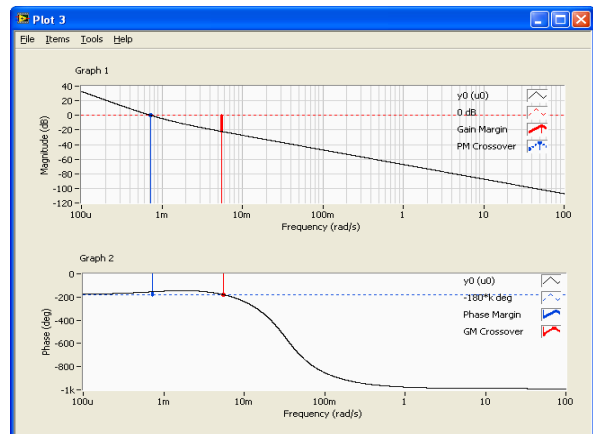
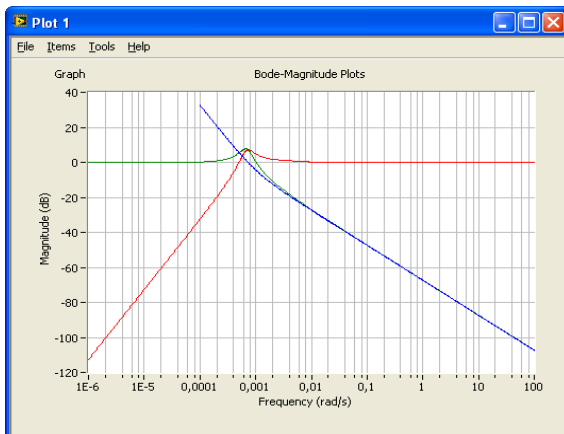


# Control Theory with MathScript Examples

Hans-Petter Halvorsen, 2016.10.26



# Table of Contents

Table of Contents .....	ii
1 Introduction.....	1
1.1 MathScript.....	1
2 MathScript Basics .....	2
2.1 Introduction.....	2
2.2 How do you start using MathScript? .....	2
2.3 Basic Operations.....	3
2.4 Vectors and Matrices .....	4
2.5 Linear Algebra .....	7
2.6 Plotting .....	8
2.6.1 Subplots.....	9
2.7 User-Defined Functions in MathScript .....	12
2.8 Scripts.....	12
2.9 Flow Control .....	15
2.10 Control Design in MathScript .....	15
3 Transfer Functions .....	17
3.1 Introduction.....	17
3.1.1 1.order system.....	17
3.1.2 1.order system with time-delay .....	18
3.1.3 2.order system.....	18
3.2 MathScript.....	19
3.3 First order Transfer Function.....	21
3.3.1 1.order system with time-delay .....	23

---

3.4	Second order Transfer Function .....	25
3.5	Simulation .....	26
3.6	Block Diagrams .....	26
3.7	Analysis of Standard Functions.....	27
3.7.1	Integrator .....	28
3.7.2	1. order system.....	30
3.7.3	2. order system.....	33
4	State-space Models .....	35
4.1	Introduction.....	35
4.2	MathScript.....	36
5	Time-delay and Pade'-approximations.....	39
5.1	Introduction.....	39
5.2	MathScript.....	40
6	Stability Analysis .....	45
6.1	Introduction.....	45
6.2	Poles .....	46
6.3	Feedback Systems .....	49
6.3.1	Loop Transfer function .....	49
6.3.2	Tracking transfer function .....	50
6.3.3	Sensitivity transfer function .....	50
6.3.4	Characteristic Polynomial.....	50
7	Frequency Response.....	52
7.1	Introduction.....	52
7.2	MathScript.....	53
7.3	Examples .....	54
7.4	Standard Transfer functions.....	61
7.4.1	Amplifier (Norwegian: "Forsterker"):	61

---

7.4.2	Integrator .....	62
7.4.3	Derivator.....	63
7.4.4	1. Order system .....	64
7.4.5	2. Order system .....	65
7.4.6	Zero part (Norwegian: “Nullpunktsledd”) .....	66
7.4.7	Time delay (Norwegian: “Tidsforsinkelse”) .....	66
8	Frequency response Analysis.....	68
8.1	Introduction.....	68
8.2	MathScript.....	70
9	Stability Analysis in the Frequency Domain.....	74
9.1	Introduction.....	74
9.2	MathScript.....	75
Appendix A – MathScript Functions .....		80
Basic Functions .....		80
Basic Plotting Functions .....		80
Functions used for Control and Simulation .....		81

# 1 Introduction

This document presents some control theory and lots of examples of how you may implement it in MathScript.

This document gives an introduction to the following topics:

- Transfer Functions and Block Diagrams
- State-Space Models
- Time-delay and Pade' approximations
- Frequency Response
- Frequency Response Analysis
- Stability Analysis

MathScript has lots of built-in functionality for these applications. In each chapter we will give a short overview to the theory behind, before we dig into the MathScript Examples. Since MathScript is almost identical to MATLAB, you can use MATLAB instead in most of the examples shown.

## 1.1 MathScript

MathScript is a high-level, text-based programming language. MathScript includes more than 800 built-in functions and the syntax is similar to MATLAB. You may also create custom-made m-file like you do in MATLAB.

MathScript is well suited for practical implementations of control theory.

MathScript (**LabVIEW MathScript RT Module**) is an add-on module to LabVIEW but you don't need to know LabVIEW programming in order to use MathScript, because MathScript is a text-based language similar to MATLAB.

For more information about MathScript, please read the Tutorial "[LabVIEW MathScript](http://home.hit.no/~hansha/?tutorial=mathscript)" (<http://home.hit.no/~hansha/?tutorial=mathscript>).

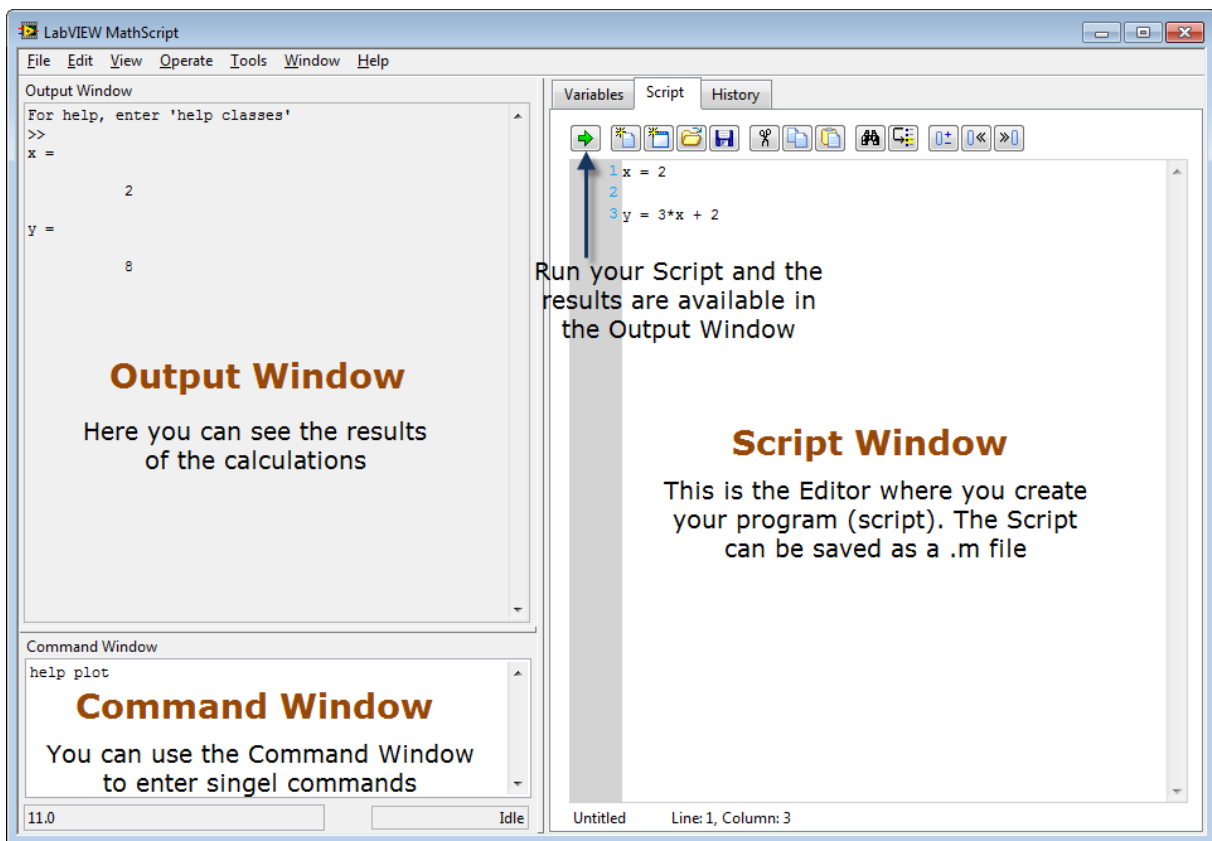
Additional exercises are given in the course "[So You Think You Can MathScript](http://home.hit.no/~hansha/?lab=mathscript)" (<http://home.hit.no/~hansha/?lab=mathscript>).

# 2 MathScript Basics

## 2.1 Introduction

MathScript is a high-level, text-based programming language. MathScript includes more than 800 built-in functions and the syntax is similar to MATLAB. You may also create custom-made m-file like you do in MATLAB.

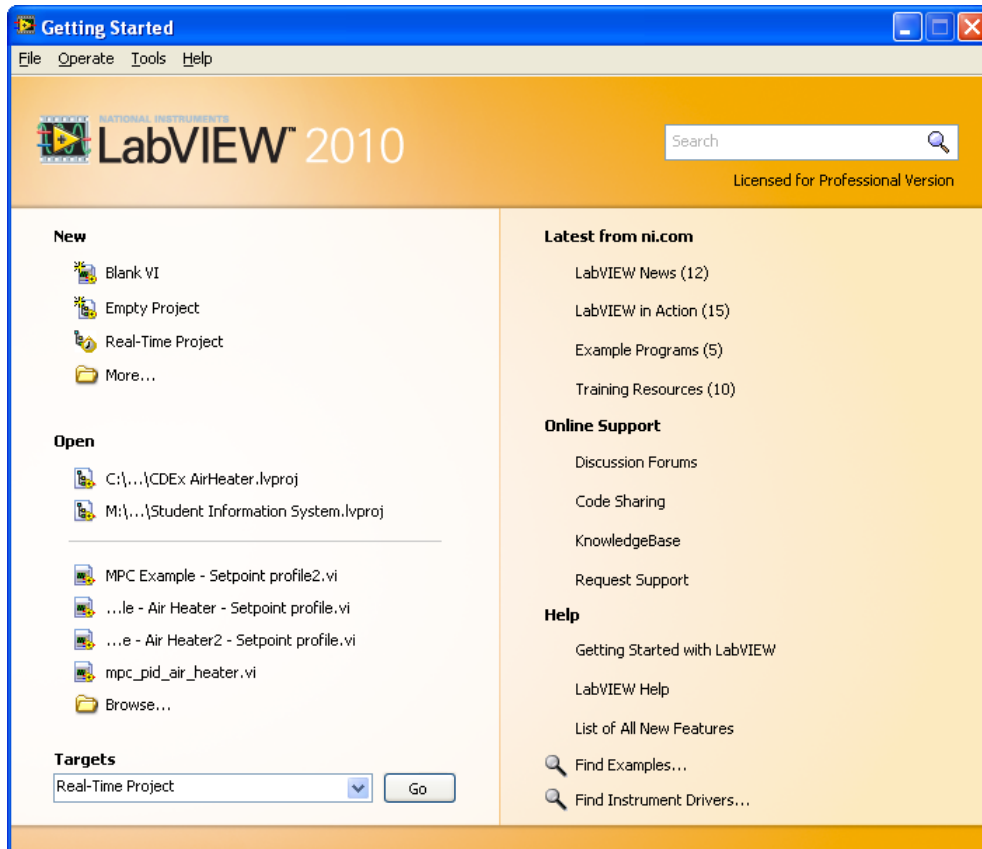
MathScript is an add-on module to LabVIEW but you don't need to know LabVIEW programming in order to use MathScript.



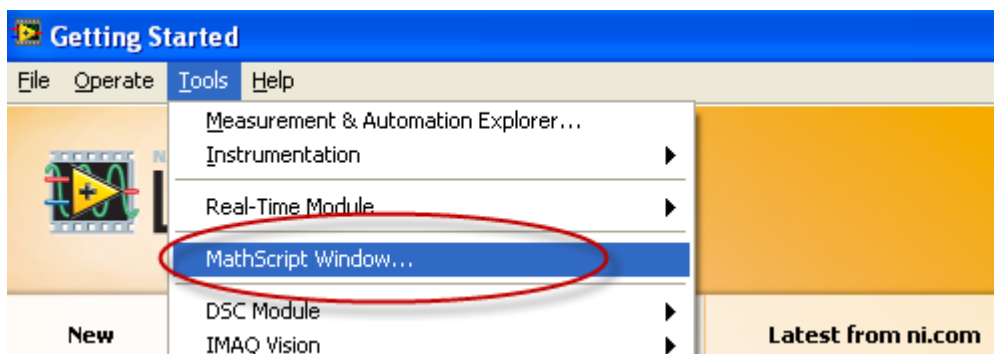
For more information about MathScript, please read the Tutorial "[LabVIEW MathScript](#)".

## 2.2 How do you start using MathScript?

You need to install [LabVIEW](#) and the [LabVIEW MathScript RT Module](#). When necessary software is installed, start MathScript by open LabVIEW:



In the **Getting Started** window, select **Tools -> MathScript Window...**:



## 2.3 Basic Operations

### Variables:

Variables are defined with the assignment operator, “=”. MathScript is dynamically typed, meaning that variables can be assigned without declaring their type, and that their type can change. Values can come from constants, from computation involving values of other variables, or from the output of a function.

### Example:

```
>> x = 17
x =
  17
>> x = 'hat'
x =
hat
>> x = [3*4, pi/2]
x =
  12.0000    1.5708
>> y = 3*sin(x)
y =
 -1.6097    3.0000
```

[End of Example]

**Note!** MathScript is case sensitive! The variables  $x$  and  $X$  are not the same.

**Note!** Unlike many other languages, where the semicolon is used to terminate commands, in MathScript the semicolon serves to suppress the output of the line that it concludes.

