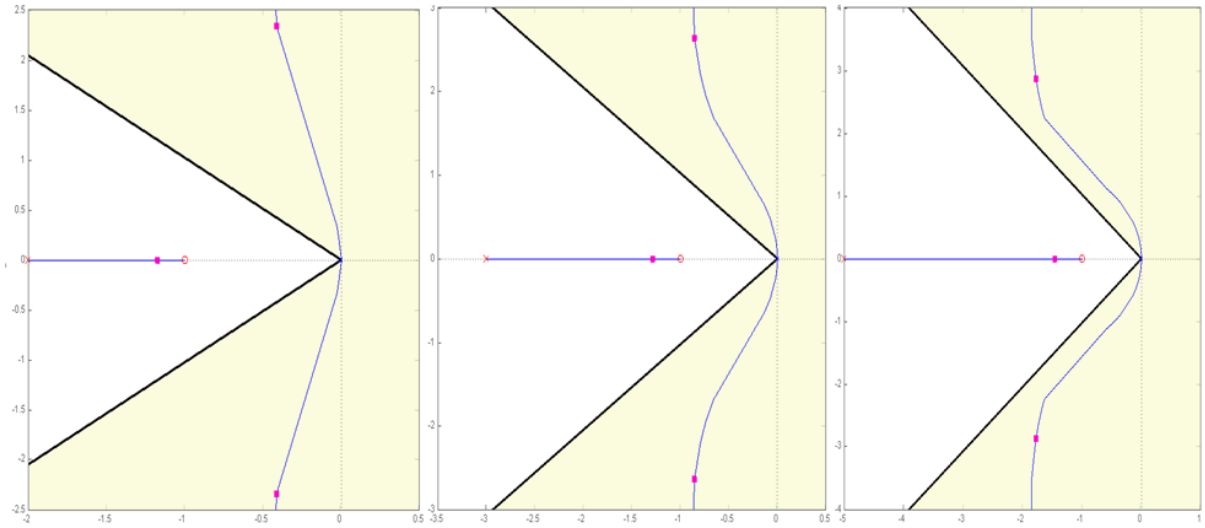
To this system we add a lead controller in order to change the systems stability. Our aim is to find a lead or PD controller such that for an appropriate gain value K the system has dominant conjugate poles with a $\zeta = 0.707$ damping factor.

Solution. Let us consider the compensator

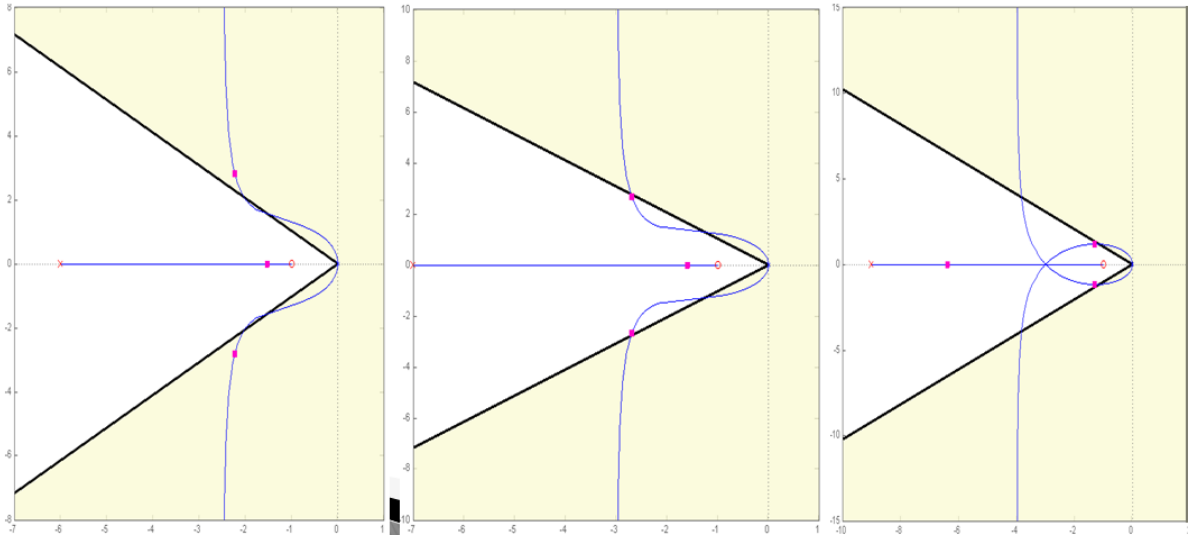
$$G_c(s) = K\frac{s+1}{s+r\cdot 1} \tag{13.2}$$

After we import the system in sisotool, we try different values for r, moving the pole of the system further to the left in the complex plane little by little. This can be done either in the compensator editor or by moving the pole of the compensator in the root locus (the red "x"). This change in the root locus is displayed in the following figures

$C(s) = K\frac{(s+1)}{(s+2)}$ $C(s) = K\frac{(s+1)}{(s+3)}$ $C(s) = K\frac{(s+1)}{(s+5)}$



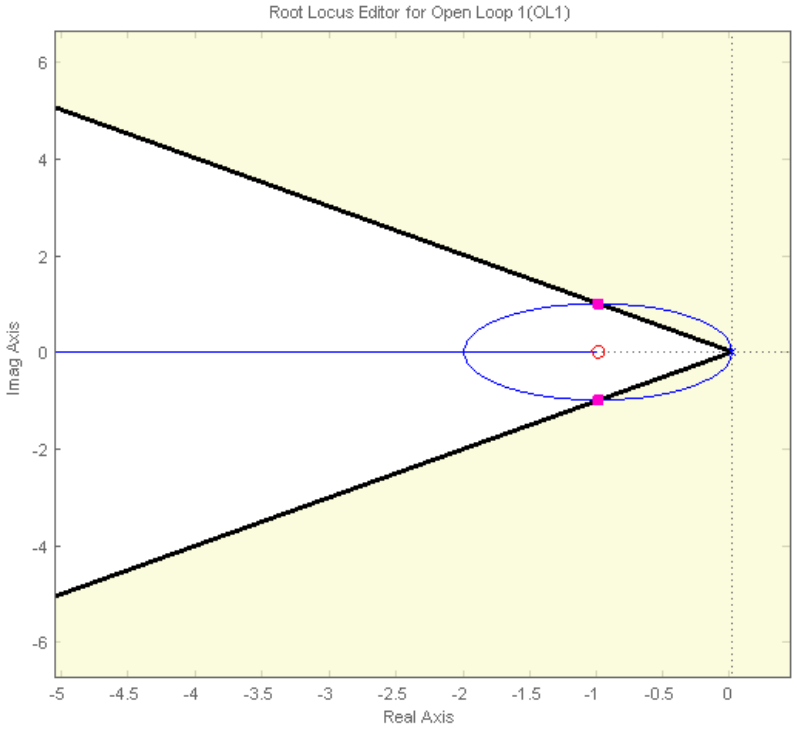
$$C(s) = K\frac{(s+1)}{(s+6)} \qquad C(s) = K\frac{(s+1)}{(s+7)} \qquad C(s) = K\frac{(s+1)}{(s+9)}$$



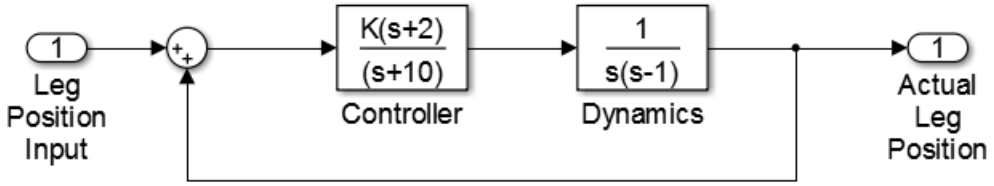
The same pole placement can be achieved by using the PD controller

$$G_c(s) = 6.671 \cdot 10^{-06}(s+1)$$

which will yield the following root locus



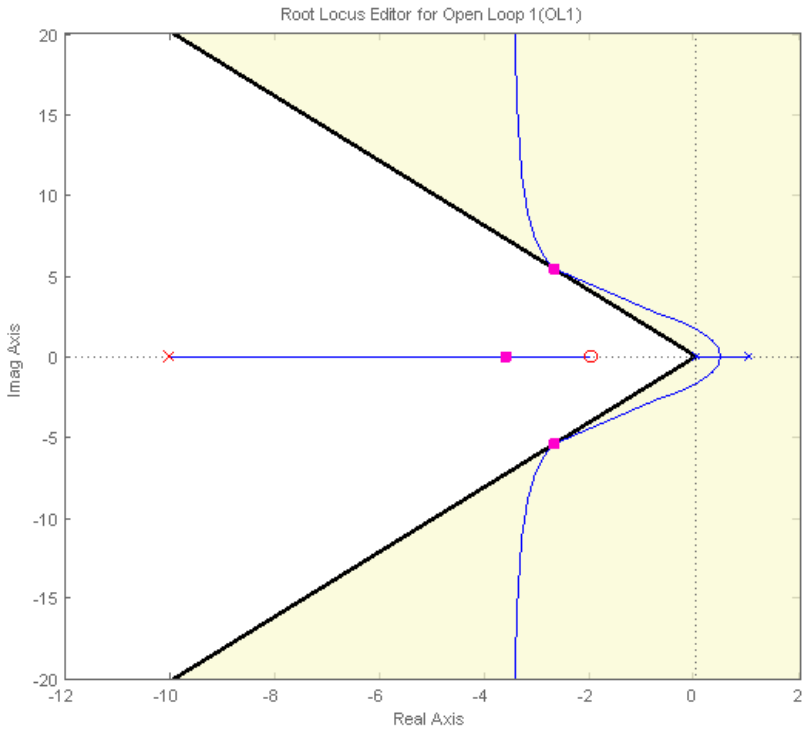
Example 13.2.3. [Dorf and Bishop, 2009, de Panne et al., 1992] Automatic control systems are used to aid and control the walk of partially disabled persons. One model of a system that is open-loop unstable is shown in the following figure



1. Using the root locus in SISOtool, find the value of K that achieves the maximum number of damping factor ζ.
2. Plot the step response of the system.

Solution. Importing the system in sisotool, we plot the root locos of the system. We can see that the addition of the lead compensator can drive the system to stability for gain values $K > 2.6$.