Try the following:

```
>> a=5
a =
   5
>> a=6;
>>
```

As you see, when you type a semicolon (;) after the command, MathScript will not respond.

It is normal it enter one command in each line, like this:

```
x = [0:0.1:1];
y = sin(x)
```

But we can also enter more than one command on one line:

```
x = [0:0.1:1]; y = sin(x)
```

or:

```
x = [0:0.1:1], y = sin(x)
```

## 2.4 Vectors and Matrices

**Vectors:**



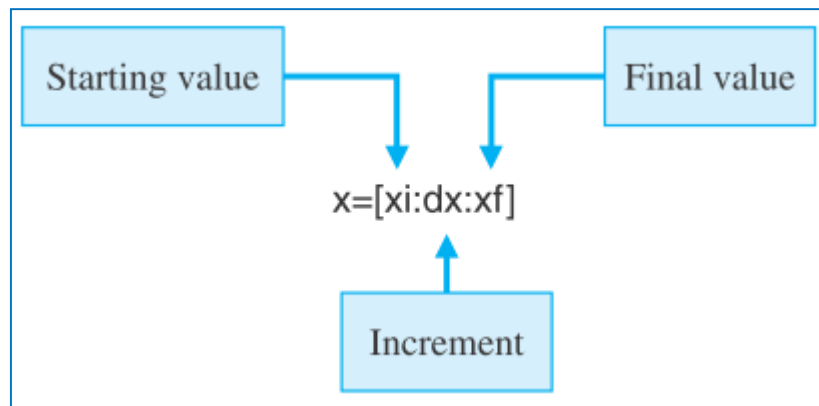
Given the following vector:

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

This can be implemented in MathScript like this:

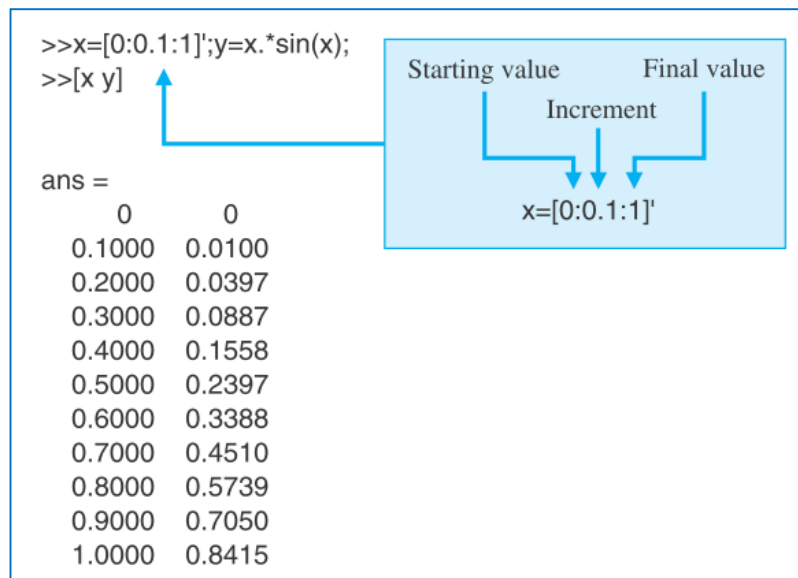
```
x = [1 2 3]
```

The “**colon notation**” is very useful for creating vectors:



### Example:

This example shows how to use the colon notation creating a vector and do some calculations.



### Matrices:

Given the following matrix:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

MathScript Code:

```
A=[0 1; -2 -3]
```

Given the following matrix:

$$C = \begin{bmatrix} -1 & 2 & 0 \\ 4 & 10 & -2 \\ 1 & 0 & 6 \end{bmatrix}$$

MathScript Code:

```
C=[-1 2 0; 4 10 -2; 1 0 6]
```

### How to get a subset of a matrix:

→ Find the value in the second row and the third column of matrix C:

```
C(2,3)
```

This gives:

```
ans =
     -2
```

→ Find the second row of matrix C:

```
C(2,:)
```

This gives:

```
ans =
     4     10     -2
```

→ Find the third column of matrix C:

```
C(:,3)
```

This gives:

```
ans =
     0
    -2
     6
```

**Deleting Rows and Columns:**

You can delete rows and columns from a matrix using just a pair of square brackets [].

**Example:**

Given

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

We define the matrix A:

```
>>A=[0 1; -2 -3];
```

To delete the second column of a matrix A, we use:

```
>>A(:,2) = []
A =
     0
    -2
```

[End of Example]

## 2.5 Linear Algebra

Linear algebra is a branch of mathematics concerned with the study of matrices, vectors, vector spaces (also called linear spaces), linear maps (also called linear transformations), and systems of linear equations.

MathScript are well suited for Linear Algebra. Here are some useful functions for Linear Algebra in MathScript:

Function	Description	Example
<b>rank</b>	Find the rank of a matrix. Provides an estimate of the number of linearly independent rows or columns of a matrix A.	>>A=[1 2; 3 4] >>rank(A)
<b>det</b>	Find the determinant of a square matrix	>>A=[1 2; 3 4] >>det(A)
<b>inv</b>	Find the inverse of a square matrix	>>A=[1 2; 3 4] >>inv(A)
<b>eig</b>	Find the eigenvalues of a square matrix	>>A=[1 2; 3 4] >>eig(A)
<b>ones</b>	Creates an array or matrix with only ones	>>ones(2) >>ones(2,1)
<b>eye</b>	Creates an identity matrix	>>eye(2)
<b>diag</b>	Find the diagonal elements in a matrix	>>A=[1 2; 3 4] >>diag(A)

Type “**help matfun**” (Matrix functions - numerical linear algebra) in the Command Window for more information, or type “**help elmat**” (Elementary matrices and matrix manipulation).

You may also type “**help <functionname>**” for help about a specific function.

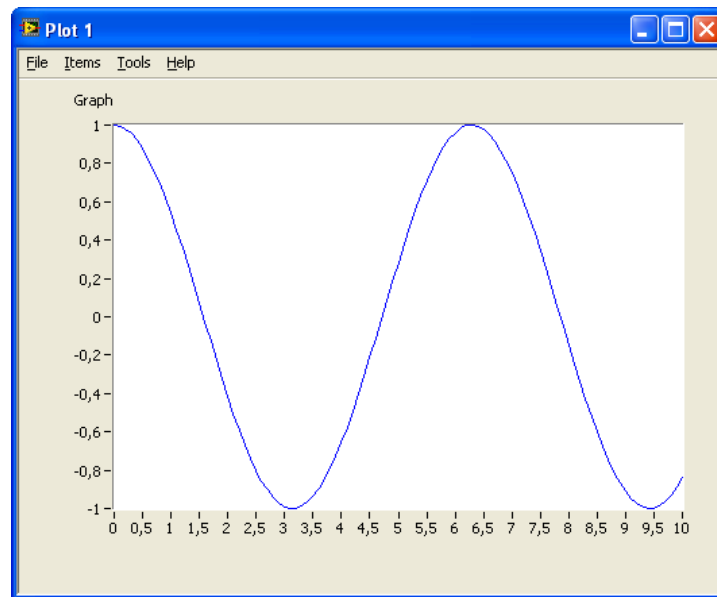
## 2.6 Plotting

MathScript has lots of functionality for Plotting. The simplest and most used is the **plot** function.

### Example:

```
>>t=[0:0.1:10];
>>y=cos(t);
>>plot(t,y)
```

This gives the following plot:



[End of Example]

MathScript has lots of built-in functions for plotting:

Function	Description	Example
<b>plot</b>	Generates a plot. <code>plot(y)</code> plots the columns of <code>y</code> against the indexes of the columns.	<pre>&gt;&gt;X = [0:0.01:1]; &gt;&gt;Y = X.*X; &gt;&gt;plot(X, Y)</pre>
<b>figure</b>	Create a new figure window	<pre>&gt;&gt;figure &gt;&gt;figure(1)</pre>
<b>subplot</b>	Create subplots in a Figure. <code>subplot(m,n,p)</code> or <code>subplot(mnp)</code> , breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot. The axes are counted along the top row of the Figure window, then the second row, etc.	<pre>&gt;&gt;subplot(2,2,1)</pre>
<b>grid</b>	Creates grid lines in a plot. "grid on" adds major grid lines to the current plot. "grid off" removes major and minor grid lines from the current plot.	<pre>&gt;&gt;grid &gt;&gt;grid on &gt;&gt;grid off</pre>
<b>axis</b>	Control axis scaling and appearance. "axis([xmin xmax ymin ymax])" sets the limits for the x- and y-axis of the current axes.	<pre>&gt;&gt;axis([xmin xmax ymin ymax]) &gt;&gt;axis off &gt;&gt;axis on</pre>
<b>title</b>	Add title to current plot <code>title('string')</code>	<pre>&gt;&gt;title('this is a title')</pre>
<b>xlabel</b>	Add xlabel to current plot	<pre>&gt;&gt; xlabel('time')</pre>

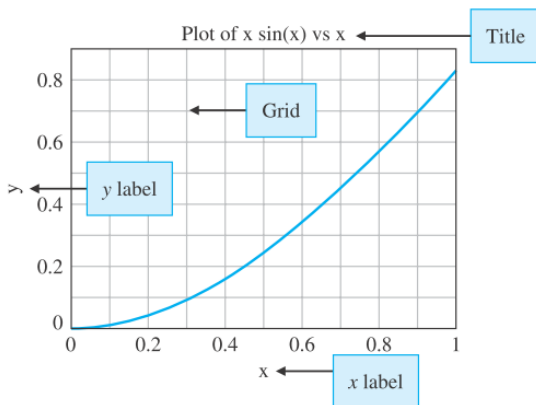
	<code>xlabel('string')</code>	
<b>ylabel</b>	Add ylabel to current plot <code>ylabel('string')</code>	<code>&gt;&gt; ylabel('temperature')</code>
<b>legend</b>	Creates a legend in the corner (or at a specified position) of the plot	<code>&gt;&gt; legend('temperature')</code>
<b>hold</b>	Freezes the current plot, so that additional plots can be overlaid	<code>&gt;&gt;hold on</code> <code>&gt;&gt;hold off</code>

## Example:

Here we see some examples of how to use the different plot functions:

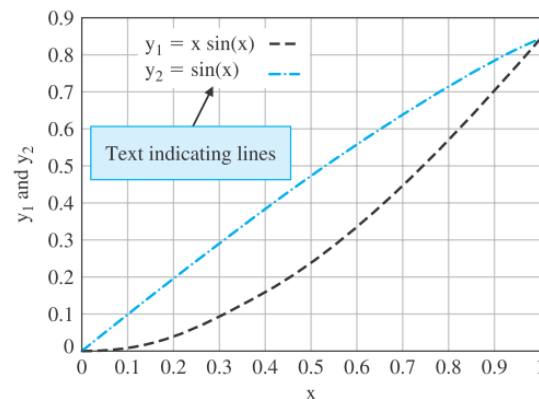
```
>>x=[0:0.1:1];
>>y=x.*sin(x);
>>plot(x,y)
>>title('Plot of x sin(x) vs x ')
>>xlabel('x')
>>ylabel('y')
>>grid on
```

(a)



```
>> x=[0:0.1:1];
>> y1=x.*sin(x); y2=sin(x);
>> plot(x,y1,'-.',x,y2,'-.')
>> text(0.1,0.85,'y_1 = x sin(x) ---')
>> text(0.1,0.80,'y_2 = sin(x) .\_.\_.')
>> xlabel('x'), ylabel('y_1 and y_2'), grid on
```

(a)

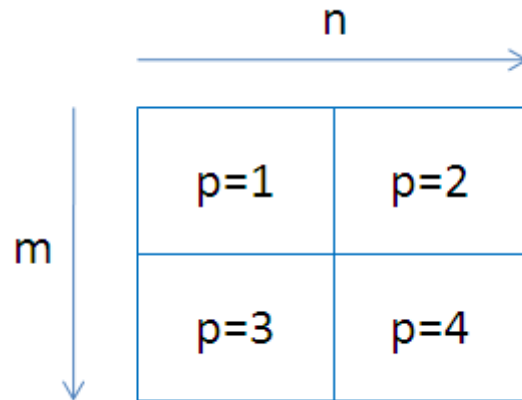


[Figure: R. C. Dorf and R. H. Bishop, Modern Control Systems, Eleventh Edition: Pearson Prentice Hall]

[End of Example]

## 2.6.1 Subplots

The subplot command enables you to display multiple plots in the same window or print them on the same piece of paper. Typing “subplot(m,n,p)” partitions the figure window into an m-by-n matrix of small subplots and selects the pth subplot for the current plot. The plots are numbered along the first row of the figure window, then the second row, and so on.



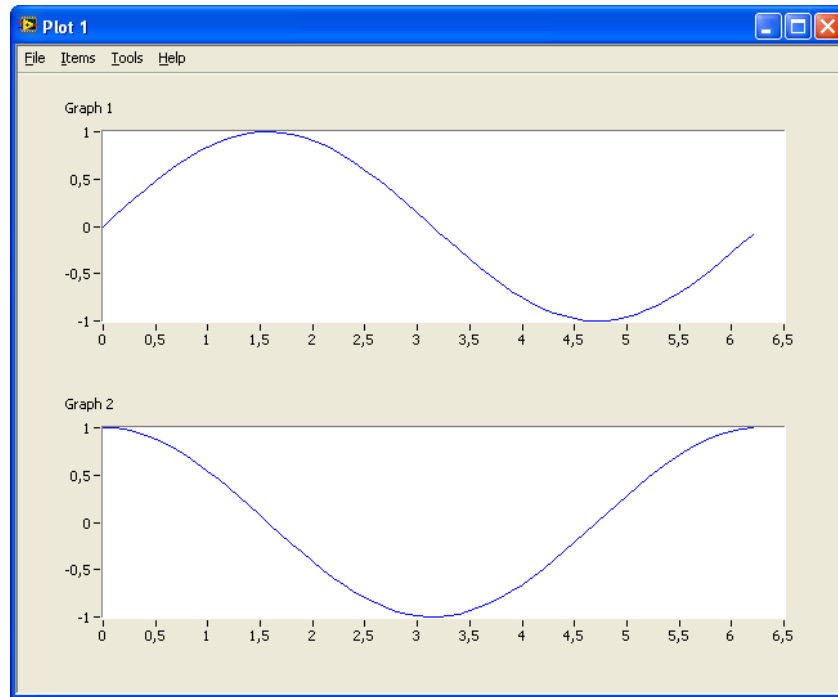
The syntax is as follows:

```
subplot(m,n,p)
```

### Example:

```
x=0:0.1:2*pi;  
  
subplot(2,1,1)  
y=sin(x);  
plot(x,y)  
  
subplot(2,1,2)  
z=cos(x);  
plot(x,z)
```

This gives the following plot:



[End of Example]

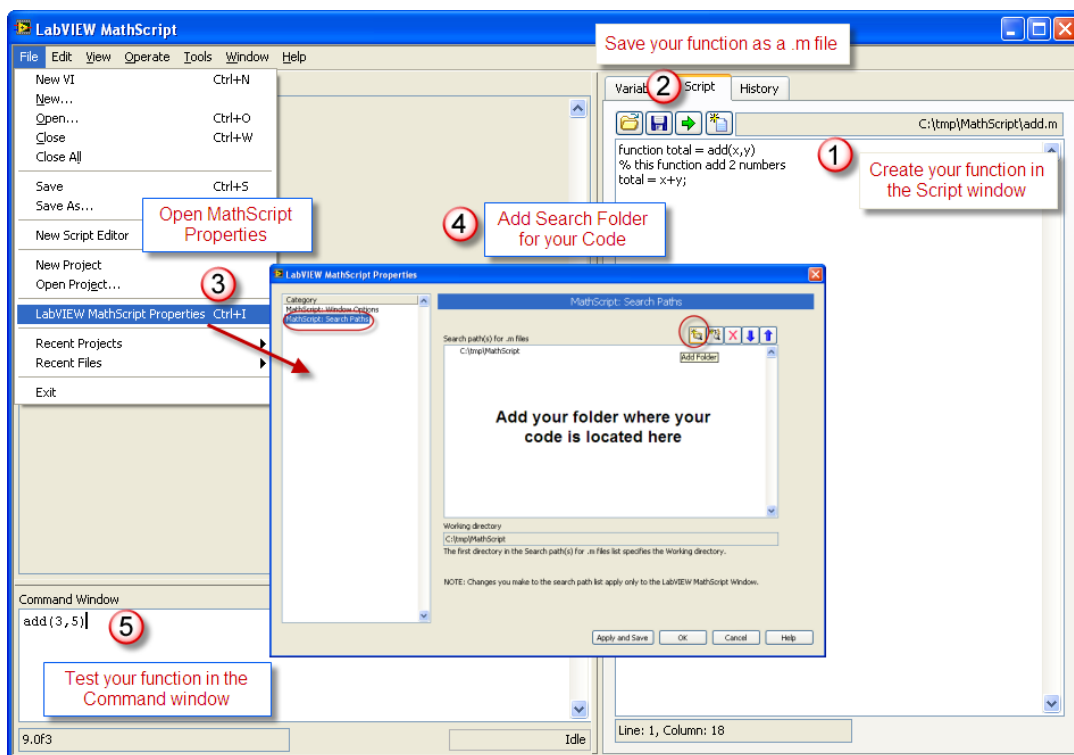
## 2.7 User-Defined Functions in MathScript

MathScript includes more than 800 built-in functions that you can use but sometimes you need to create your own functions.

To define your own function in MathScript, use the following syntax:

```
function outputs = function_name(inputs)
% documentation
...
```

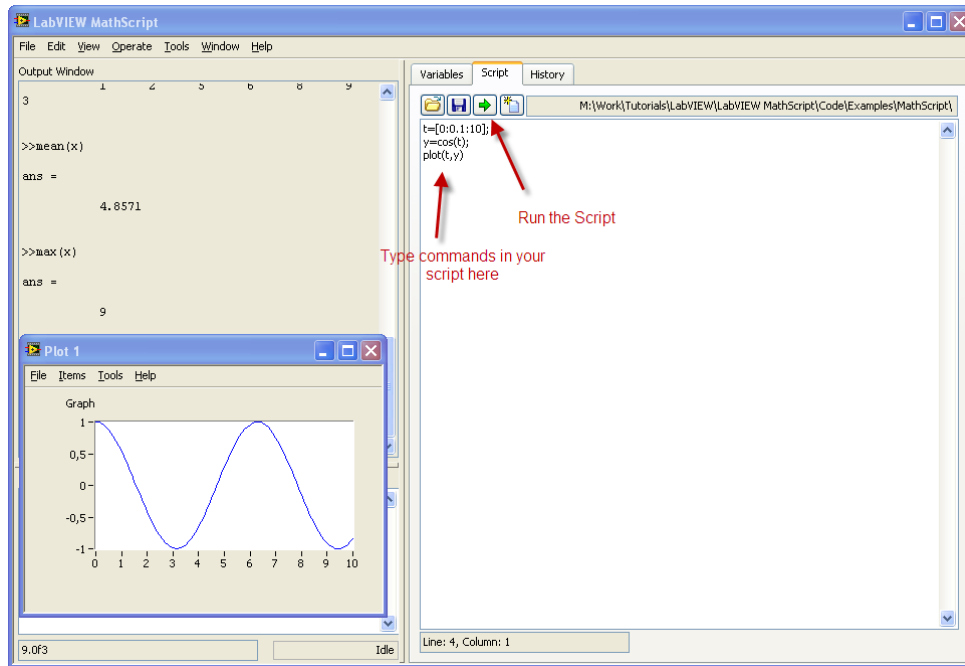
The figure below illustrates how to create and use functions in MathScript:



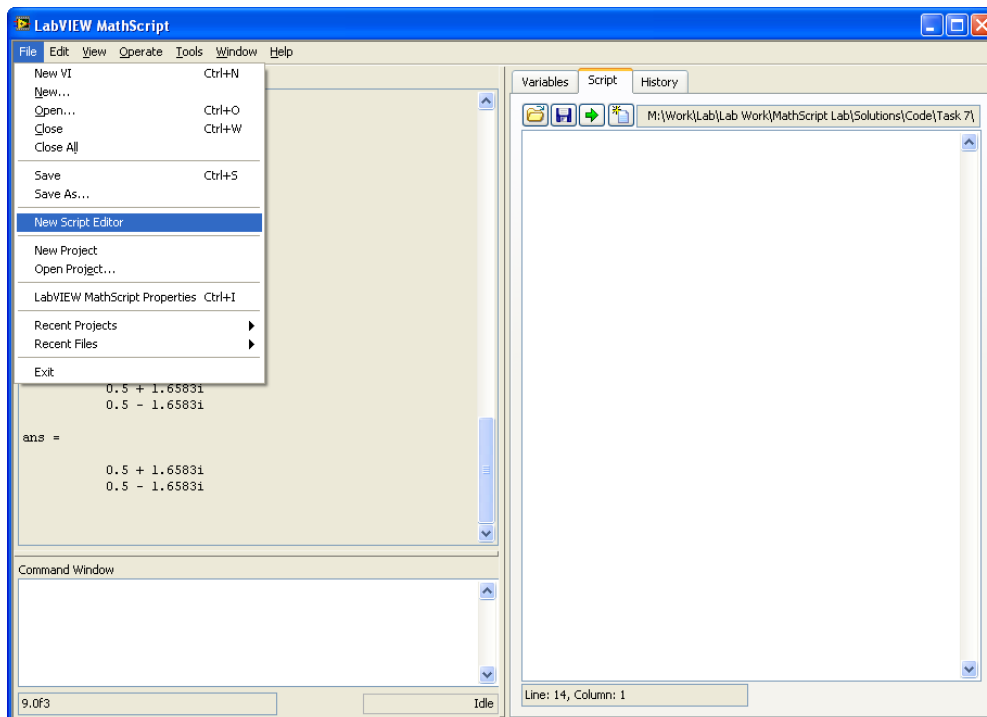
## 2.8 Scripts

A script is a sequence of MathScript commands that you want to perform to accomplish a task. When you have created the script you may save it as a m-file for later use.

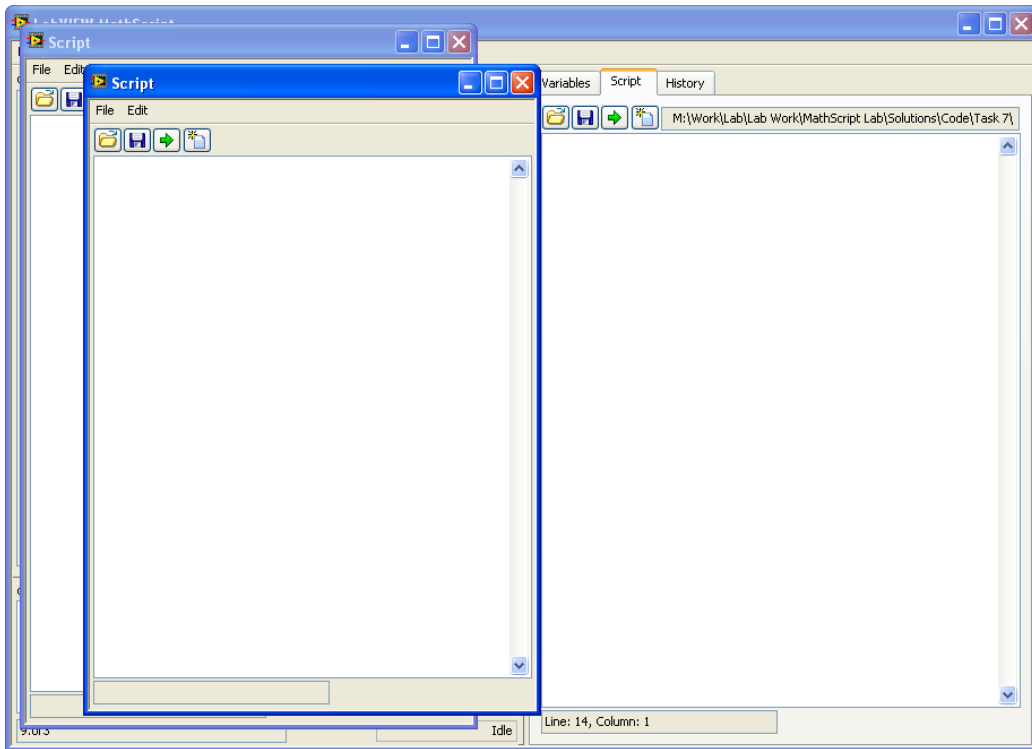




You may also have multiple Script Windows open at the same time by selecting **"New Script Editor"** in the File menu:



This gives:



## 2.9 Flow Control

You may use different loops in MathScript

- For loop
- While loop

If you want to control the flow in your program, you may want to use one of the following:

- If-else statement
- Switch and case statement

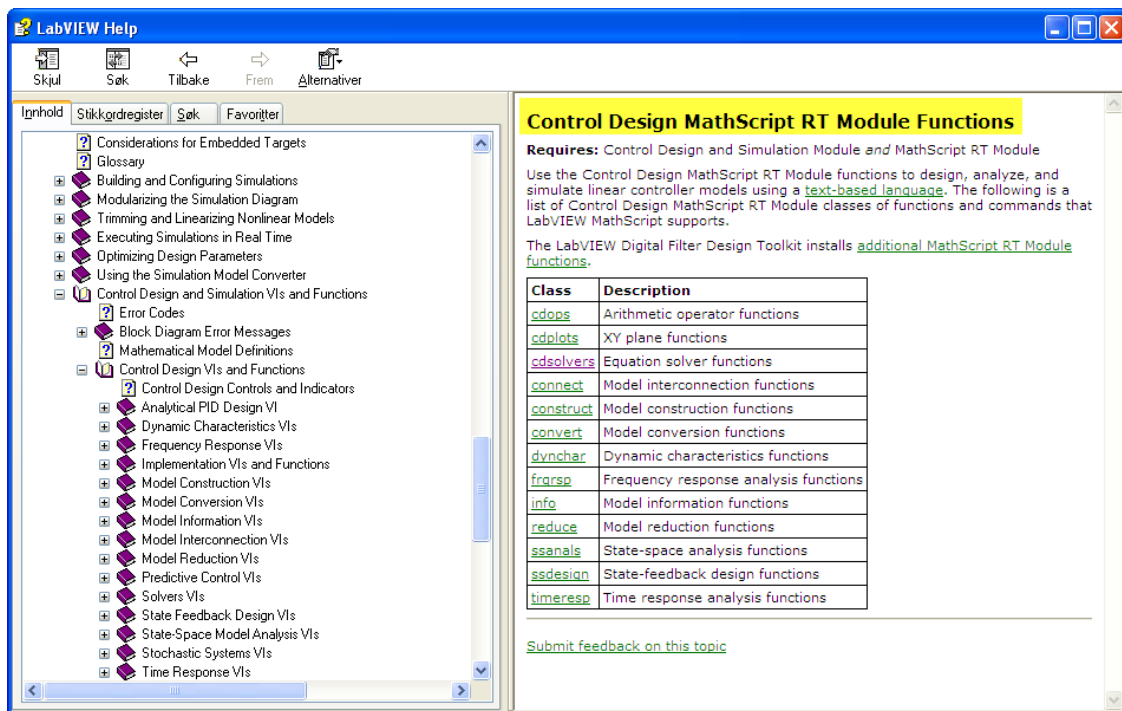
### Example:

```
function av = calc_average2(x)
%This function calculates the average of a vector x
N=length(x);
tot=0;
for i=1:N
    tot=tot+x(i);
end
av=tot;
```

[End of Example]

## 2.10 Control Design in MathScript

Type “**help cdt**” in the Command Window in the MathScript environment. The LabVIEW Help window appears:



Use the Help window and read about some of the functions available for control design and simulation.

See Appendix A for a list of some of the most used functions with description and examples.

# 3 Transfer Functions

## 3.1 Introduction

Transfer functions are a model form based on the Laplace transform. Transfer functions are very useful in analysis and design of linear dynamic systems.

A general Transfer function is on the form:

$$H(S) = \frac{y(s)}{u(s)}$$

Where y is the output and u is the input.

A general transfer function can be written on the following general form:

$$H(s) = \frac{\text{numerator}(s)}{\text{denominator}(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0}$$

The Numerators of transfer function models describe the locations of the zeros of the system, while the Denominators of transfer function models describe the locations of the poles of the system.