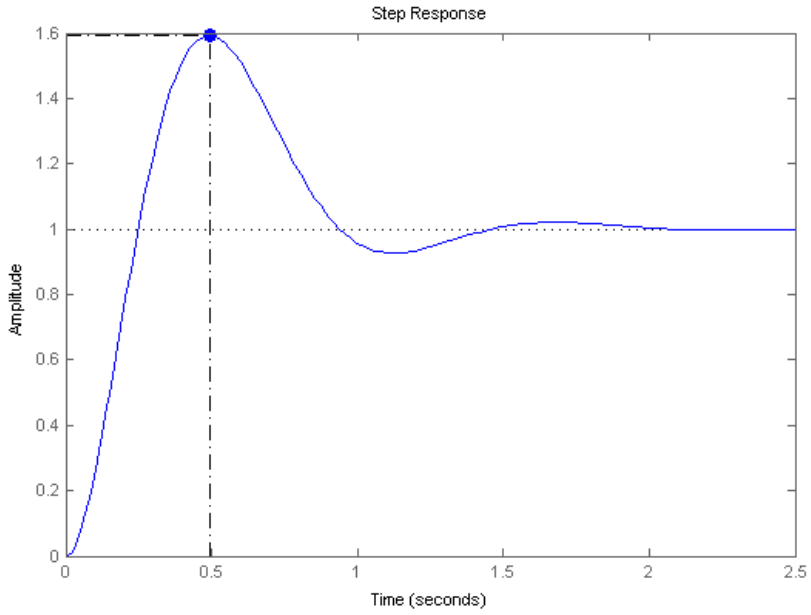
Now, the maximum damping factor for complex roots is achieved for the smaller angle the root locus makes with the real axis, as is seen in the following figure



This point corresponds to a damping factor $\zeta = 0.446$ and the compensator is

$$G_c(s) = \frac{65.968(s+2)}{(s+10)}$$

the step response for the system is the following



Example 13.2.4. [Dorf and Bishop, 2009, Martin, 1999] A unity feedback control system for a robot submarine has a plant with a third-order transfer function

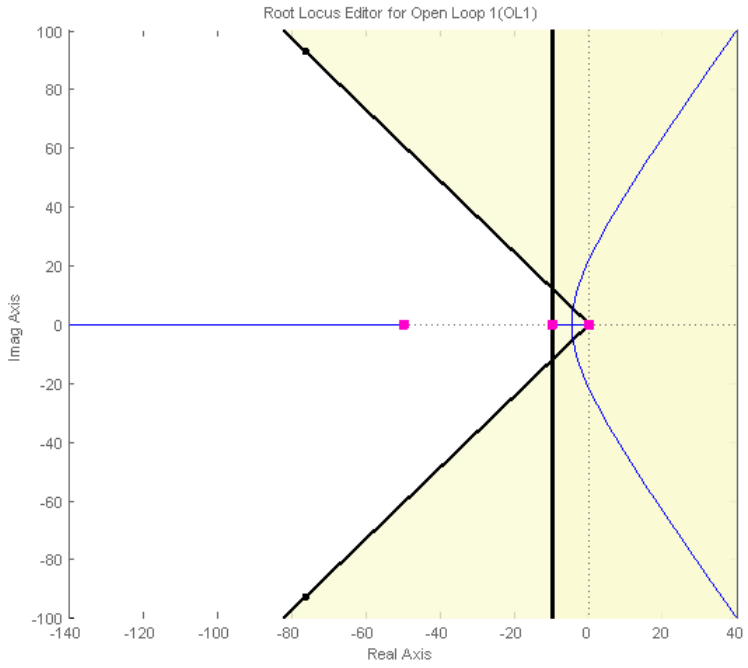
$$G(s) = K\frac{1}{s(s+10)(s+50)}$$

Add a lead controller to this system, in order to achieve an overshoot of approximately 7.5% and a settling time of 0.4sec. Let the zero of the controller be located at -15.

Solution. Let us consider the lead controller

$$G_c(s) = K\frac{s+15}{s+r\cdot 15} \tag{13.3}$$

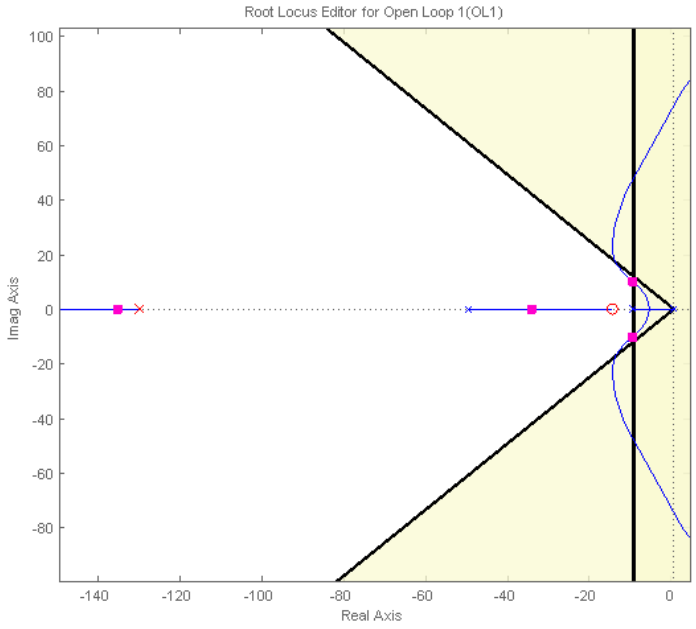
This means that since we have created a lead controller, the pole must be further on the left than the zero on the complex plane. Importing the system and design requirements in sisotool we get the following root locus (before we add the controller)

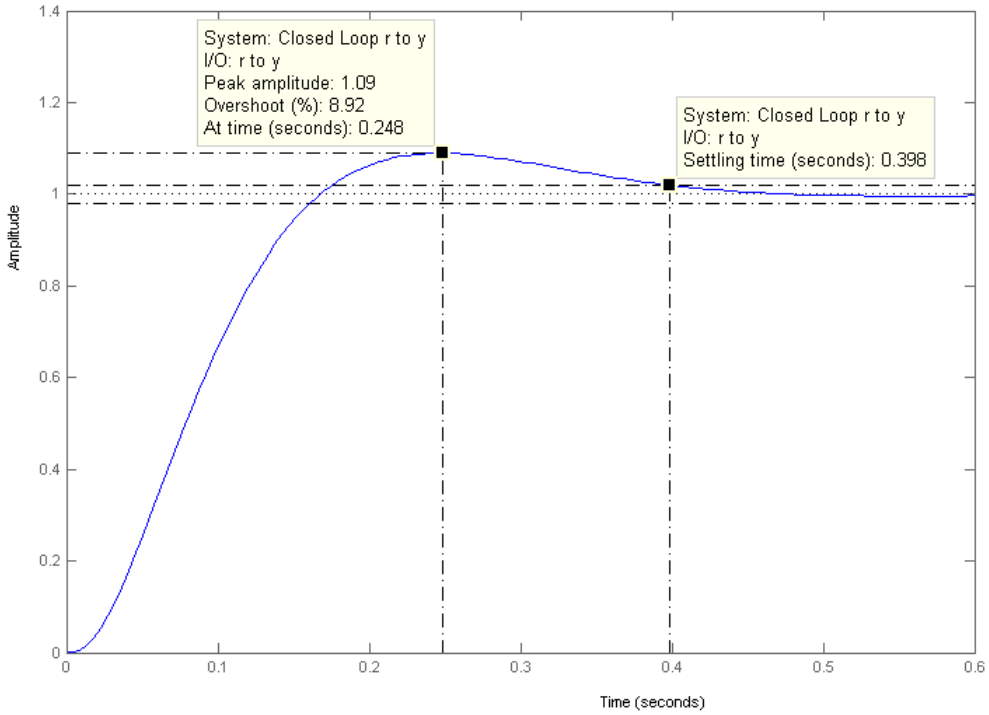


So the system obviously does not satisfy the design requirements. Now, adding the controller and trying different values for its pole, we end up with

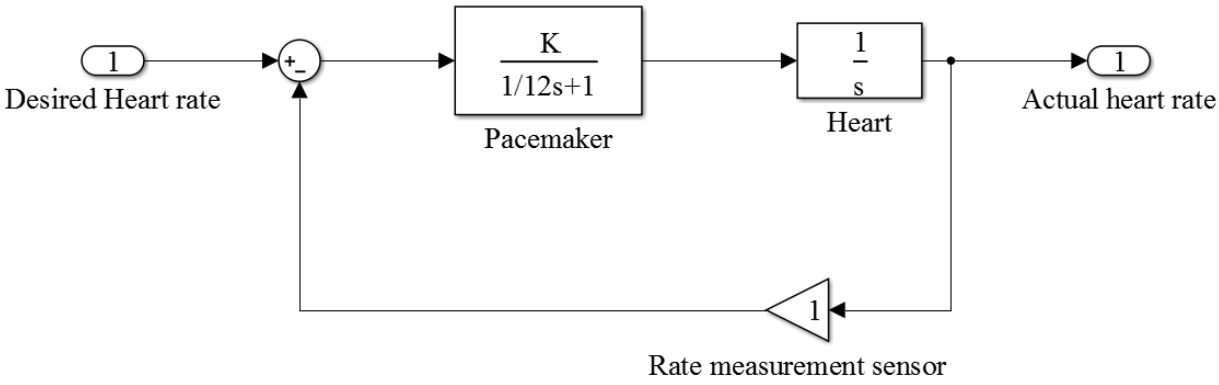
$$G_c(s) = 63493\frac{s+15}{s+130} \tag{13.4}$$

which gives us the following root locus and step response





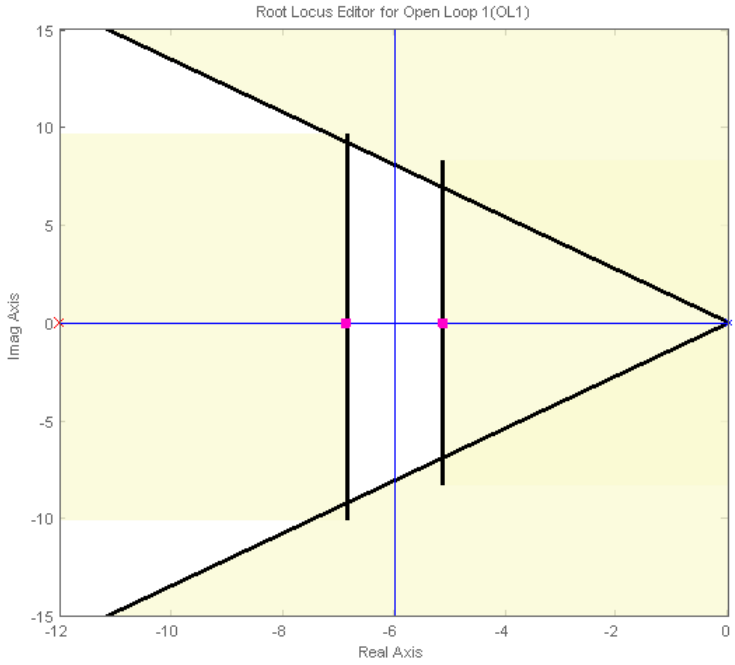
Example 13.2.5. [Dorf and Bishop, 2009] Electronic pacemakers for human hearts regulate the speed of the heart pump. A proposed closed-loop system that includes a pacemaker and the measurement of the heart rate is the following



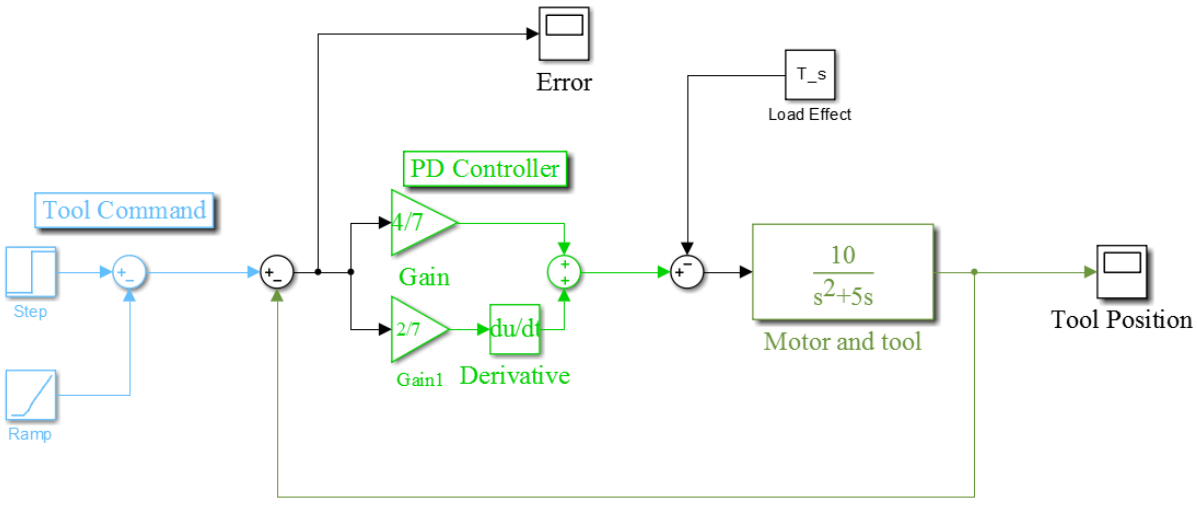
Find the range of the amplifier gain K to yield a system with a settling time to a step disturbance of less than 1 second. and an overshoot of less than 10%.