Below we will learn more about 2 important special cases of this general form, namely the 1.order transfer function and the 2.order transfer function.

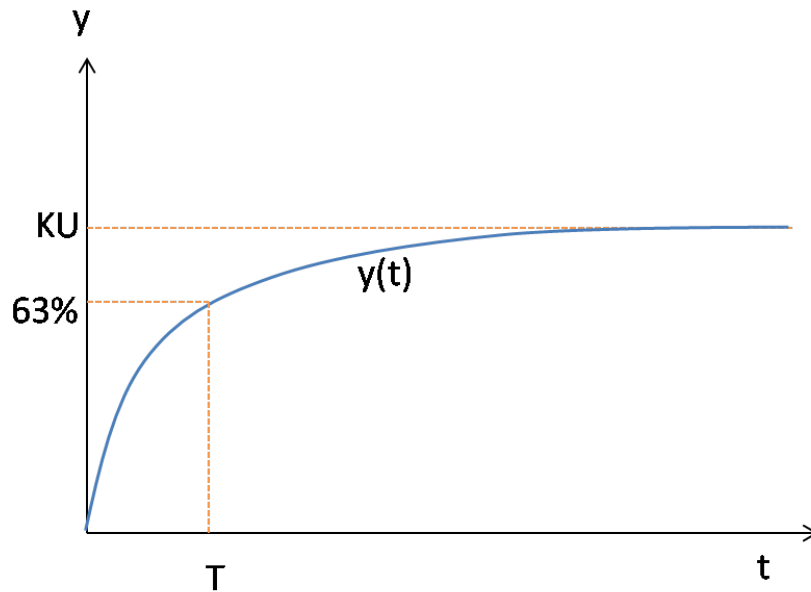
### 3.1.1 1.order system

A 1.order transfer function:

$$H(s) = \frac{K}{Ts + 1}$$

Where  $K$  is the Gain and  $T$  is the Time constant.

A step response of such a transfer function has the following characteristics:

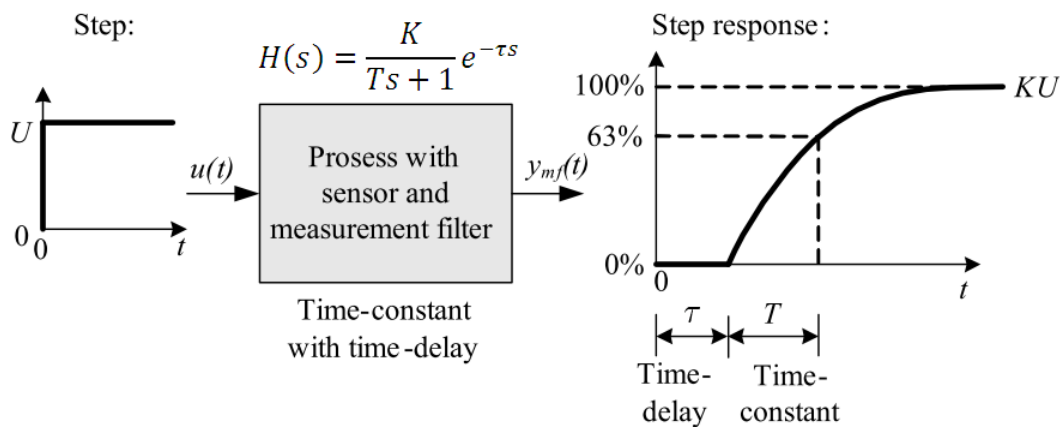


### 3.1.2 1.order system with time-delay

A 1.order transfer function with time-delay may be written as:

$$H(s) = \frac{K}{Ts + 1} e^{-\tau s}$$

A step response of such a transfer function has the following characteristics:



[Figure: F. Haugen, Advanced Dynamics and Control: TechTeach, 2010]

From the step response of such a system we can easily find  $K$ ,  $T$  and  $\tau$ .

More about time-delays in a later chapter.

### 3.1.3 2.order system

A 2.order transfer function:

$$H(s) = \frac{K}{\left(\frac{s}{\omega_0}\right)^2 + 2\zeta \frac{s}{\omega_0} + 1}$$

More about 1.order and 2.order transfer functions later in this chapter.

## 3.2 MathScript

MathScript has several functions for creating transfer functions:

Function	Description	Example
<b>tf</b>	Creates system model in transfer function form. You also can use this function to state-space models to transfer function form.	<pre>&gt;num=[1]; &gt;den=[1, 1, 1]; &gt;H = tf(num, den)</pre>
<b>Sys_order1</b>	Constructs the components of a first-order system model based on a gain, time constant, and delay that you specify. You can use this function to create either a state-space model or a transfer function model, depending on the output parameters you specify.	<pre>&gt;K = 1; &gt;tau = 1; &gt;H = sys_order1(K, tau)</pre>
<b>Sys_order2</b>	Constructs the components of a second-order system model based on a damping ratio and natural frequency you specify. You can use this function to create either a state-space model or a transfer function model, depending on the output parameters you specify.	<pre>&gt;dr = 0.5 &gt;wn = 20 &gt;[num, den] = sys_order2(wn, dr) &gt;SysTF = tf(num, den)</pre>
<b>pid</b>	Constructs a proportional-integral-derivative (PID) controller model in parallel, series, or academic form. Refer to the LabVIEW Control Design User Manual for information about these three forms.	<pre>&gt;Kc = 0.5; &gt;Ti = 0.25; &gt;SysOutTF = pid(Kc, Ti, 'academic');</pre>

Given the general transfer function:

$$H(s) = \frac{\text{numerator}(s)}{\text{denominator}(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \dots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \dots + a_1 s + a_0}$$

In MathScript we can define such a transfer function using the built-in **tf** function as follows:

```
num=[bm, bm_1, bm_2, ... , b1, b0];
den=[an, an_1, an_2, ... , a1, a0];
H = tf(num, den)
```

### Example:

1. Given the following transfer function:

$$H(s) = \frac{2s^2 + 3s + 4}{5s + 9}$$

MathScript Code:

```
num=[2, 3, 4];
den=[5, 9];
H = tf(num, den)
```

2. Given the following transfer function:

$$H(s) = \frac{4s^4 + 3s + 4}{5s^2 + 9}$$

MathScript Code:

```
num=[4, 0, 0, 3, 4];
den=[5, 0, 9];
H = tf(num, den)
```

Note! If some of the orders are missing, we just put in zeros. The transfer function above can be rewritten as:

$$H(s) = \frac{4s^4 + 0 \cdot s^3 + 0 \cdot s^2 + 3s + 4}{5s^2 + 0 \cdot s + 9}$$

3. Given the following transfer function:

$$H(s) = \frac{7 + 3s + 2s^2}{5s + 6s^2}$$

We need to rewrite the transfer function to get it in correct orders:

$$H(s) = \frac{2s^2 + 3s + 7}{6s^2 + 5s}$$

MathScript Code:

```
num=[2, 3, 7];
den=[6, 5, 0];
H = tf(num, den)
```

[End of Example]

For creating more complex transfer functions, some of the following functions are useful:

Function	Description	Example
<b>conv</b>	Computes the convolution of two vectors or matrices. Example: $H(s) = \frac{K}{(2s + 1)(3s + 1)}$	>den1 = [2, 1]; >den2 = [3, 1]; >C = conv(den1, den2)
<b>series</b>	Connects two system models in series to produce a model SysSer with input and output connections you specify. Example: $H(s) = H_1(s)H_2(s)$	>Hseries = series(H1,H2)
<b>feedback</b>	Connects two system models together to produce a closed-loop model using negative or positive feedback connections	>SysClosed = feedback(SysIn_1, SysIn_2)



## 3.3 First order Transfer Function

A first order transfer function is given on the form:

$$H(s) = \frac{K}{Ts + 1}$$

Where

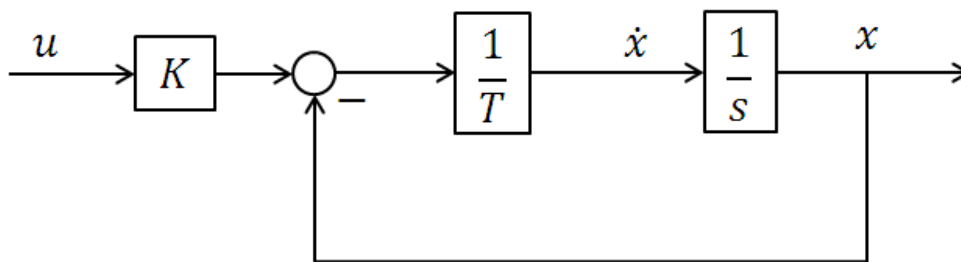
$K$  is the Gain

$T$  is the Time constant

In the time domain we get the following differential equation (using Inverse Laplace):

$$\dot{x} = \frac{1}{T}(-x + Ku)$$

We can draw the following block diagram of the system:



### Example:

We will use the `tf` function in MathScript to define the transfer function:

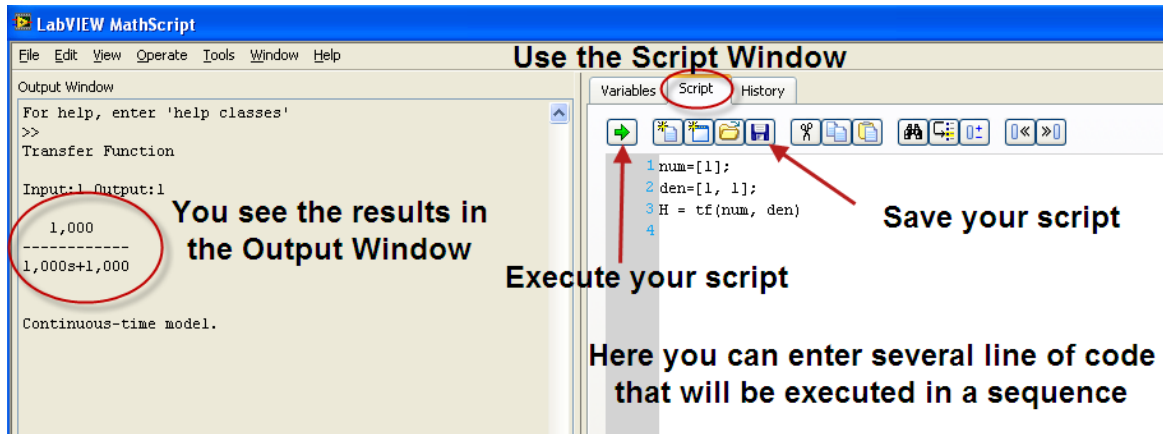
$$H(s) = \frac{K}{Ts + 1}$$

We set  $K = 1$  and  $T = 1$ .

MathScript Code using the `tf` function:

```
K=1;
T=1;
num=[K];
den=[T, 1];
H = tf(num, den)
```

We enter the code shown above in the Script window as shown below:



MathScript Code using the `sys_order1` function:

```

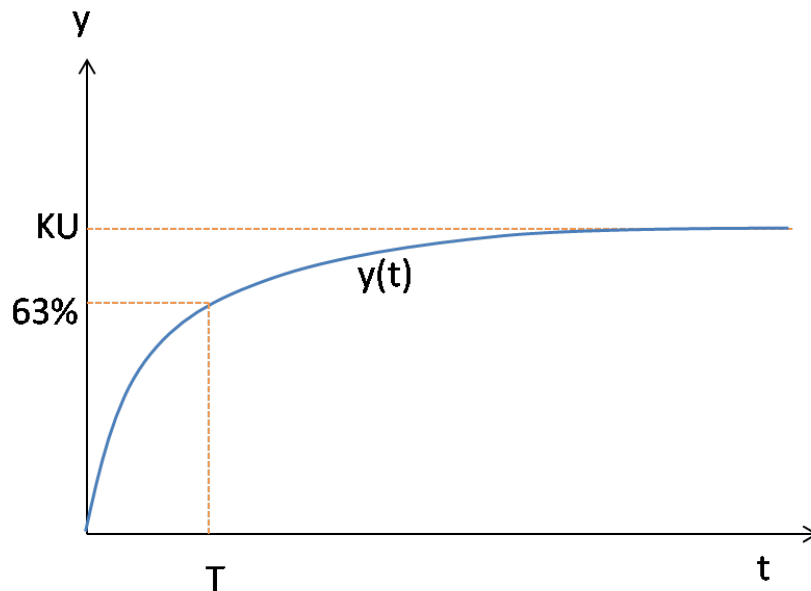
K = 1;
T = 1;
H = sys_order1(K, T)

```

[End of Example]

### Step Response:

The step response for a 1.order transfer function has the following characteristics (a step  $U$  at  $t = 0$ ):



The time constant  $T$  is defined as the time where the response reaches 63% of the steady state value.