Solution. Designing the system in sisotool, we can see from the root locus that the limit values of K to achieve a 10% overshoot is around K=8.6. For this K the settling time is 0.58sec which is accepted. Making the same analysis for settling time, we find that the

acceptable values for both a maximum 10% overshoot and a maximum 1sec settling time, the acceptable values for K are $2.9 \leq K \leq 8.6$. The root locus for the given values is the following.



Example 13.2.6. [Dorf and Bishop, 2009] A machine tool is designed to follow the input $r(t) = (1 - t)u(t)$, with as little error as possible, where u(t) is the unit step function. The system's configuration is the following

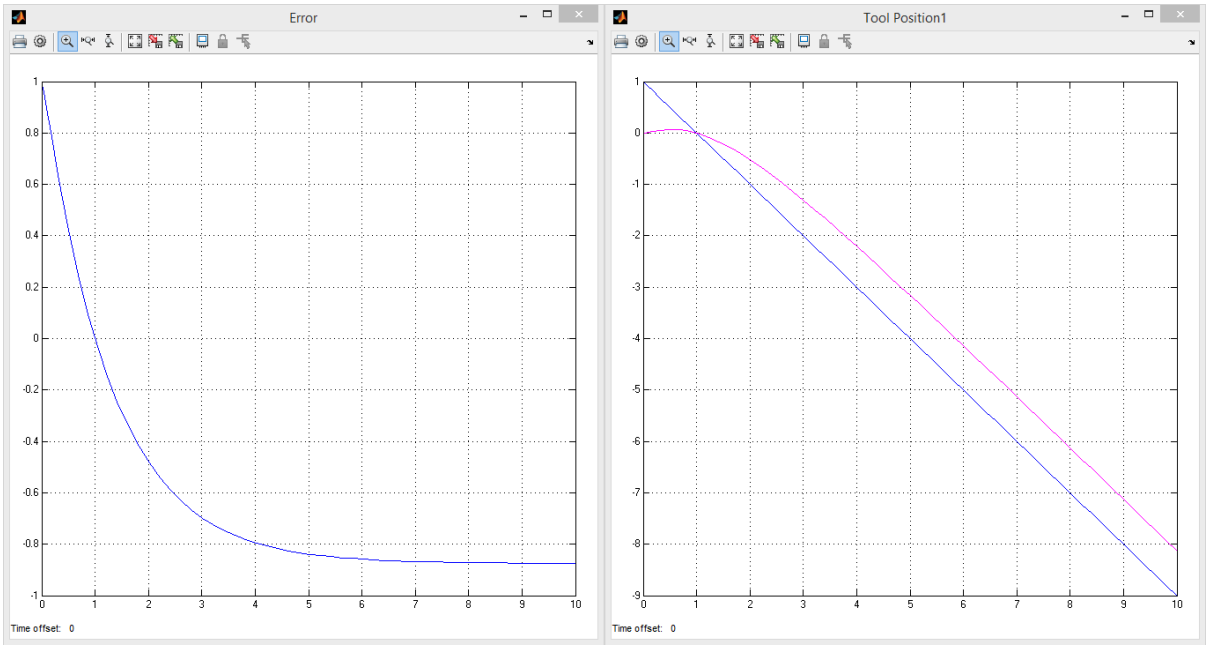


After you design the system in Simulink:

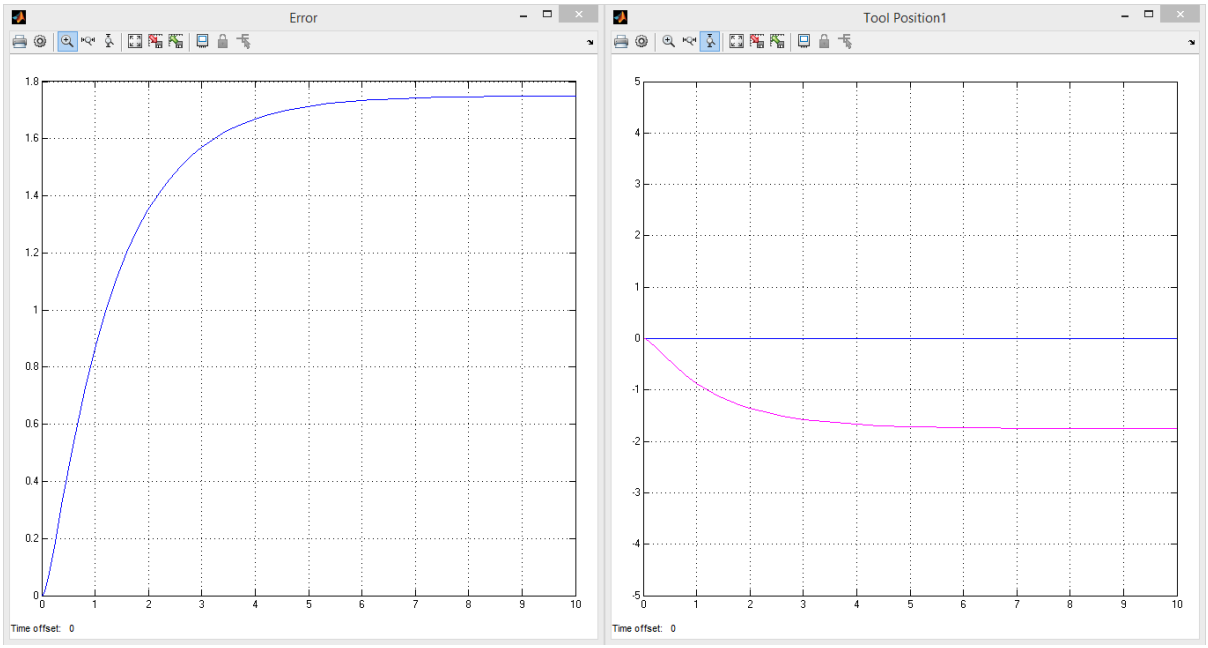
1. Plot the error between real and desired position for the given input, when the Load effect is zero, for $0 \leq t \leq 10$.

2. Plot the error between real and desired position when the input is zero and the Load effect is $T_d(s) = \frac{1}{s}$.

Solution. We begin by designing the system in Simulink and tuning all the systems parameters, such as the simulation time for the model and the step time in the step block. For a load effect equal to zero, we get the following error and response plots



Now, if we assume a zero input and a load effect equal to a step signal, we get the following results



13.3 Improving Steady State Response

To improve the steady state response of the system, PI and Lag compensators are used. Their characteristics are shown in the following table

Ideal Proportional Integral (PI)	$C(s) = K\frac{s+a}{s}$	Pure integrator that increases the system type and drives the error to zero.
Lag	$C(s) = K\frac{s+r\cdot a}{s+a}$, $1 \leq r \leq 10$	Approximates an integrator and reduces the error.

The idea behind an Integral compensator is to increase the system Type. By that, we mean the number of system's poles located at 0. So in general, if we consider systems of the form

$$G(s) = \frac{(s+z_1)(s+z_2)\cdots}{s^n(s+p_1)(s+p_2)\cdots} \tag{13.5}$$

for $n = 0$, the system is of Type 0, for $n = 1$ of Type 1 etc. The system Type is crucial in defining the steady state error of the system to different kind of responses. For example, a Type 1 system will have zero steady state error to step input, constant error to ramp input and an infinite error to parabola input.

The zero a is always placed near zero, so that the angular contribution of the controller is approximately equal to zero. Thus the root locus remains unchanged.

Lag compensators are actually an approximation of a PI. The change these compensators bring about on the root locus will be made clear in the following examples.

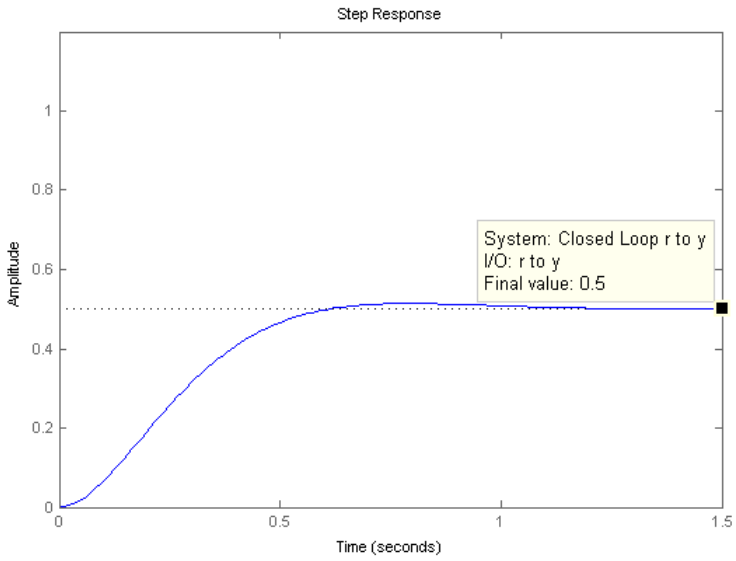
13.3.1 Examples

Example 13.3.1. Consider the system with unity feedback and plant

$$G(s) = \frac{18}{(s+3)(s+6)}$$

1. Add a PI controller such that the system has zero steady state error to step input and settling time less than 1sec.
2. Plot the system's step response.

Solution. The step response of the uncompensated system is



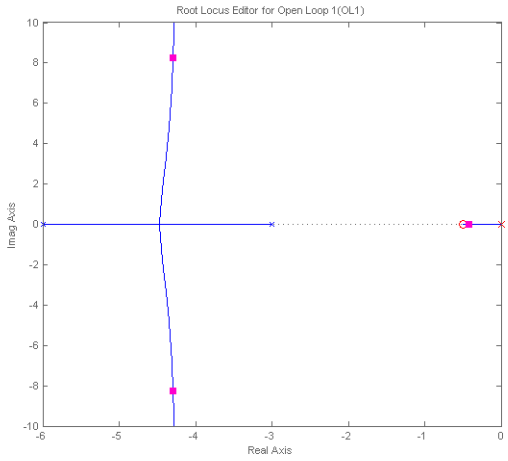
and it is clear that the steady state error is $e = 1 - 0.5 = 0.5$. Now, let us consider the PI controller

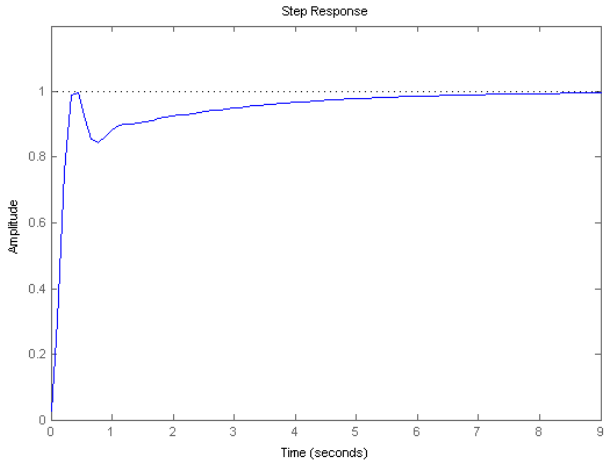
$$G_c(s) = K\frac{s+a}{s}$$

The pole is at zero in order to raise the systems type to 1. The zero a can be put near zero, in order to keep the root locus unchanged. Choosing

$$G_c(s) = 4\frac{s+0.5}{s}$$

we get the following root locus and step response





which makes it clear that the error is equal to zero.

Chapter 14

Discrete Time Systems

14.1 Introduction

A system can represent physical dynamic phenomena or dynamic phenomena such as those encountered in economics or other social sciences. If the time space is continuous, the system is known as a continuous-time system. However, if the input and state vectors are defined only for discrete instants of time k, where k ranges over the integers, the time space is discrete and the system is referred to as a discrete-time system. In this chapter we will discuss converting continuous time models into discrete time (or difference equation) models using Matlab.