### Example:

Given the following 1.order transfer function:

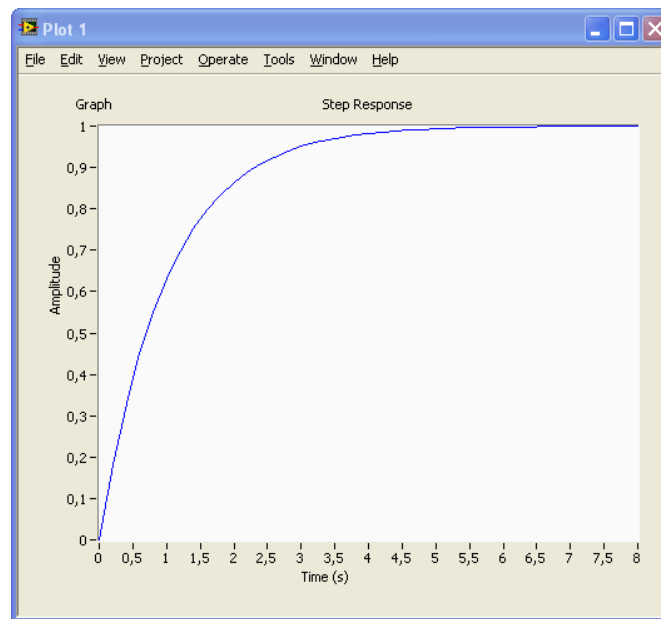
$$H(s) = \frac{1}{s + 1}$$

$(K = 1, T = 1)$

We create the following code in order to plot the step response for this system:

```
K=1;
T=1;
num=[K];
den=[T, 1];
H = tf(num, den);
Step(H)
```

This gives the following step response:



[End of Example]

### 3.3.1 1.order system with time-delay

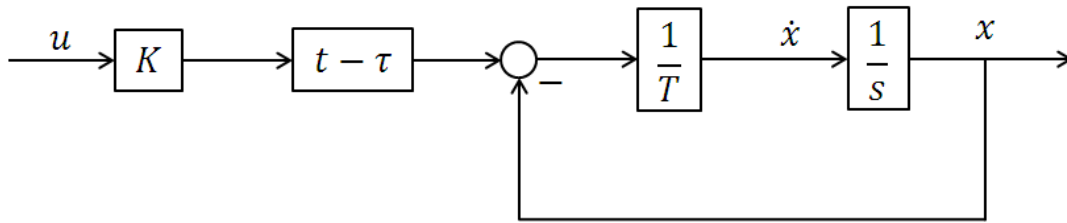
A 1.order system with time-delay has the following transfer function:

$$H(s) = \frac{K}{Ts + 1} e^{-\tau s}$$

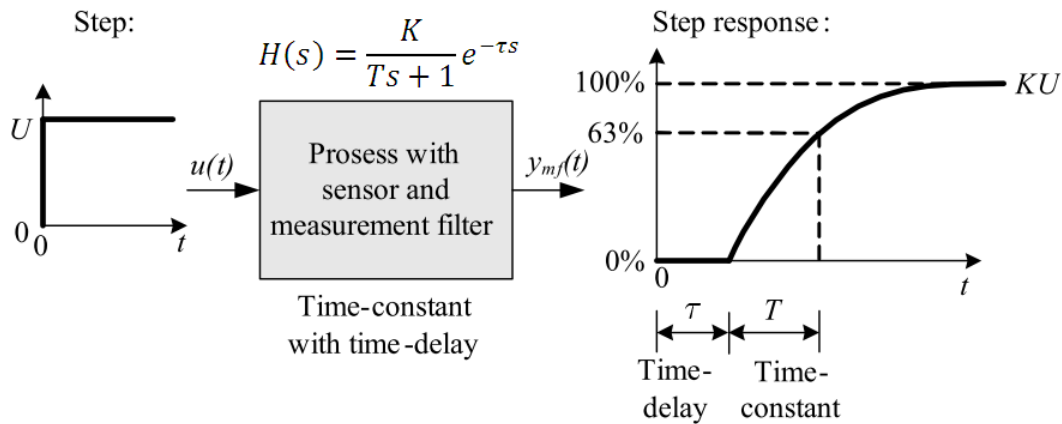
In the time domain we get the following differential equation (using Inverse Laplace):

$$\dot{x} = \frac{1}{T} (-x + Ku(t - \tau))$$

We can draw the following block diagram of the system:



### Step Response:



[Figure: F. Haugen, Advanced Dynamics and Control: TechTeach, 2010]

### Example:

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

( $K = 1, T = 2, \tau = 3$ )

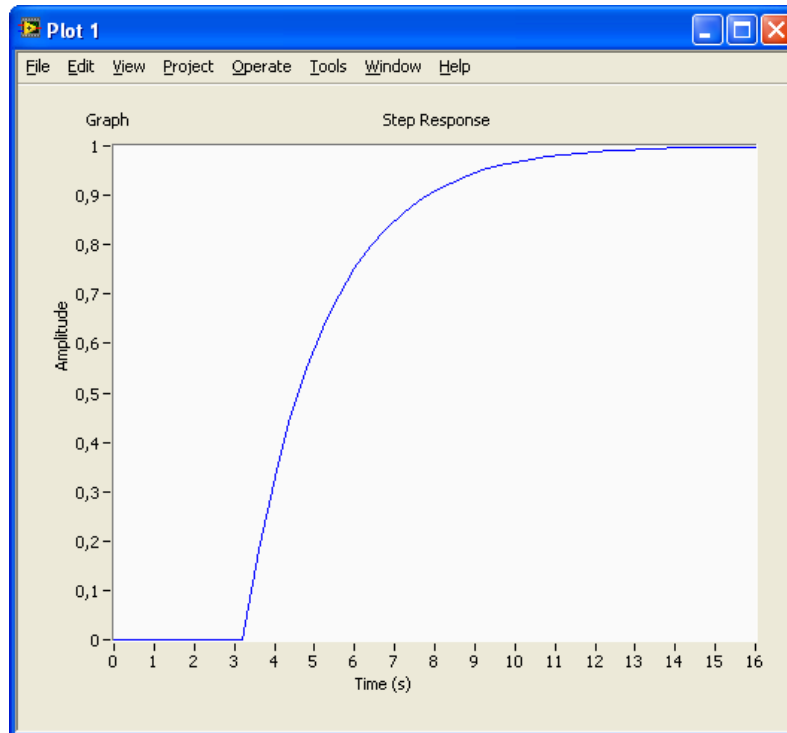
The MathScript code becomes (using the built-in `sys_order1` function):

```
K = 1;
T = 2;
delay=3;

H = sys_order1(K, T, delay)

step(H)
```

The plot of the step response becomes:



[End of Example]

## 3.4 Second order Transfer Function

A second order transfer function is given on the form:

$$H(s) = \frac{K}{\left(\frac{s}{\omega_0}\right)^2 + 2\zeta \frac{s}{\omega_0} + 1}$$

Where

$K$  is the gain

$\zeta$  zeta is the relative damping factor

$\omega_0$ [rad/s] is the undamped resonance frequency.

### Example:

Define the transfer function in MathScript. Set  $K = 1, \zeta = 1, \omega_0 = 1$

Use the **tf** function or the **sys\_order2** function in MathScript

MathScript Code:

```
num=[1];
den=[1, 2, 1];
H = tf(num, den)
```

or:

```
dr = 1
wn = 1
[num, den] = sys_order2(wn, dr)
H = tf(num, den)
```

[End of Example]

**2.order system - special case: When  $\zeta > 0$  and the poles are real and distinct we have:**

$$H(s) = \frac{K}{(T_1s + 1)(T_2s + 1)}$$

We see that this system can be considered as two 1.order systems in series.

$$H(s) = H_1(s)H_1(s) = \frac{K}{(T_1s + 1)} \cdot \frac{1}{(T_2s + 1)} = \frac{K}{(T_1s + 1)(T_2s + 1)}$$

## 3.5 Simulation

MathScript has several functions used for simulation purposes:

Function	Description	Example
<b>plot</b>	Generates a plot. plot(y) plots the columns of y against the indexes of the columns.	<pre>&gt;X = [0:0.01:1]; &gt;Y = X.*X; &gt;plot(X, Y)</pre>
<b>step</b>	Creates a step response plot of the system model. You also can use this function to return the step response of the model outputs. If the model is in state-space form, you also can use this function to return the step response of the model states. This function assumes the initial model states are zero. If you do not specify an output, this function creates a plot.	<pre>&gt;num=[1,1]; &gt;den=[1,-1,3]; &gt;H=tf(num,den); &gt;t=[0:0.01:10]; &gt;step(H,t);</pre>
<b>lsim</b>	Creates the linear simulation plot of a system model. This function calculates the output of a system model when a set of inputs excite the model, using discrete simulation. If you do not specify an output, this function creates a plot.	<pre>&gt;t = [0:0.1:10] &gt;u = sin(0.1*pi*t) &gt;lsim(SysIn, u, t)</pre>

**Plots functions:** Here are some useful functions for creating plots: **plot**, **figure**, **subplot**, **grid**, **axis**, **title**, **xlabel**, **ylabel**, **semilogx** – for more information about the plots function, type “**help plots**”. Or type “**help <functionname>**”.

## 3.6 Block Diagrams

MathScript have built-in functions for manipulating block diagrams and transfer functions.

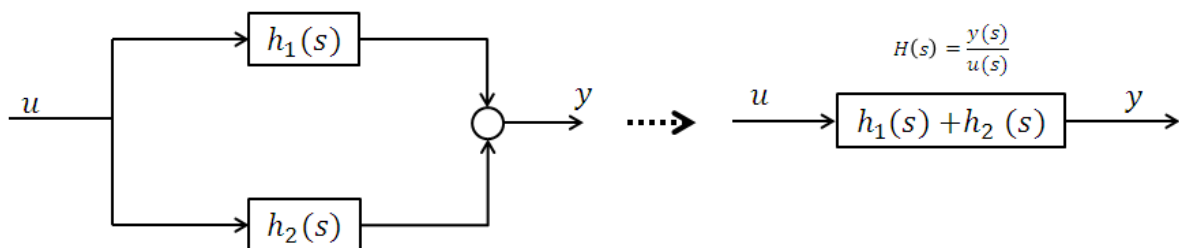
**Serial:**



MathScript:

```
...
H = series(h1, h2)
```

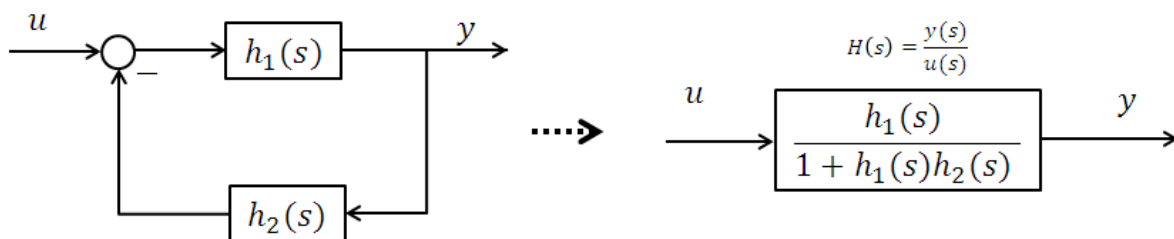
Parallel:



MathScript:

```
...
H = parallel(h1, h2)
```

Feedback:



MathScript:

```
...
H = feedback(h1, h2)
```

## 3.7 Analysis of Standard Functions

Here we will take a closer look at the following standard functions:

- Integrator
- 1. Order system
- 2. Order system

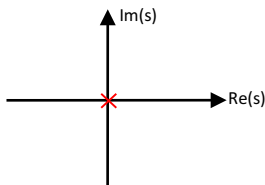
### 3.7.1 Integrator

The transfer function for an Integrator is as follows:

$$H(s) = \frac{K}{s}$$

#### Pole(s):

The Integrator has a pole in origo:  $p = 0$



In MathScript you may use the **poles** function in order to find the poles.

#### Example:

```
K=1;
T=1;
num=[K];
den=[T 1];
H=tf(num,den);
p=poles(H)
```

[End of Example]

#### Step response:

Note! In MathScript we can use the **step** function for this purpose.

Here we will find the mathematical expression for the step response ( $\mathbf{y}(t)$ ):

The Laplace Transformation pair for a step is as follows:

$$\frac{1}{s} \Leftrightarrow 1$$

The step response of an integrator then becomes:



$$y(s) = H(s)u(s) = \frac{K}{s} \cdot \frac{U}{s} = KU \frac{1}{s^2}$$

We use the following Laplace Transformation pair in order to find  $y(t)$ :

$$\boxed{\frac{1}{s^2} \Leftrightarrow t}$$

Then we get:

$$\underline{y(t) = KUt}$$

→ We see that the step response of the integrator is a Ramp.

Conclusion: A bigger K will give a bigger slope (In Norwegian: “stigningstall”) and the integration will go faster. The simulation in MathScript below will also show this.

Below we will show this by using the **step** function in MathScript:

### Example:

In MathScript we use the **step** function for simulation of a step response. We set K=0.2, 1, 5.

MathScript Code:

```
t=[0:0.5:5];
K=0.2
num=[K];
den=[1 0];
H1=tf(num,den);

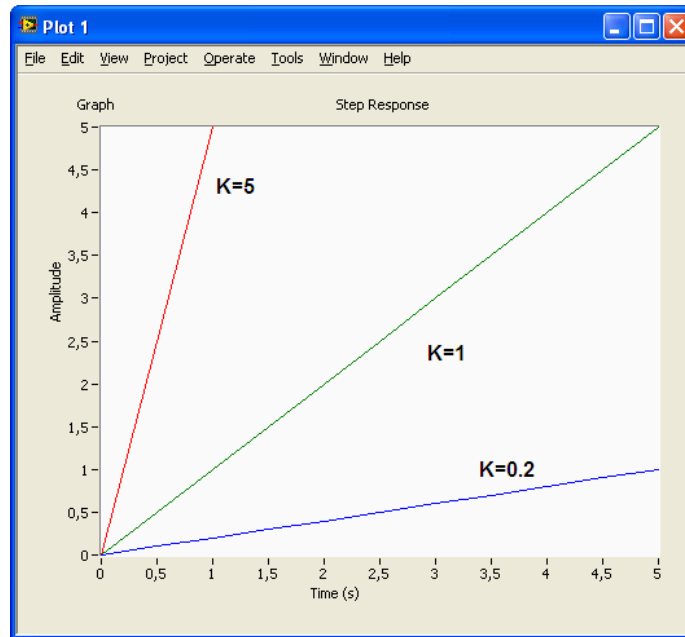
K=1
num=[K];
den=[1 0];
H2=tf(num,den);

K=5
num=[K];
den=[1 0];
H3=tf(num,den);

step(H1,H2,H3,t)
axis([0, 5, 0, 5])
```

Note! Using a For Loop in this case would be a better approach.

Plot:



[End of Example]

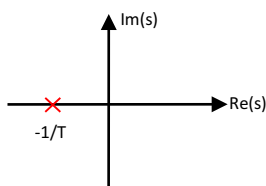
### 3.7.2 1. order system

The transfer function for a 1.order system is as follows:

$$H(s) = \frac{K}{Ts + 1}$$

**Pole(s):**

A 1.order system has a pole:  $p = -\frac{1}{T}$



In MathScript you may use the **poles** function in order to find the poles. The function **pzgraph** plots the poles and zeros

**Step response:**

Note! In MathScript we can use the **step** function for this purpose.

Here we will find the mathematical expression for the step response ( $y(t)$ ):

$$y(s) = H(s)u(s)$$

Where

$$u(s) = \frac{U}{s}$$

We use inverse Laplace and find the corresponding transformation pair in order to find  $y(t)$ .

$$y(s) = \frac{K}{Ts + 1} \cdot \frac{U}{s}$$

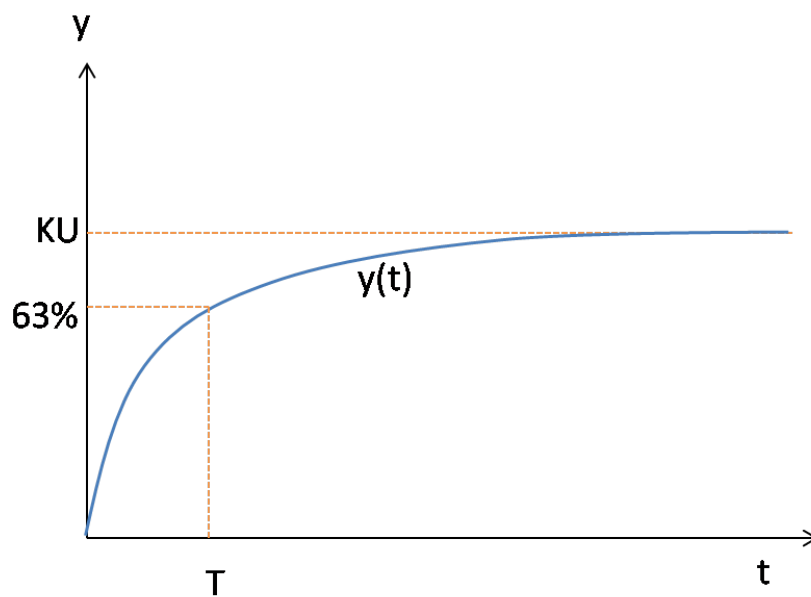
We use the following Laplace transform pair:

$$\frac{k}{(Ts + 1)s} \Leftrightarrow k(1 - e^{-t/T})$$

This gives:

$$\underline{y(t) = KU(1 - e^{-t/T})}$$

The step response is as follows:



Below we will show this by using the **step** function in MathScript:

### Example:

For different values for K, eg., K=0.5, 1, 2 and T=1. We use the **step** function in MathScript.

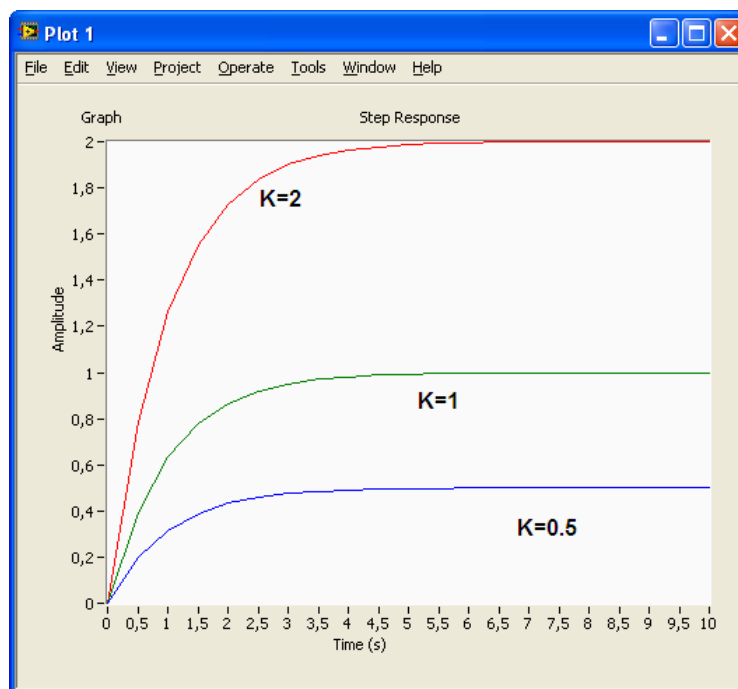
```
t=[0:0.5:10];
den=[1 1];

K=0.5;
num=[K];
H1=tf(num,den);
```

```
K=1;  
num=[K];  
H2=tf(num,den);  
  
K=2;  
num=[K];  
H3=tf(num,den);  
step(H1,H2,H3,t);  
axis([0,10,0,2]);
```

Note! Using a For Loop in this case would be a better approach.

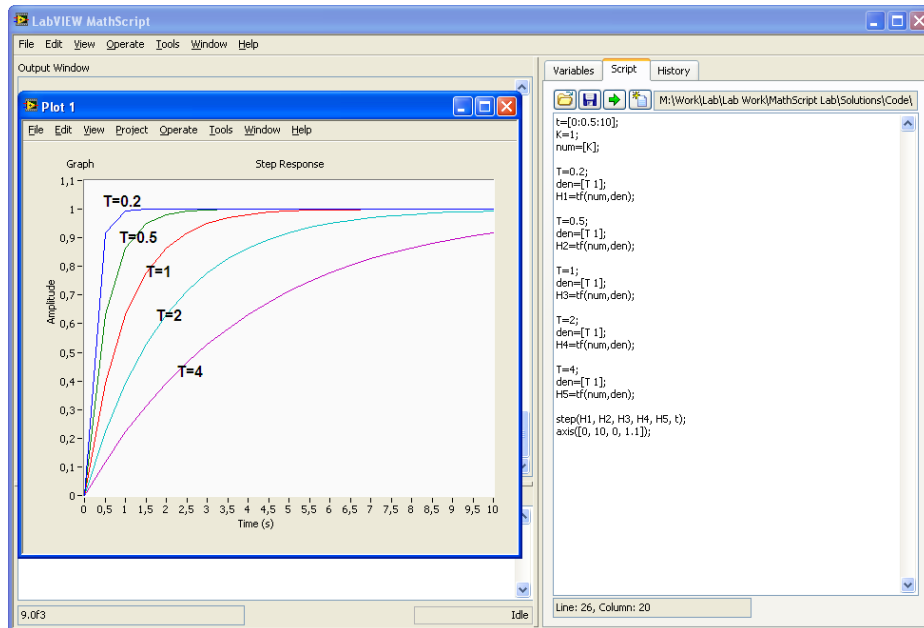
Below we see the plot for this:



[End of Example]

### Example:

For different values for T: T=0.2, 0.5, 1, 2, 4 and K=1



Note! Using a For Loop in this case would be a better approach.

We see from Figure above that smaller T (Time constant) gives faster response.

[End of Example]

### 3.7.3 2. order system

The transfer function for a 2. order system is as follows:

$$H(s) = \frac{K \omega_0^2}{s^2 + 2\zeta \omega_0 s + \omega_0^2} = \frac{K}{\left(\frac{s}{\omega_0}\right)^2 + 2\zeta \frac{s}{\omega_0} + 1}$$

Where

- $K$  is the gain
- $\zeta$  zeta is the relative damping factor
- $\omega_0$ [rad/s] is the undamped resonance frequency.

We have that:

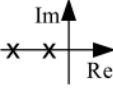
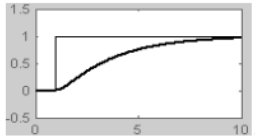
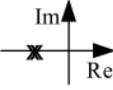
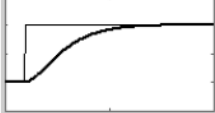
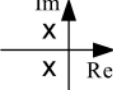
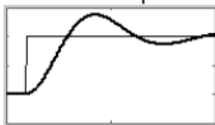
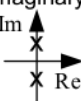
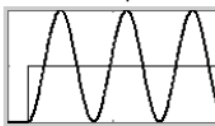
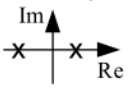
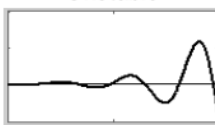
Value of $\zeta$	Poles $p_1$ and $p_2$	Type of step response $y(t)$
$\zeta > 1$	Real and distinct 	Overdamped 
$\zeta = 1$	Real and multiple 	Critically damped 
$0 < \zeta < 1$	Complex conj. 	Underdamped 
$\zeta = 0$	Imaginary 	Undamped 
$\zeta < 0$	Pos. real part 	Unstable 

Figure: F. Haugen, Advanced Dynamics and Control: TechTeach, 2010.

→ Show this in MathScript

**Special case: When  $\zeta > 0$  and the poles are real and distinct we have:**

$$H(s) = \frac{K}{(T_1s + 1)(T_2s + 1)}$$

We see that this system can be considered as two 1.order systems in series:

$$H(s) = H_1(s)H_2(s) = \frac{K}{(T_1s + 1)} \cdot \frac{1}{(T_2s + 1)} = \frac{K}{(T_1s + 1)(T_2s + 1)}$$

# 4 State-space Models

## 4.1 Introduction

A state-space model is a structured form or representation of a set of differential equations. State-space models are very useful in Control theory and design. The differential equations are converted in matrices and vectors, which is the basic elements in MathScript.

We have the following equations:

$$\begin{aligned} \dot{x}_1 &= a_{11}x_1 + a_{21}x_2 + \dots + a_{n1}x_n + b_{11}u_1 + b_{21}u_2 + \dots + b_{n1}u_n \\ &\vdots \\ \dot{x}_n &= a_{1n}x_1 + a_{2n}x_2 + \dots + a_{nn}x_n + b_{1n}u_1 + b_{2n}u_2 + \dots + b_{nn}u_n \\ &\vdots \end{aligned}$$

This gives on vector form:

$$\begin{aligned} \underbrace{\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix}}_{\dot{x}} &= \underbrace{\begin{bmatrix} a_{11} & \dots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \dots & a_{nm} \end{bmatrix}}_A \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_x + \underbrace{\begin{bmatrix} b_{11} & \dots & b_{n1} \\ \vdots & \ddots & \vdots \\ b_{1m} & \dots & b_{nm} \end{bmatrix}}_B \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}}_u \\ \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_y &= \underbrace{\begin{bmatrix} c_{11} & \dots & c_{n1} \\ \vdots & \ddots & \vdots \\ c_{1m} & \dots & c_{nm} \end{bmatrix}}_C \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_x + \underbrace{\begin{bmatrix} d_{11} & \dots & d_{n1} \\ \vdots & \ddots & \vdots \\ d_{1m} & \dots & d_{nm} \end{bmatrix}}_D \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}}_u \end{aligned}$$

This gives the following compact form of a general linear State-space model:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

### Example:

We have the following system:

$$\dot{x}_1 = x_2$$

$$2\dot{x}_2 = -2x_1 - 6x_2 + 4u_1 + 8u_2$$

$$y = 5x_1 + 6x_2 + 7u_1$$

Convert to the following state-space form:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

First we do:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -x_1 - 3x_2 + 2u_1 + 4u_2$$

$$y = 5x_1 + 6x_2 + 7u_1$$

This gives:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix}}_A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 0 \\ 2 & 4 \end{bmatrix}}_B \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$y = \underbrace{\begin{bmatrix} 5 & 6 \end{bmatrix}}_C \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 7 & 0 \end{bmatrix}}_D \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

→ Try to define this State-Space model in MathScript.

[End of Example]

## 4.2 MathScript

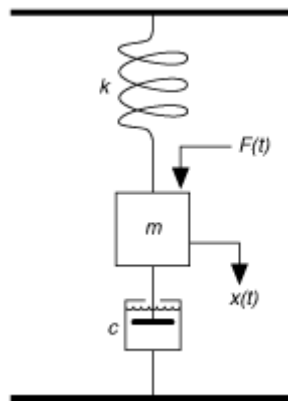
MathScript has several functions for creating state-space models:

Function	Description	Example
<b>ss</b>	Constructs a model in state-space form. You also can use this function to convert transfer function models to state-space form.	<pre>&gt;A = [1 2; 3 4] &gt;B = [0; 1] &gt;C = B' &gt;SysOutSS = <b>ss</b>(A, B, C)</pre>
<b>Sys_order1</b>	Constructs the components of a first-order system model based on a gain, time constant, and delay that you specify. You can use this function to create either a state-space model or a transfer function model, depending on the output parameters you specify.	<pre>&gt;K = 1; &gt;tau = 1; &gt; [A, B, C, D] = <b>sys_order1</b>(K, tau)</pre>
<b>Sys_order2</b>	Constructs the components of a second-order system model based on a damping ratio and natural frequency you specify. You can use this function to create either a state-space model or a transfer function model, depending on the output parameters you specify.	<pre>&gt;dr = 0.5 &gt;wn = 20 &gt; [A, B, C, D] = <b>sys_order2</b>(wn, dr) &gt;SysSS = <b>ss</b>(A, B, C, D)</pre>

### Example:



Given a **mass-spring-damper** system:



Where  $c$ =damping constant,  $m$ =mass,  $k$ =spring constant,  $F=u$ =force

The state-space model for the system is:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u$$

$$y = [1 \quad 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

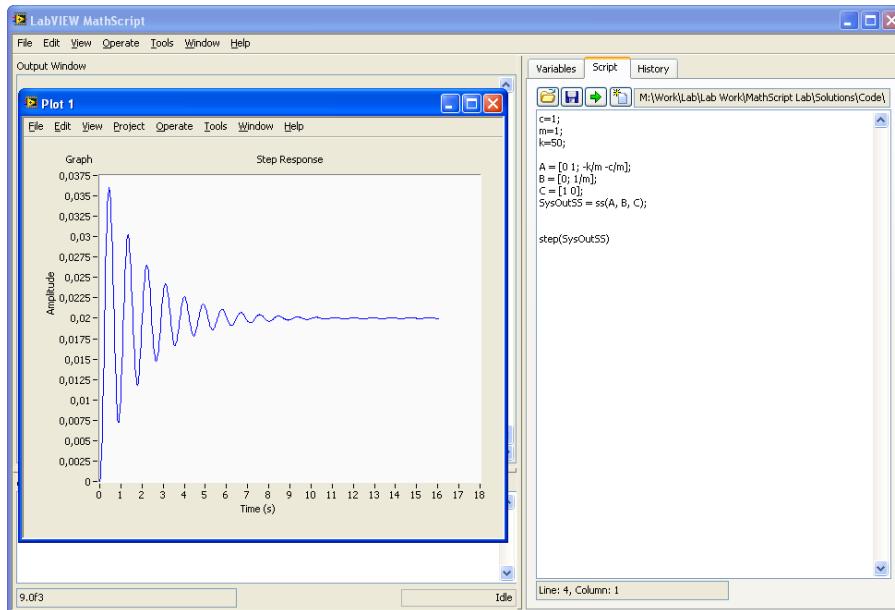
Define the state-space model above using the **ss** function in MathScript. Set some arbitrary values for  $c$ =damping constant,  $m$ =mass,  $k$ =spring constant.