14.2 Discretization

Discretization is also concerned with the transformation of continuous differential equations into discrete difference equations. We can define continuous-time linear models in Matlab using the `tf`, `ss`, and `zpk` commands. The command `ss` creates a state-space model, whereas `tf` and `zpk` represent transfer function models and zero-pole-gain models respectively.

`sys=ss(A,B,C,D)`	Creates a state-space model object representing the continuous-time state-space model: $$x'(t) = Ax(t) + Bu(t)$$ $$y(t) = Cx(t) + Du(t)$$ where `A,B,C,D` are real or complex valued matrices, x(t) is the state vector, u(t) is input vector, and y(t) is the output trajectory.

The syntax for creating discrete-time models is similar to that for continuous-time models, except that we must also provide a sample time (sampling interval in seconds).

`sys=ss(A,B,C,D,Ts)`	Creates the discrete-time state-space model: $$x[n+1] = Ax[n] + Bu[n]$$ $$y[n] = Cx[n] + Du[n]$$ with sample time `Ts` (in seconds)
`sys=tf(num,den,Ts)`	Creates a discrete-time transfer function with sample time `Ts` (in seconds). The input arguments `num` and `den` are as in the continuous-time case and must list the numerator and denominator coefficients in descending powers of z.
`sys=zpk(z,p,k,Ts)`	Creates a discrete-time zero-pole-gain model with sample time `Ts` (in seconds). The input arguments `z,p,k` are as in the continuous-time case.

For example, to specify the discrete-time transfer function: $H(z) = \dfrac{z-1}{z^2 - 1.85z + 0.9}$ with sampling period Ts = 0.2s, we type the following:

```
num=[1 -1];
den=[1 -1.85 0.9];
H=tf(num,den,0.2)
```

To specify the discrete-time state-space model:

$$x[k+1] = 0.5x[k] + u[k]$$

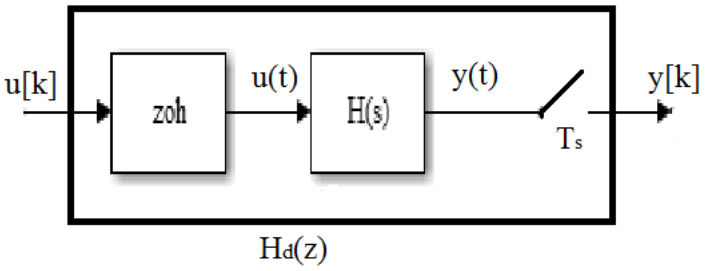
$$y[k] = 0.2x[k],$$

with sampling period Ts = 0.1s, we type:

```
sys = ss(.5,1,.2,0,0.1)
```

In Matlab, the `c2d` command discretizes continuous-time models. Conversely, `d2c` converts discrete-time models to continuous time. Commands `c2d` and `d2c` support the following discretization and interpolation methods:

• *Zero-Order Hold* (`zoh`): The Zero-Order Hold method provides an exact match between the continuous and discrete-time systems in the time domain for staircase inputs. The following block diagram shows the zero-order-hold discretization $H_d(z)$ of a continuous-time linear model $H(s)$.



The zoh block generates the continuous-time input signal u(t) by holding each sample value u(k) constant over one sample period:

$$u(t) = u[k], \qquad kT_s \le t \le (k+1)T_s$$

The signal u(t) is the input to the continuous system $H(s)$. The output y[k] results from sampling y(t) every T_s seconds.

• *First-Order Hold* (`foh`): The First-Order Hold method provides an exact match between the continuous and discrete-time systems in the time domain for linear inputs. FOH differs from ZOH by the underlying hold mechanism. To turn the input samples u[k] into a continuous input u(t), FOH uses linear interpolation between samples:

$$u(t) = u[k] + \frac{t - kT_s}{T_s}(u[k+1] - u[k]), \qquad kT_s \le t \le (k+1)T_s$$

This method is generally more accurate than ZOH for systems driven by smooth inputs.

• *Impulse-Invariant Mapping [used for `c2d` only]* (`impulse`): The Impulse-Invariant Mapping produces a discrete-time model with the same impulse response as the continuous time system.

• *Tustin Approximation* (`tustin`): The Tustin approximation yields the best frequency-domain match between the continuous-time and discretized systems. This method relates the s-domain and z-domain transfer functions using the approximation:

$$z = e^{sT_s} \approx \frac{1 + sT_s/2}{1 - sT_s/2}$$

In `c2d` conversions, the discretization $H_d(z)$ of a continuous transfer function `H(s)` is: $H_d(z) = H(s')$, with $s' = \frac{2}{T_s}\frac{z-1}{z+1}$ and the `d2c` conversion relies on the inverse correspondence $H(s) = H_d(z')$, with $z' = \frac{1 + sT_s/2}{1 - sT_s/2}$.

• *Zero-Pole Matching* (`matched`):The method of conversion by computing zero-pole matching equivalents applies only to SISO systems. The poles and zeros of the continuous and

discretized systems are related by the transformation: $z_i = e^{s_i T_s}$, where z_i is the ith pole or zero of the discrete-time system, s_i is the ith pole or zero of the continuous-time system and T_s is the sample time.

`c2d(sys,Ts,method)`	Discretizes the continuous-time dynamic system model `sys` using the specified discretization method `method`. We write: `c2d(sys,Ts,'zoh')`, `c2d(sys,Ts,'foh')`, `c2d(sys,Ts,'impulse')`, `c2d(sys,Ts,'tustin')` or `c2d(sys,Ts,'matched')`, according to the method that we want to use.
`d2c(sys,method)`	Produces a continuous-time model that is equivalent to the discrete-time dynamic system model `sys` using the specified conversion method `method`. We write: `c2d(sys,Ts,'zoh')`, `c2d(sys,Ts,'foh')`, `c2d(sys,Ts,'tustin')` or `c2d(sys,Ts,'matched')`, according to the method that we want to use.
`sys1=d2d(sys,Ts)`	Resamples the discrete-time dynamic system model `sys` to produce an equivalent discrete-time model `sys1` with the new sample time `Ts` (in seconds), using zero-order hold on the inputs.
`sys1=d2d(sys,Ts,method)`	Resamples the discrete-time dynamic system model `sys` to produce an equivalent discrete-time model `sys1` with the new sample time `Ts` (in seconds), using the specified resampling method `method`, which can be `'zoh'` (Zero-order hold) or `'tustin'` (Tustin approximation).

14.3 Examples

Example 14.3.1. Discretize the following state-space system with zero-order hold and first-order hold:

$$x'(t) = \begin{pmatrix} -0.5 & 2 \\ 0 & -0.5 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t)$$
$$y(t) = \begin{pmatrix} 1 & 0 \end{pmatrix} x(t)$$

Solution.

```
A=[-0.5 2;0 -0.5];
B=[0;1];
C=[1 0];
```

```
D=0;
sys=ss(A,B,C,D); %  Creates the state-space model
sys_d1=c2d(sys,1,'foh') % Discretizes sys using first-order hold
sys_d2=c2d(sys,1,'zoh') % Discretizes sys using zero-order hold
```

Example 14.3.2. [Kelley and Peterson, 2001] In a certain agricultural delta, populations of owls and mice coexist under normal conditions in a predator and prey relationship, to a stable equilibrium population of O thousand owls and M million mice. However, extreme winter conditions can reduce the owl population drastically. In the model below, the owl and mouse populations are gradually restored to their normal equilibria.

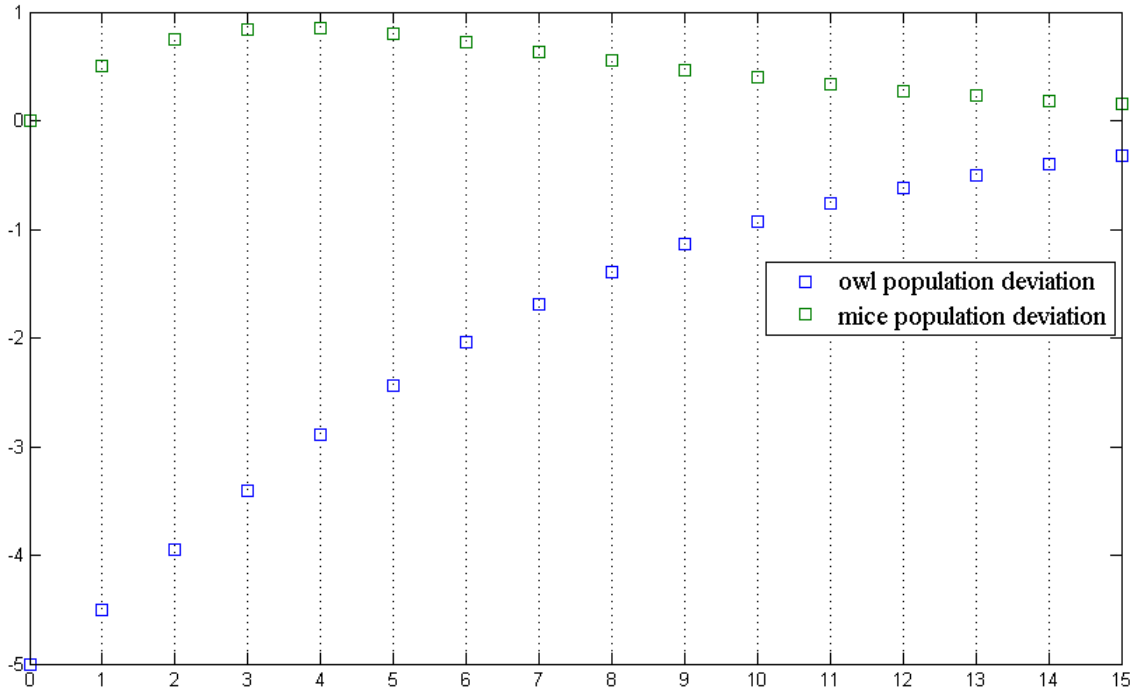
Let x(k) and y(k) denote the deviations of the owl and mouse populations respectively, from their usual levels at the beginning of the k-th year, so that $O+x(t)$ is the population of owls (in thousands) and $M+y(k)$ is the population of mice (in millions). Their relation is described by the following system of two difference equations

$$\begin{pmatrix} x(k+1) \\ y(k+1) \end{pmatrix} = \begin{pmatrix} 0.9 & 0.2 \\ -0.1 & 0.6 \end{pmatrix} \begin{pmatrix} x(k) \\ y(k) \end{pmatrix}$$

Define the following system as a discrete time state space system with a period of T=1 year and plot the two states for initial conditions $x(0) = -5$ and $y(0) = 0$ for a period of 15 years.

Solution.

```
% First we define the discrete time system
sys=ss([0.9,0.2;-0.1,0.6],[0;0],eye(2),0,1);
y=initial(sys,[-5,0],15);
% The above vector y has two columns, one for each state of the system
years=0:15;
plot(years,y(:,1)','s')
hold all
plot(years,y(:,2)','s')
legend('owl population deviation','mice population deviation')
```



From the figure we can observe that the deviations of the two populations tend to zero after a period of about 14 years.

Example 14.3.3. Consider a system with the following transfer function

$$G(s) = \frac{49}{s^2 + 2s + 49}$$

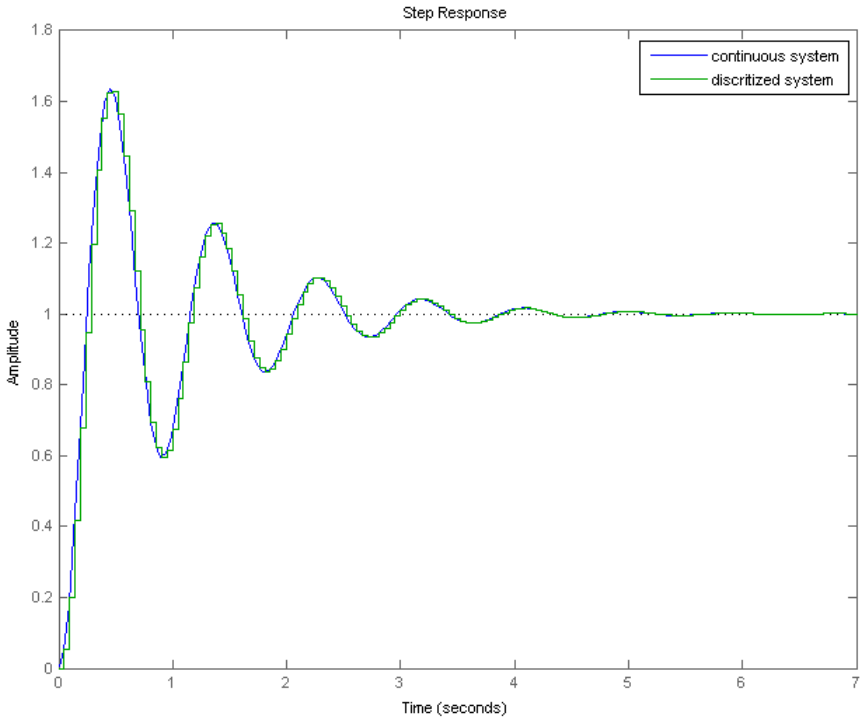
Discretize the following system, choosing a sampling period that accurately preserves its dynamic characteristics.

Solution. We should choose a sampling frequency that is at least 2 times higher than the natural frequency of the system . Since $\omega = 7$, a sampling time of

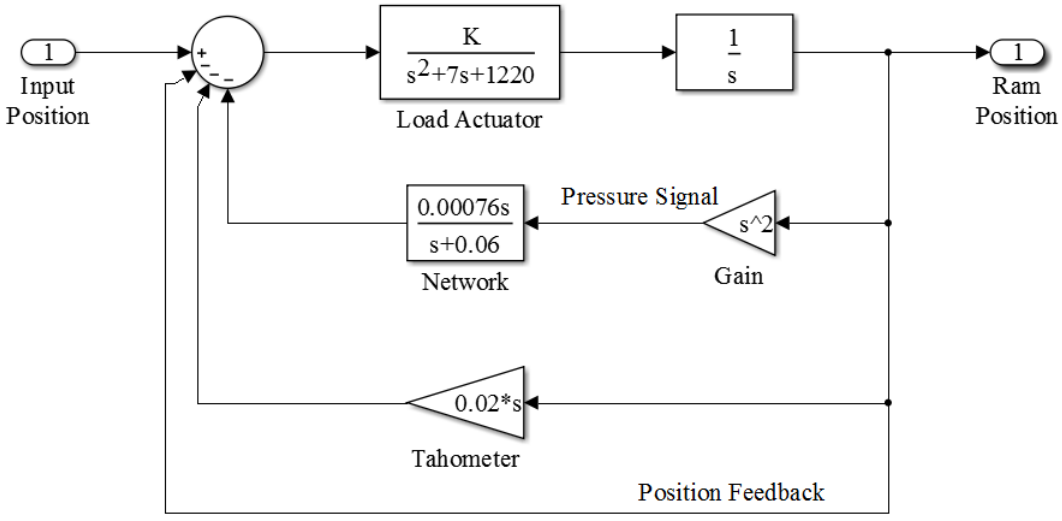
$$\omega_s = 3\omega \Rightarrow 1/T_s = 3\omega \Rightarrow T_s = 1/21 = 0.0476sec$$

will yield an accurate enough discrete time system.

```
sys=tf(49,[1 2 49])
sysd=c2d(sys,0.0476)
step(sys)
hold all
step(sysd)
legend('continuous system','discritized system')
```



Example 14.3.4. [Nise, 2013, Hardy, 1967] A six-degrees-of-freedom industrial robot equipped to perform arc welding and transfer objects according to a desired program can be described by the following system:



1. If K=64510, define the system in Matlab using either the commands `series, ... feedback` or Sisotool and find its transfer function.

2. Discretize the system using zero order hold for sampling time T=0.02sec.
3. Plot the step responses of the two systems on the same graph.

Solution. We will begin by first defining all the individual blocks in matlab and then calculate the system's transfer function.

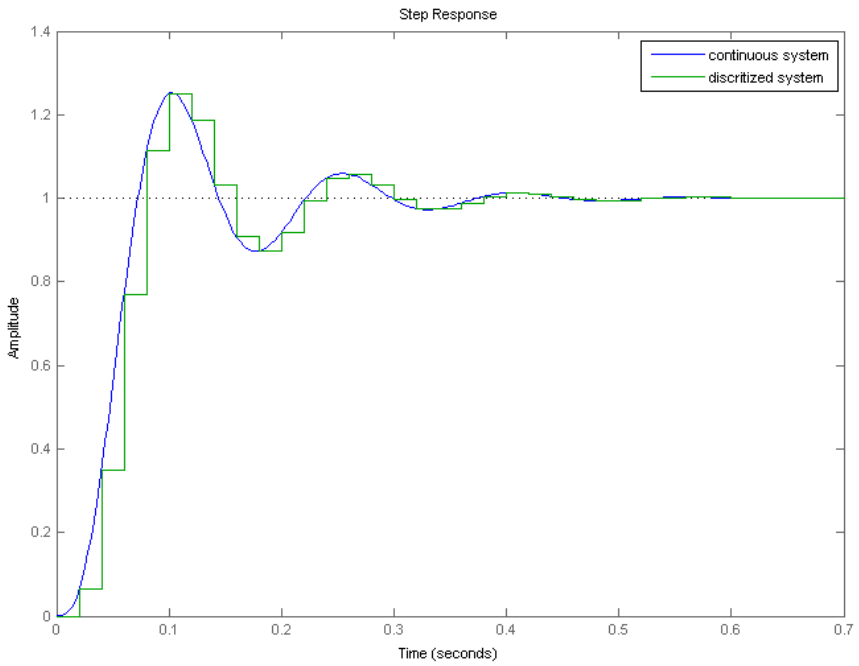
```
open_loop=tf([64510],[1 7 1220])*tf(1,[1 0]);
feedback_sys=tf([0.00076 0 0 0],[1 0.06])+ tf([0.02 0],1) +1;
% Closed loop system
closed_sys=feedback(open_loop,feedback_sys)
% Discretized system
closed_sys_d=c2d(closed_sys,0.02,'zoh')
% Step Responses
step(closed_sys)
hold all
step(closed_sys_d)
```

The resulting closed loop systems are

$$G(s) = \frac{64510(s+0.06)}{(s+36.09)(s+0.06)(s^2+19.94s+1788)}$$

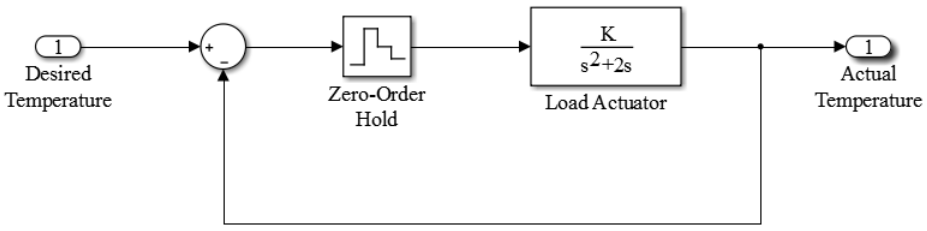
$$G(z) = \frac{0.063228(z+2.735)(z-0.9988)(z+0.2093)}{(z-0.9988)(z-0.4859)(z^2-1.116z+0.6711)}$$

and their step responses are the following



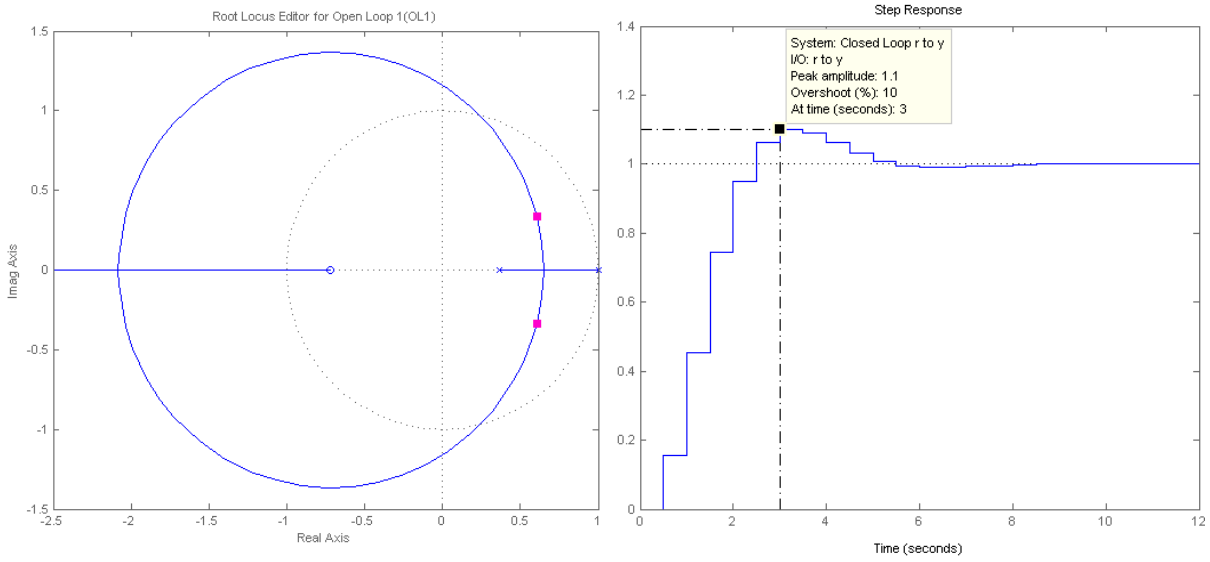
Example 14.3.5. [Nise, 2013, Dorf and Kusiak, 2007] Plastic extrusion is a well-established method widely used in the polymer processing industry. Such extruders typically consist of a large barrel divided into several temperature zones, with a hopper at one end and a die at the other. Polymer is fed into the barrel in raw and solid form from the hopper and is pushed forward by a powerful screw. Simultaneously, it is gradually heated while passing through the various temperature zones set in gradually increasing temperatures. The heat produced by the heaters in the barrel, together with the heat released from the friction between the raw polymer and the surfaces of the barrel and the screw, eventually causes the melting of the polymer, which is then pushed by the screw out from the die, to be processed further for various purposes (for an explanatory image, please check out the references).

The output variables are the outflow from the die and the polymer temperature. The main controlling variable is the screw speed, since the response of the process to it is rapid. The control system describing this procedure is the following



Discretize the system, choosing a small sample time and a gain such that the system has no more than 10% overshoot.

Solution. We will discretize the system using a sample time of T=0.5sec. Importing the system in sisotool, we find that the gain should be no more than about 1.6878.



Chapter 15

Nonlinear Systems

15.1 Introduction

In this chapter examples of nonlinear systems will be presented. In linear systems, it is always possible to find the explicit solution using analysis methods. This is not the case for nonlinear systems, where the computation of the solution can usually be achieved only using numerical analysis methods. Additionally, new questions arise regarding the existence of solution for given initial conditions.

Another notable feature of nonlinear systems is the high sensitivity to distrurbances in the initial conditions. A small change in the initial condition of such a system (i.e. due to rounding error) can yield a completely different response. This fact makes the long term prediction of real life nonlinear phenomena an extremely tedious task.

An extensive analysis of nonlinear system theory can be found in [Hirsch et al., 2013] and [Khalil, 2002]. Here, we shall solve examples of nonlinear systems using Matlab and Simulink and present the dynamic behavior of their responses.