We will use MathScript to define the state space model:

MathScript Code:

```
c=1;
m=1;
k=1;
A = [0 1; -k/m -c/m];
B = [0; 1/m];
C = [1 0];
SysOutSS = ss(A, B, C)
```

We use the **step** function in MathScript in order to simulate the step response:



Try with different values of  $c$ ,  $m$  and  $k$  and watch the results.

[End of Example]

# 5 Time-delay and Padé'-approximations

## 5.1 Introduction

Time-delays are very common in control systems. The Transfer function of a time-delay is:

$$H(s) = e^{-\tau s}$$

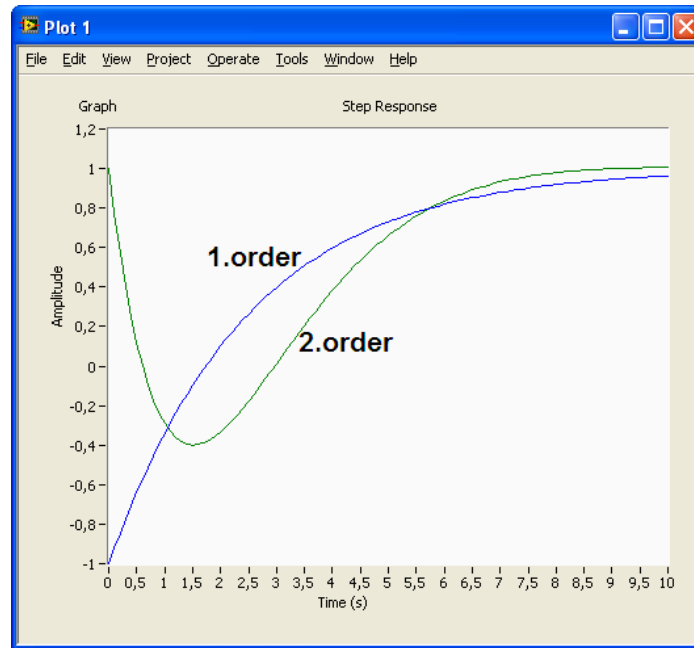
A 1.order transfer function with time-delay may be written as:

$$H(s) = \frac{K}{Ts + 1} e^{-\tau s}$$

In some situations it is necessary to substitute  $e^{-\tau s}$  with an approximation, e.g., the Padé-approximation:

$$e^{-\tau s} \approx \frac{1 - k_1 s + k_2 s^2 + \dots \pm k_n s^n}{1 + k_1 s + k_2 s^2 + \dots + k_n s^n}$$

Below we see a 1.order and a 2.order Padé-approximation:



## 5.2 MathScript

MathScript has a built-in function called **pade** for creating transfer functions for time-delays:

Function	Description	Example
<b>pade</b>	Incorporates time delays into a system model using the Pade approximation method, which converts all residuals. You must specify the delay using the set function. You also can use this function to calculate coefficients of numerator and denominator polynomial functions with a specified delay.	<pre>&gt;[num, den] = pade(delay, order) &gt;[A, B, C, D] = pade(delay, order)</pre>
<b>sys_order1</b>		<pre>&gt;K=4; T=3; delay=5; &gt;H = sys_order1(K, T, delay)</pre>
<b>set</b>		<pre>&gt;H = set(H1, 'inputdelay', delay);</pre>
<b>series</b>		<pre>&gt;H = series(H1, H2);</pre>

### Example:

Here are some examples of how to use the **pade** function:

```

SysCon = zpk(1, 3.2, 6)
SysCon = set(SysCon, 'inputdelay', 6, 'outputdelay', 1.1)
SysDel = pade(SysCon, 2)

delay = 1.2
order = 3
[num, den] = pade(delay, order)

```

Or here is an alternative without using the **pade** function:

```

s=tf('s'); %Defines s to be the Laplace variable used in transfer functions
K=1; T=1; %Gain and time-constant

```

```

H1=tf(K/(T*s+1)); %Creates H as a transfer function
delay=1; %Time-delay

H2=set(H1,'inputdelay',delay);%Defines H2 as H1 but with time-delay

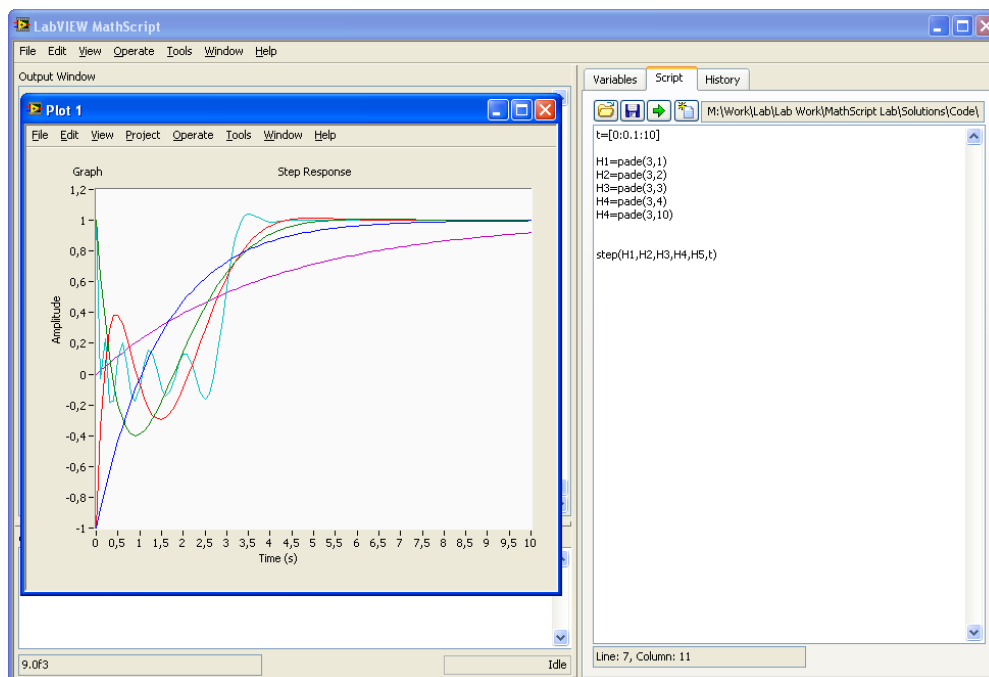
figure(1) %Plot of simulated responses will shown in Figure 1
step(H1,H2) %Simulates with unit step as input, and plots responses.

```

[End of Example]

### Example:

This example shows Pade' approximations with different orders:



[End of Example]

If we have a 1.order system, we can also use the function `sys_order1()`.

### Example:

Given the following transfer function:

$$H(s) = \frac{3}{2s + 1} e^{-4s}$$

We define the transfer function using `sys_order1` with the following code:

```

K = 3;
T = 2;

```

```

delay = 4;
H = sys_order1(K, T, delay)

```

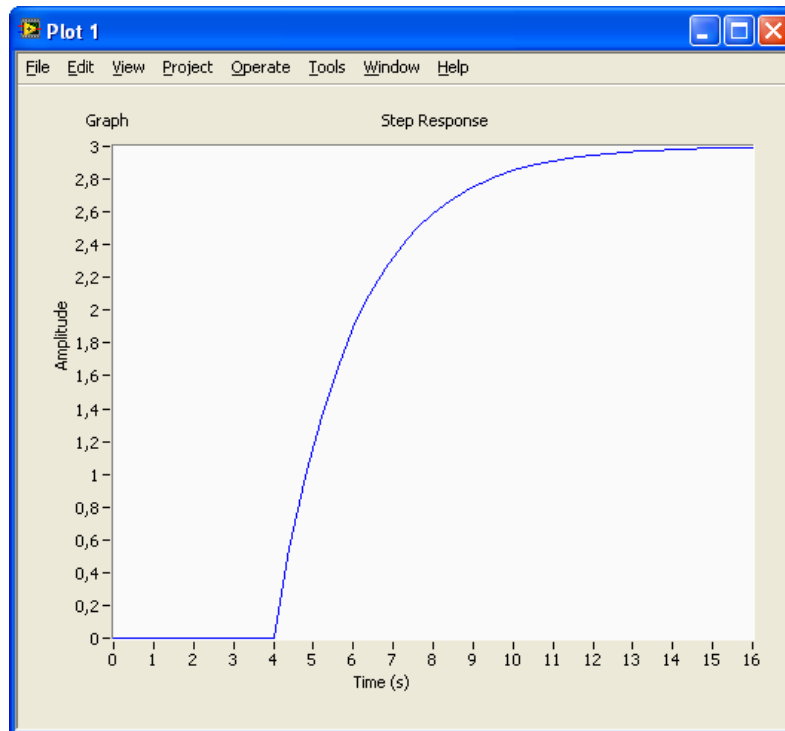
In addition we find the step response:

```

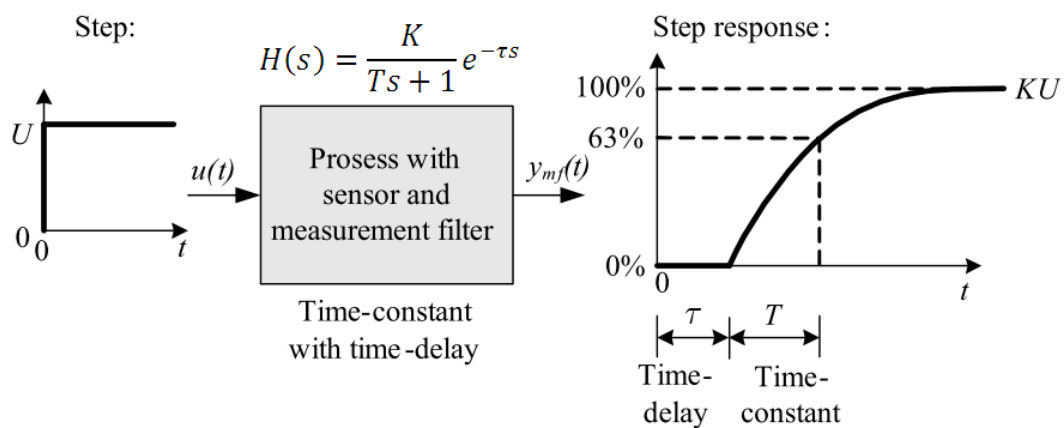
step(H)

```

This gives the following plot:

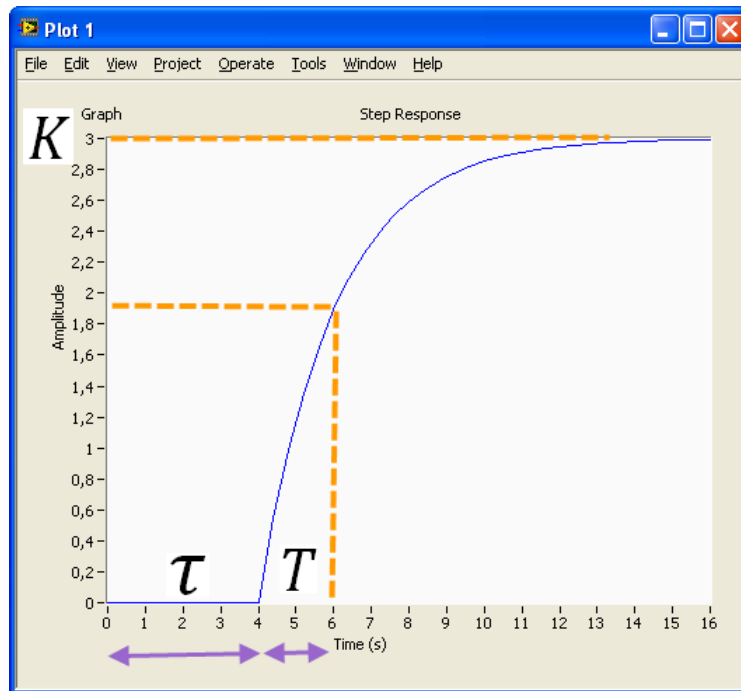


We know that a step response of such a transfer function has the following characteristics:



[Figure: F. Haugen, Advanced Dynamics and Control: TechTeach, 2010]

So we can easily find  $K$ ,  $T$  and  $\tau$  from the graph:



Another way to define  $H(s) = \frac{3}{2s+1}e^{-4s}$  is using the `tf()` and `set()` functions:

```
s = tf('s')
H1 = tf(K/(T*s+1));
H = set(H1, 'inputdelay', delay);

step(H)
```

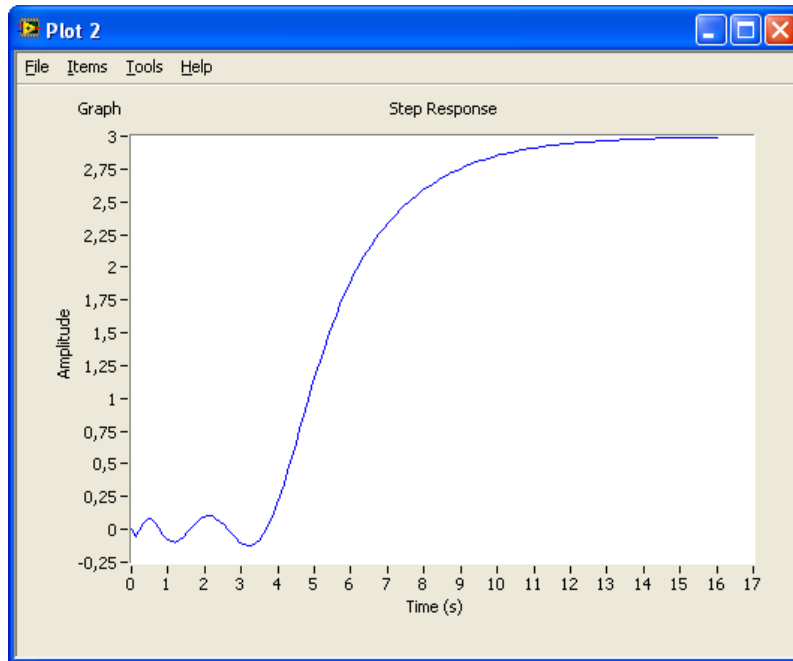
We can also combine `tf()` and the `pade()` function like this:

```
num = [K];
den = [T, 1];
H1 = tf(num, den);

order = 5;
H2 = pade(delay, order)
H = series(H1, H2)
step(H)
```

In the last example we first defined the transfer function  $\frac{3}{2s+1}$  using the `tf()` function, then we defined the time delay  $e^{-4s}$  using the `pade()` function. Finally we have combined these 2 functions using the `series()` function.