15.2 Examples

Example 15.2.1 (The Zaslavsky Map)**.** The Zaslavsky Map [Zaslavsky, 1978] is a two dimensional discrete time dynamical system that exhibits chaotic behavior. It is described by the following nonlinear equations

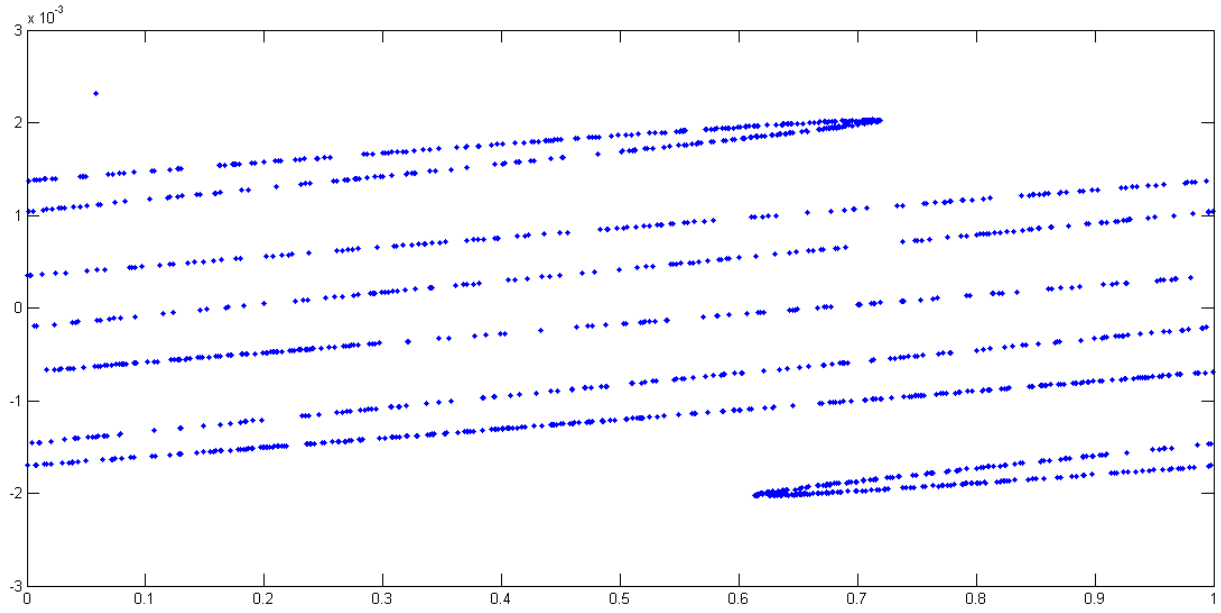
$$\begin{gathered} x_{n+1} = \big(x_n + \tfrac{\Omega}{2\pi} + (\tfrac{a\Omega}{2\pi\Gamma})(1-e^{-\Gamma})y_n + \tfrac{K}{\Gamma}(1-e^{-\Gamma})cos(2\pi x_n)\big) mod 1 \\ y_{n+1} = e^{-\Gamma}(y_n + \epsilon cos(2\pi x_n)) \end{gathered}$$

where $K = a\epsilon\Omega/2\pi$, Ω, Γ, ϵ are the parameters of the system. Plot the points (x_i, y_i) of the system for the first 1500 terms of the sequence, for $\Omega = 100$, $\epsilon = 0.3$, $\Gamma = 5$, $a = 1.885$, $K = 9$.

Solution.

```
%% The Zaslavsky Map

%% Defining initial conditions
x(1)=0.1;
y(1)=0.1;
e=0.3;
v=0.2;
r=5;
m=(1-exp(-r))/r;
omega=100;
k=9;
a=1.885;

%% Computing the values for x(i),y(i)
% We will compute the first 1500 values. They are more than enough to ...
   get a clear view of the result.
for i=2:1500
    x(i)=mod(x(i-1)+omega/(2*pi)+(a*omega)/(2*pi*r)*(1-exp(-r))*y(i-1)+...
        (k/r)*(1-exp(-r))*cos(2*pi*x(i-1)),1);
    % Since x(i) is computed mod1 we always have  0<=x(i)<1
    y(i)=exp(-r)*(y(i-1)+e*cos(2*pi*x(i-1)));
end

%% Plotting the results
plot(x,y,'.','MarkerSize',1.4)
axis([0 1 -0.003 0.003])
```

The map of the points (x_i, y_i) is the following



Example 15.2.2 (The Henon Map)**.** The Henon Map [Hénon, 1976] is a discrete time system that maps a point (x_n, y_n) using the following formula

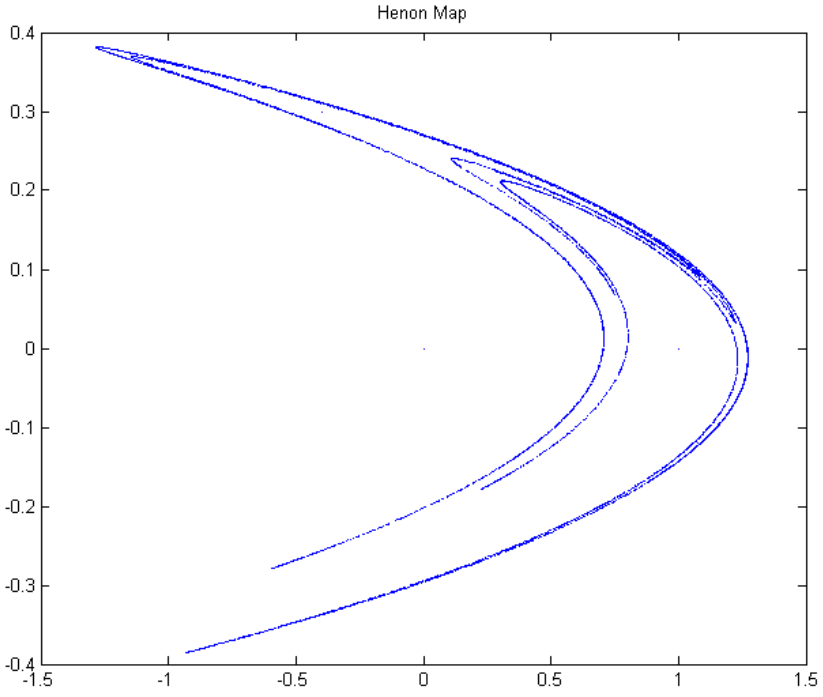
$$\begin{aligned} x_{n+1} &= 1 - ax_n^2 + y_n \\ y_{n+1} &= bx_n \end{aligned} \tag{15.1}$$

Where a and b are the system parameters. Compute and plot the first 10000 points of this mapping for a=1.4 and b=0.3.

Solution.

```
%% We start by defining the initial values
x(1)=0;
y(1)=0;
a=1.4;
b=0.3;

%% Now we begin the iteration (10000 iterations):
for i=2:10000
    x(i)=1-1.4*(x(i-1)^2)+y(i-1);
    y(i)=b*x(i-1);
end
plot(x,y,'.','MarkerSize',4)
title('Henon Map')
```

The plot of the above mapping is

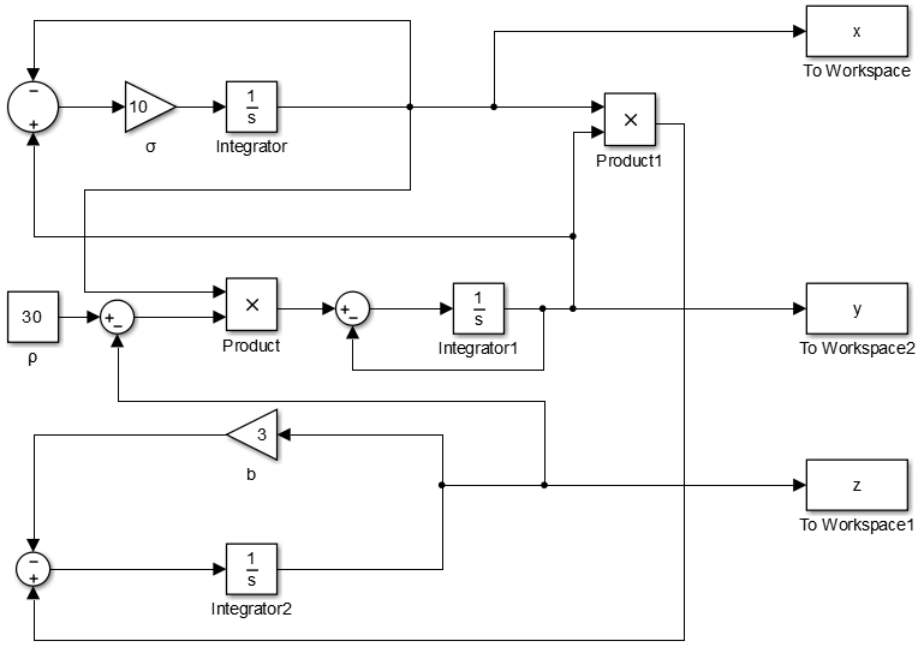


Example 15.2.3 (The Lorenz Attractor)**.** The Lorenz Attractor [Hirsch et al., 2013] is a famous nonlinear three dimensional system of differential equations that finds application in biology, circuit theory, mechanics, lasers and chemical reactions. The system is described by the following differential equations

$$\begin{aligned} x'(t) =& \sigma\big(y(t) - x(t)\big) \\ y'(t) =& x(t)(\rho - z(t)) - y(t) \\ z'(t) =& \beta z(t) + x(t)y(t) \end{aligned}$$

Where σ, ρ, β are the system parameters. Design the system in simulink and plot the points (x_i, y_i, z_i) for $\sigma = 10$, $\rho = 30$, $\beta = -3$.

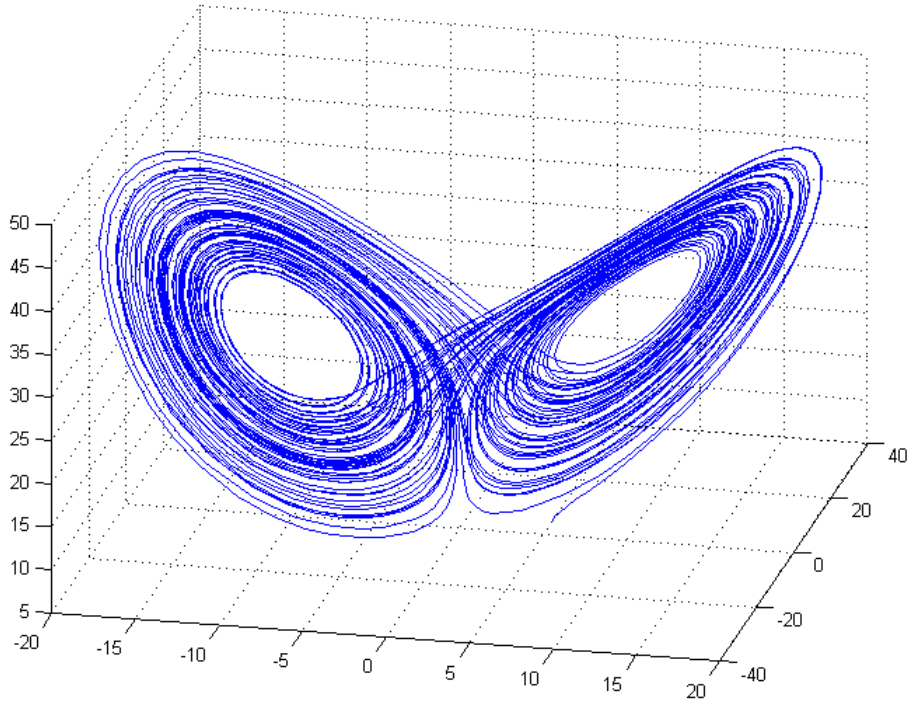
Solution. In order to solve the system, we choose here to design it first in Simulink and save the simulation results in variables. For the parameter blocks σ, ρ, β we can choose different values any time we run a simulation.



In order for the simulation results to be smooth, it is critical that we change the Refine Factor from Simulation Parameters of the model equal to 4. After the simulation is over (we can choose 70sec), we use the following commands to produce a rottating plot for the points (x_i, y_i, z_i).

```
view(3)
axis([-20,20,-30,30,0,50])
f=plot3(x(1),y(1),z(1));
```

```
for i=2:length(tout)
    f=plot3(x(1:i),y(1:i),z(1:i));
    grid
    view(-37.5+i, 24)
    pause(0.01)
end
```

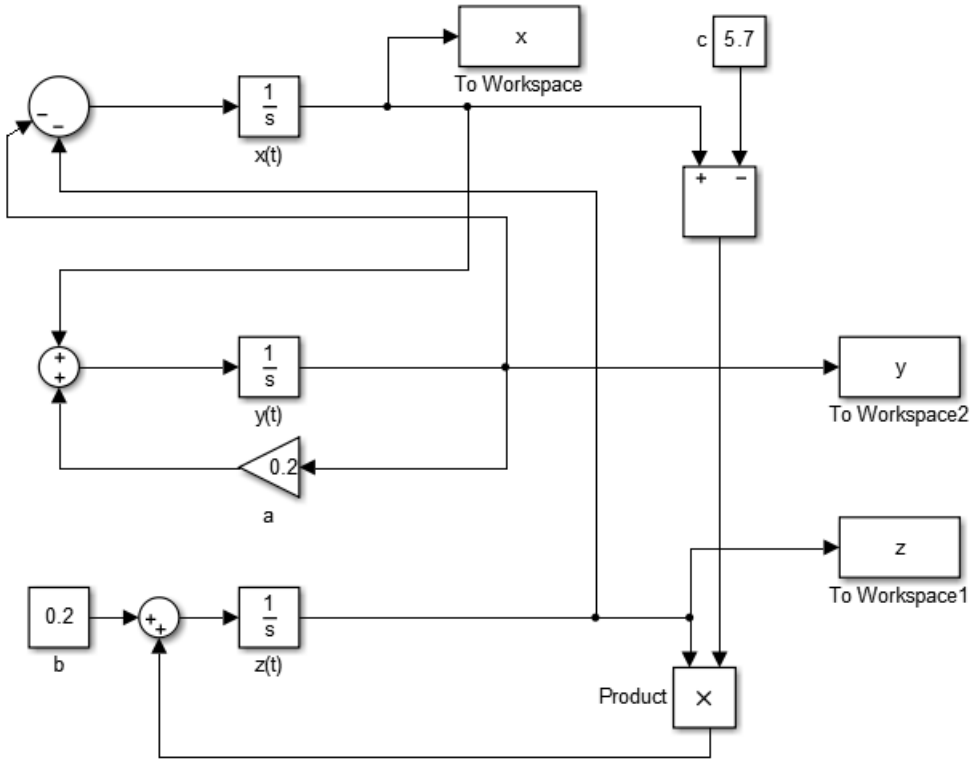


Example 15.2.4 (The Rössler Attractor)**.** The Rössler Attractor [Rössler, 1976] is another three dimensional system of nonlinear differential equations

$$\begin{aligned} x'(t) =& -y(t) - z(t) \\ y'(t) =& x(t) + ay(t) \\ z'(t) =& b + z(t)\big(x(t) - c\big) \end{aligned}$$

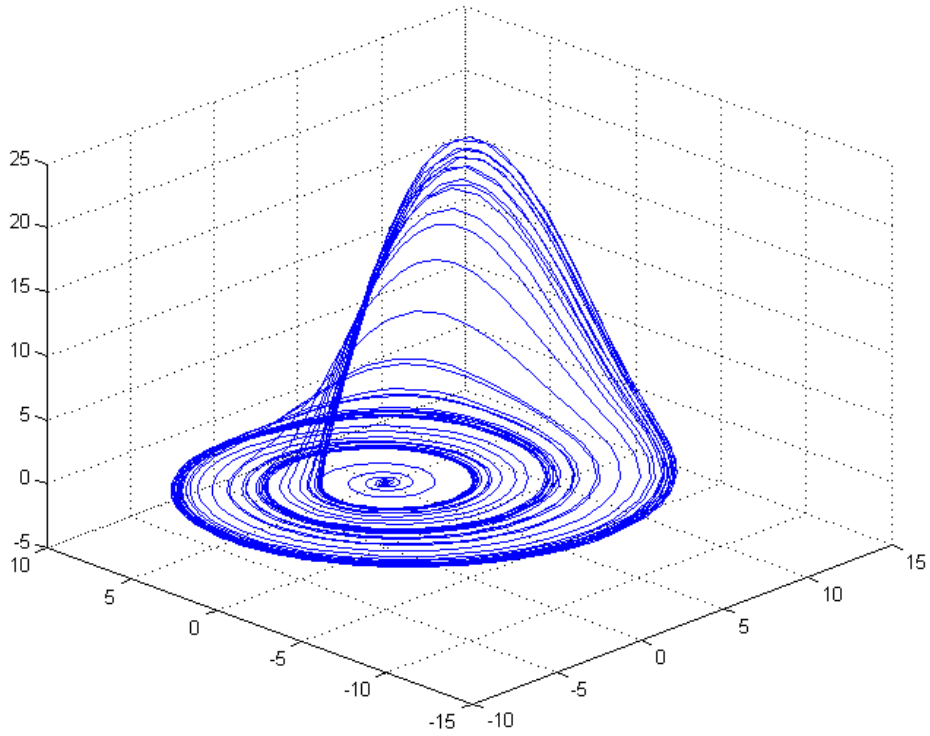
Where $a = 0.2$, $b = 0.2$, $c = 5.7$ are the system parameters. Design this system in matlab and plot the points (x_i, y_i, z_i).

Solution. Again changing the Refine Factor to 4 we design the nonlinear equations and simulate the system for time T=300.



Again, we use the following commands to produce a rottating plot for (x_i, y_i, z_i).

```
view(3)
axis([-10,15,-15,10,-5,25])
f=plot3(x(1),y(1),z(1));

for i=2:length(tout)
    f=plot3(x(1:i),y(1:i),z(1:i));
    grid
    view(-37.5+i, 24)
    pause(0.01)
end
```

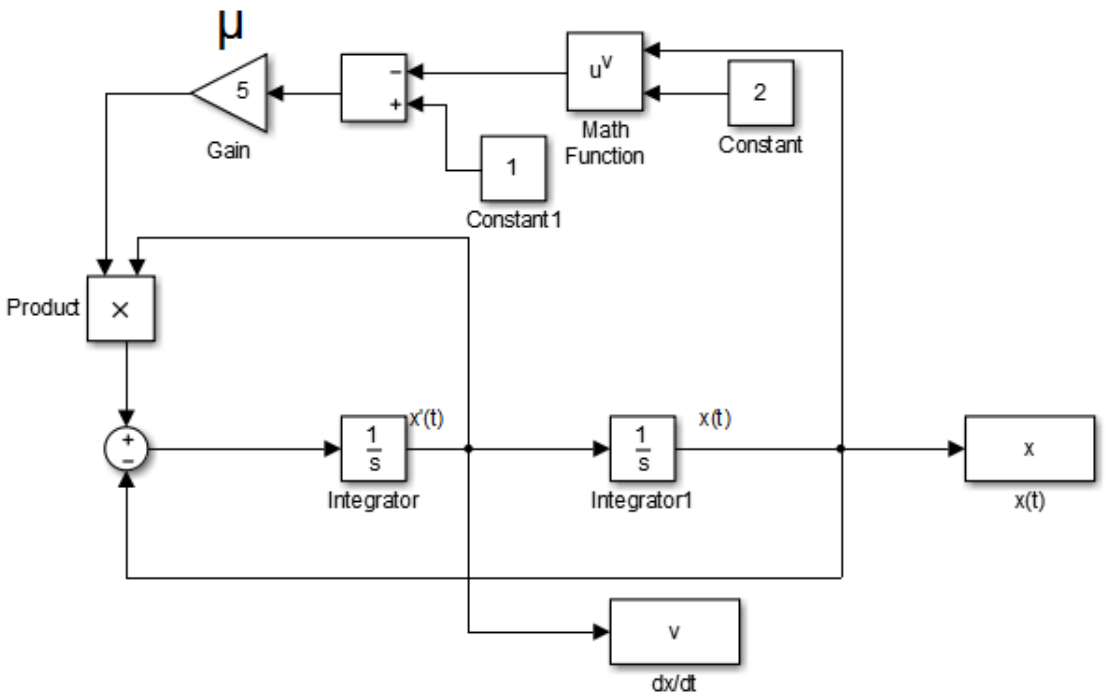


Example 15.2.5 (Van der Pol Oscillator)**.** The Van der Pol Oscillator [van der Pol, 1926] is a nonlinear equation that finds applications in various scientific fields like biology and seismology. It is described by the following differential equation

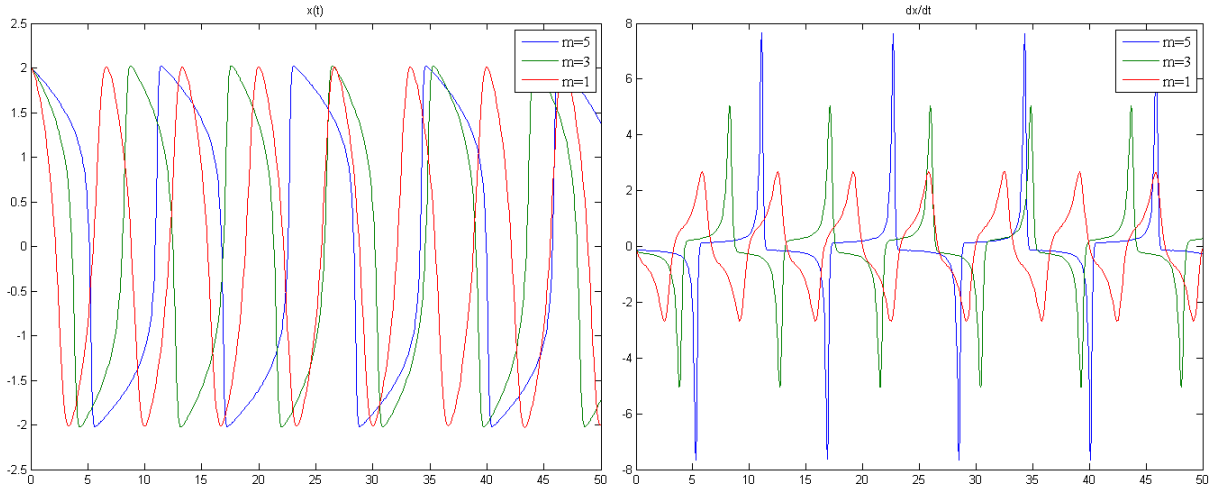
$$x''(t) - \mu(1 - x(t)^2)x'(t) + x(t) = 0 \tag{15.2}$$

From the above equation it is clear that for $\mu = 0$ the equation becomes linear. Simulate the system for different values of μ and plot the values $x(t)$, $x'(t)$ against time, as well as $(x(t), x'(t))$.

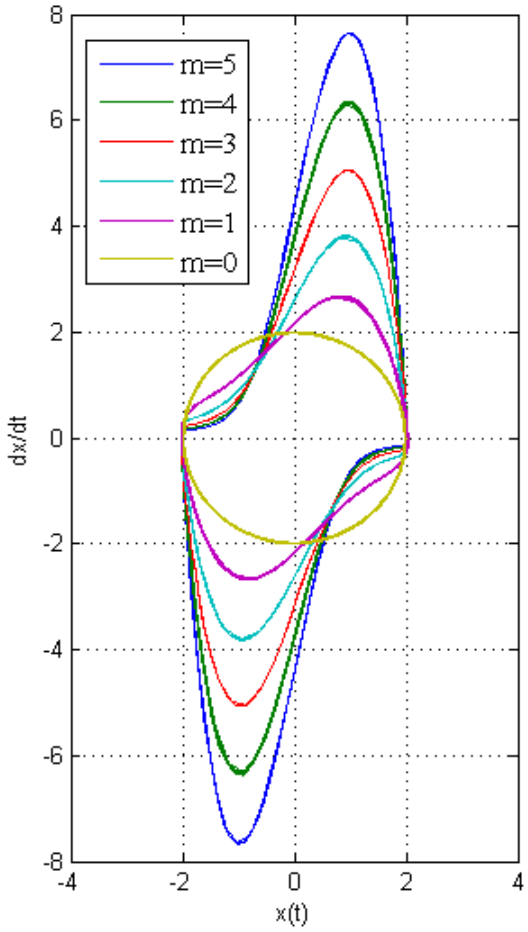
Solution. Again changing the Refine Factor to 4 we design the nonlinear equations and simulate the system for time T=200sec.



The functions $x(t)$ and $x'(t)$ for $T = 50sec$ can be seen in the following figures



While the oscillation between these two functions for different values of μ can be seen in the following figure


m=5
m=4
m=3
m=2
m=1
m=0
dx/dt
x(t)
8
6
4
2
0
-2
-4
-6
-8
-4
-2
0
2
4

Acknowledgments

This book was typeset by the authors using LaTeX, a free package of macro commands based on TeX, originally created by Leslie Lamport. The official page is http://www.latex-project.org/

The authors would also like to acknowledge Florian Knorn for the creation of the `mcode` package, which provided a handy way of visualizing Matlab code. It can be found on Matlab Central.

Last but not least, many thanks goes to the online LaTeX community of Stack-Exchange http://tex.stackexchange.com/ for being an encyclopaedia for LaTeX debugging and problem solving.

Bibliography

[Antsaklis and Michel, 2006] Antsaklis, P. J. and Michel, A. N. (2006). *Linear systems.* Boston: Birkhäuser, 2nd corrected printing edition.

[Beucher and Weeks, 2008] Beucher, O. and Weeks, M. (2008). *Introduction to MATLAB & SIMULINK: A Project Approach.* Engineering series. Infinity Science Press.

[Brian Douglas, 2013] Brian Douglas (2013). Videos on the nyquist stability criterion. https://www.youtube.com/watch?v=sof3meN96MA. Accessed: 10-07-2015.

[Cavallo et al., 1992] Cavallo, A., Maria, G. D., and Verde, L. (1992). Robust flight control systems - a parameter space design. *Journal of Guidance, Control, and Dynamics*, 15(5):1207–1215.

[Chaturvedi, 2010] Chaturvedi, D. K. (2010). *Modeling and simulation of systems using MATLAB and Simulink.* Boca Raton, FL: CRC Press.

[Cheever, 2005] Cheever, E. (2005). The nyquist plot online course. http://lpsa.swarthmore.edu/Nyquist/Nyquist.html. Accessed: 10-07-2015.

[de Panne et al., 1992] de Panne, V., Fiume, and Vranesic (1992). A controller for the dynamic walk of a biped across variable terrain. In *Proceedings of the 31st IEEE Conference on Decision and Control*, pages 2668–2673 vol.3.

[Digilent, 2010] Digilent (2010). Digital notes: Introduction to first order system responses. http://www.digilentinc.com/classroom/realanalog/text/Chapter_2p4p1.pdf. Accessed: 10-07-2015.

[Dorf and Kusiak, 2007] Dorf, R. and Kusiak, A. (2007). *Handbook of Design, Manufacturing and Automation.* John Wiley & Sons, Inc.

[Dorf and Bishop, 2009] Dorf, R. C. and Bishop, R. H. (2009). *Modern control systems.* Pearson Education, 11th edition.

[Elarafi and Hisham, 2008] Elarafi, M. and Hisham, S. (2008). Modeling and control of pH neutralization using neural network predictive controller. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 1196–1199.

[Fadali, 2015] Fadali, M. (2015). Root locus review. http://wolfweb.unr.edu/~fadali/ee472/RLocusRev.pdf. Accessed: 10-07-2015.

[Hardy, 1967] Hardy, H. (1967). Multi-Loop Servo Controls Programmed Robot. *Instruments and Control Systems*, page 105?111.

[Hénon, 1976] Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Commun. Math. Phys.*, 50:69–76.

[Hespanha, 2009] Hespanha, J. P. (2009). *Linear systems theory.* Princeton, NJ: Princeton University Press.

[Hirsch et al., 2013] Hirsch, M. W., Smale, S., and Devaney, R. L. (2013). *Differential equations, dynamical systems, and an introduction to chaos. 3rd ed.* Amsterdam: Academic Press, 3rd ed. edition.

[Hong, 2015] Hong, X. (2015). Introduction to the nyquist criterion. http://www.personal.reading.ac.uk/~sis01xh/teaching/CY2A9/Lecture7.pdf. Accessed: 10-07-2015.

[Houcque, 2005] Houcque, D. (2005). *INTRODUCTION TO MATLAB FOR ENGINEERING STUDENTS, Northwestern University.*

[Jiayu et al., 2009] Jiayu, K., Mengxiao, W., Linan, M., and Zhongjun, X. (2009). Cascade Control of the PH in an Anaerobic Wastewater Treatment System. In *Bioinformatics and Biomedical Engineering , 2009. ICBBE 2009. 3rd International Conference on*, pages 1–4.

[Kailath, 1980] Kailath, T. (1980). *Linear Systems.* Information and System Sciences Series. Prentice-Hall.

[Kappos, 2002] Kappos, E. (2002). *Classical Control Theory: A Course in the Linear Mathematics of Systems and Control.*

[Karris, 2011] Karris, S. (2011). *Introduction to Simulink with Engineering Applications.* Orchard Publications.

[Kelley and Peterson, 2001] Kelley, W. and Peterson, A. (2001). *Difference equations. An introduction with applications. 2nd ed.* San Diego, CA: Harcourt/Academic Press, 2nd ed. edition.

[Khalil, 2002] Khalil, H. K. (2002). *Nonlinear systems. 3rd ed.* Upper Saddle River, NJ: Prentice Hall, 3rd ed. edition.

[Kim et al., 2009] Kim, S., Kim, J., Yang, J., Yang, H., Park, J., and Park, Y. (2009). Tilt Detection and Servo Control Method for the Holographic Data Storage System. *Microsyst Technol*, 15:1695?1700.

[Kuo et al., 2008] Kuo, C., Tsai, C., and Tu, H. (2008). Carriage speed control of a cross-lapper system for nonwoven web quality. *Fibers and Polymers*, 9(4):495–502.

[Liceaga-Castro and van der Molen, 1995] Liceaga-Castro, E. and van der Molen, G. (1995). Submarine H_∞ depth control under wave disturbances. *Control Systems Technology, IEEE Transactions on*, 3(3):338–346.

[Linkens, 1992] Linkens, D. (1992). Adaptive and intelligent control in anesthesia. *Control Systems, IEEE*, 12(6):6–11.

[Liu et al., 2008] Liu, J.-H., ping Xu, D., and Yang, X.-Y. (2008). Multi-objective power control of a variable speed wind turbine based H_∞ on theory. In *Machine Learning and Cybernetics, 2008 International Conference on*, volume 4, pages 2036–2041.

[M. Vajta, 2000] M. Vajta (2000). Some remarks on pade approximations. In *Proceedings of the 3rd TEMPUS-INTCOM Symposium, September 9-14, 2000, Veszpr?m, Hungary*. Accessed: 10-07-2015.

[Magrab, 2011] Magrab, E. (2011). *An Engineer's Guide to MATLAB: With Applications from Mechanical, Aerospace, Electrical, Civil, and Biological Systems Engineering*. Pearson.

[Martin, 1999] Martin, F. (1999). *The Art of Robotics*. Prentice Hall.

[Mastascusa, 2015] Mastascusa, E. (2015). Online notes on nyquist plots, bucknell university course on electrical control systems. http://www.facstaff.bucknell.edu/mastascu/econtrolhtml/Freq/Freq6.html. Accessed: 10-07-2015.

[Mathworks, 2015] Mathworks (2015). Mathworks documentation center. http://www.mathworks.com/help/. Accessed: 10-07-2015.

[McMahon, 2007] McMahon, D. (2007). *MATLAB Demystified*. McGraw-Hill Education.

[Messner and Tilbury, 2015] Messner, B. and Tilbury, D. (2015). Control tutorials for matlab and simulink. introduction: System analysis. http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SystemAnalysis#8. Accessed: 10-07-2015.

[Nise, 2013] Nise, N. S. (2013). *Control Systems Engineering*. John Wiley & Sons, Inc., New York, NY, USA, 6th edition.

[Ogata, 2009] Ogata, K. (2009). *Modern control engineering*. Prentice Hall, 5th ed. edition.

[Ougrinovski, 2006] Ougrinovski, V. (2006). Nyquist stability criterion. http://seit.unsw.adfa.edu.au/staff/sites/valu/teaching/ct2/nyquist.pdf. Accessed: 10-07-2015.

[Palm III, 2011] Palm III, W. (2011). *Introduction to MATLAB for Engineers*. McGraw-Hill, 3rd edition.

[Paraskevopoulos, 1996] Paraskevopoulos, P. (1996). *Digital Control Systems*. Prentice Hall.

[Paraskevopoulos, 2001] Paraskevopoulos, P. (2001). *Modern Control Engineering*. Automation and Control Engineering. Taylor & Francis.

[Rossiter, 2015] Rossiter, J. A. (2015). Videos on classical control analysis methods, department of automatic control and systems engineering, university of sheffield. http://www.sheffield.ac.uk/acse/staff/jar/bookmaster. Accessed: 10-07-2015.

[Rössler, 1976] Rössler, O. (1976). An equation for continuous chaos. *Physics Letters A*, 57(5):397 – 398.

[Schneider, 1992] Schneider, R. (1992). Pneumatic robots continue to improve. *Hydraulics & Pneumatics*.

[Stefani, 1973] Stefani, R. (1973). Modeling human response characteristics. *COED Application Note No. 33. Computers in Education Division ofASEE*.

[Tari et al., 2005] Tari, C., Teufel, E., Pico, J., Bondia, J., and Pfleiderer, H. (2005). Comprehensive pharmacokinetic model of insulin glargine and other insulin formulations. *Biomedical Engineering, IEEE Transactions on*, 52(12):1994–2005.

[Tewari, 2002] Tewari, A. (2002). *Modern Control Design: With MATLAB and SIMULINK*. Wiley.

[Thomas et al., 2005] Thomas, B., Soleimani-Mosheni, M., and Fahlén, P. (2005). Feedforward in Temperature Control of Buildings. *Energy and Buildings*, 37:755?761.

[Ton van den Boom, 2006] Ton van den Boom (2006). Discrete-time systems analysis, lecture notes on the course of Model Predictive Control, Delft Un. of Technology. http://www.dcsc.tudelft.nl/~sc4060/transp/discreteNOTES.pdf. Accessed: 10-07-2015.

[V. Hanta and A. Prochazka, 2009] V. Hanta and A. Prochazka (2009). Rational approximation of time delay. https://www.researchgate.net/publication/237396133_RATIONAL_APPROXIMATION_OF_TIME_DELAY. Accessed: 10-07-2015.

[van der Pol, 1926] van der Pol, B. (1926). Lxxxviii. on relaxation-oscillations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):978–992.

[Yan and Lin, 2003] Yan, T. and Lin, R. (2003). Experimental modeling and compensation of pivot nonlinearly in hard disk drives. *IEEE Transactions on Magnetics*, 39:1064–1069.

[Zamboni, 2013] Zamboni, L. (2013). *Getting Started with Simulink*. Packt Publishing.

[Zaslavsky, 1978] Zaslavsky, G. (1978). The simplest case of a strange attractor. *Physics Letters A*, 69(3):145 – 147.