In the last example we get the following plot:



[End of Example]



# 6 Stability Analysis

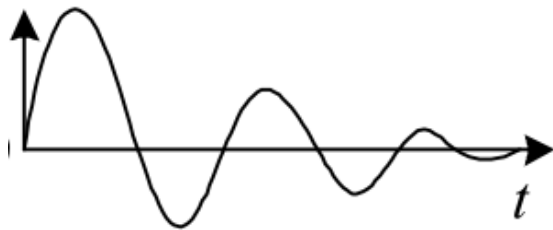
## 6.1 Introduction

A dynamic system has one of the following stability properties:

- Asymptotically stable system
- Marginally stable system
- Unstable system

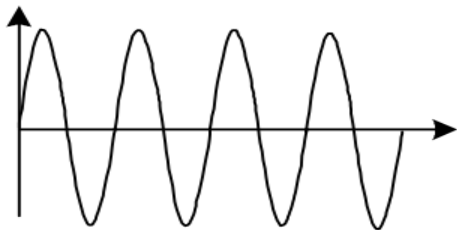
Below we see the behavior of these 3 different systems after an impulse [F. Haugen, Advanced Dynamics and Control: TechTeach, 2010]:

**Asymptotically stable system:**



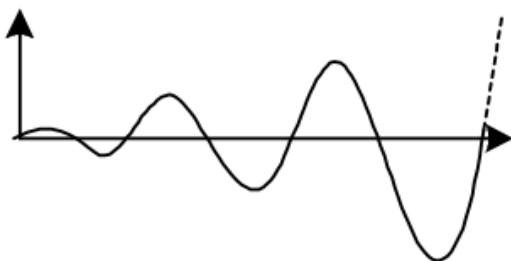
$$\lim_{t \rightarrow \infty} h(t) = 0$$

**Marginally stable system:**



$$0 < \lim_{t \rightarrow \infty} h(t) < \infty$$

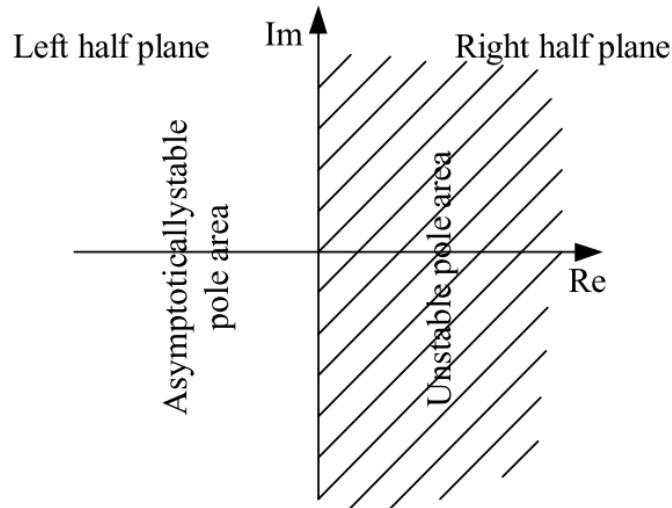
**Unstable system:**



$$\lim_{t \rightarrow \infty} h(t) = \infty$$

## 6.2 Poles

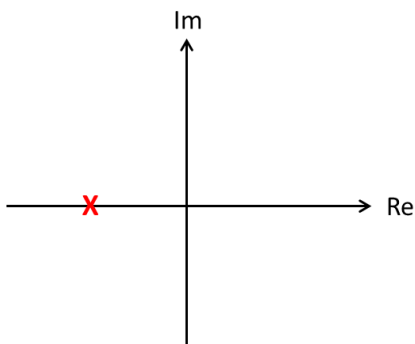
The poles are important when analysis the stability of a system. The figure below gives an overview of the poles impact on the stability of a system:



[Figure: F. Haugen, Advanced Dynamics and Control: TechTeach, 2010]

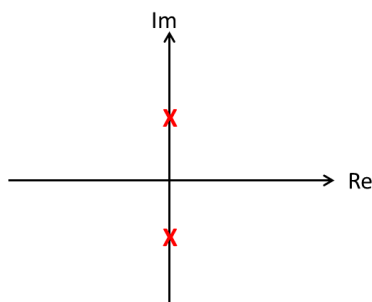
Thus, we have the following:

### Asymptotically stable system:

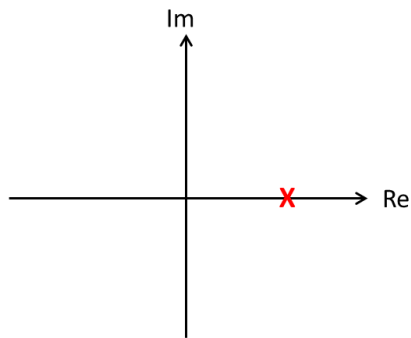


Each of the poles of the transfer function lies strictly in the left half plane (has strictly negative real part).

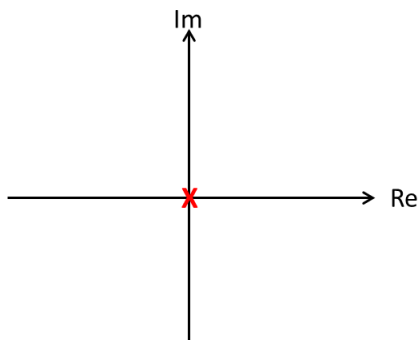
### Marginally stable system:



One or more poles lies on the imaginary axis (have real part equal to zero), and all these poles are distinct. Besides, no poles lie in the right half plane.

**Unstable system:**

At least one pole lies in the right half plane (has real part greater than zero).



Or: There are multiple and coincident poles on the imaginary axis.

Example: double integrator  $H(s) = \frac{1}{s^2}$

**Example:**

Given the following system:

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

We will analyze the stability of this system. In order to do that we will plot the step response and find the poles for this system.

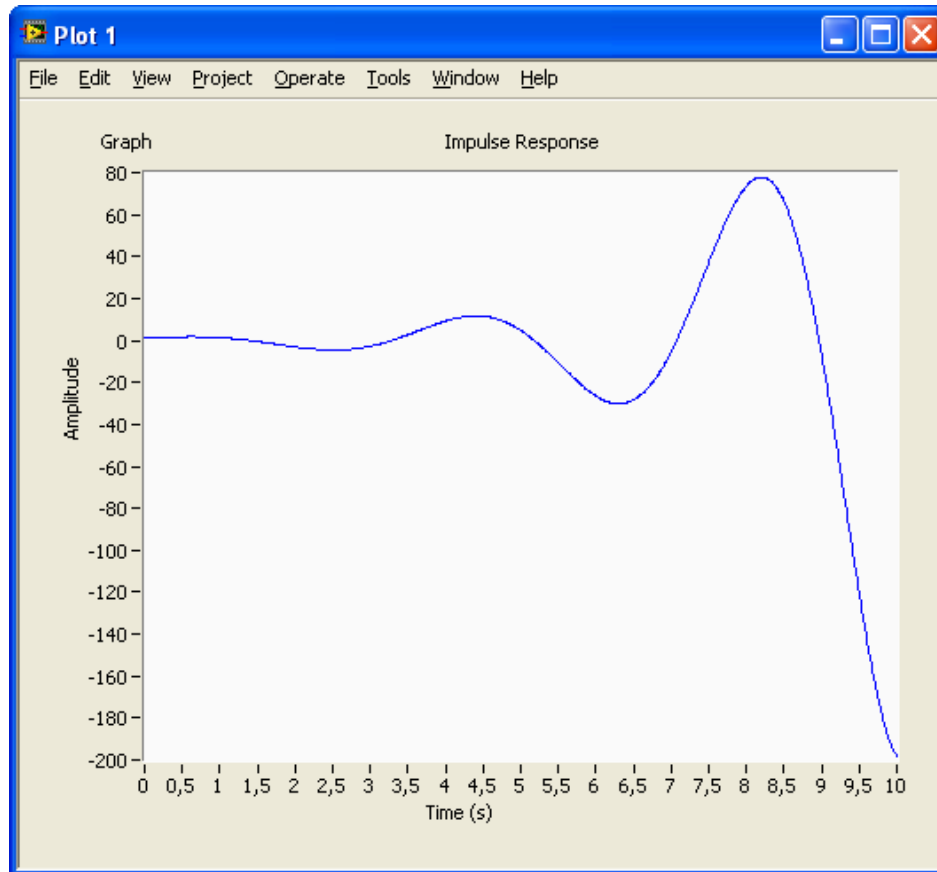
We start by defining the transfer function and plotting the impulse response:

```
clear
clc

% Define Transfer Function
num=[1,1];
den=[1,-1,3];
H=tf(num,den);

% Step Response
t=[0:0.01:10];
impulse(H,t);
```

The impulse becomes:



→ From the plot we see that the system is **unstable**.

Next we find the poles for the system:

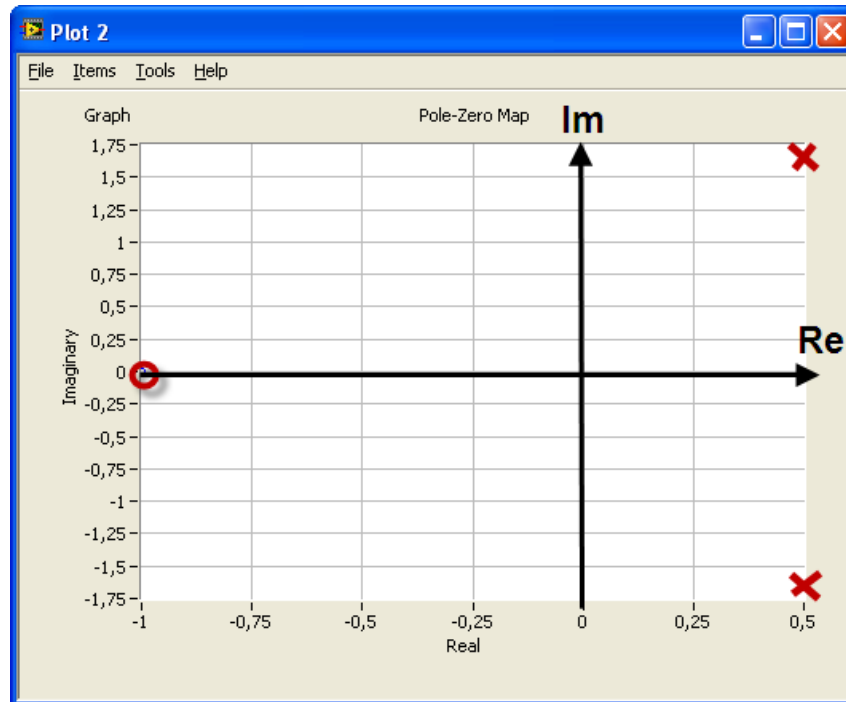
```
poles (H)
pzgraph (H)
```

The poles are as follows (found from the built-in **poles** function):

$$0.5 + 1.6583i$$

$$0.5 - 1.6583i$$

We have also used the built-in **pzgraph** function in order to plot the poles (and zeros):

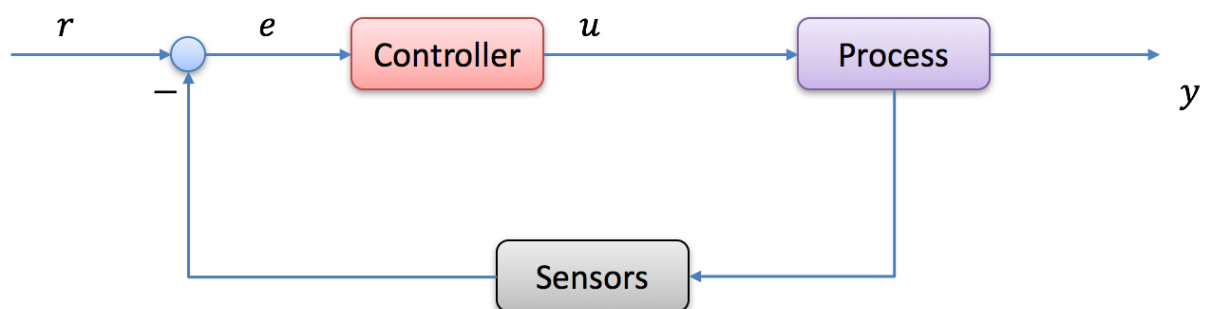


→ We see from the plot that the poles (red cross) lies in the right half plane (has real part greater than zero) and that there are multiple poles on the imaginary axis, which indicate that the system is unstable.

[End of Example]

## 6.3 Feedback Systems

Here are some important transfer functions to determine the stability of a feedback system. Below we see a typical feedback system.



### 6.3.1 Loop Transfer function

The **Loop transfer function**  $L(s)$  (Norwegian: “Sløyfetransferfunksjonen”) is defined as follows:

$$L(s) = H_c(s)H_p(s)H_m(s)$$

Where

$H_c(s)$  is the Controller transfer function

$H_p(s)$  is the Process transfer function

$H_m(s)$  is the Measurement (sensor) transfer function

Note! Another notation for  $L$  is  $H_0$

### 6.3.2 Tracking transfer function

The **Tracking transfer function  $T(s)$**  (Norwegian: “Følgeforholdet”) is defined as follows:

$$T(s) = \frac{y(s)}{r(s)} = \frac{H_c H_p H_m}{1 + H_c H_p H_m} = \frac{L(s)}{1 + L(s)} = 1 - S(s)$$

The **Tracking Property** (Norwegian: “følgeegenskaper”) is good if the tracking function  $T$  has value equal to or close to 1:

$$|T| \approx 1$$

### 6.3.3 Sensitivity transfer function

The **Sensitivity transfer function  $S(s)$**  (Norwegian: “Sensitivitetsfunksjonen/avviksforholdet”) is defined as follows:

$$S(s) = \frac{e(s)}{r(s)} = \frac{1}{1 + L(s)} = 1 - T(s)$$

The **Compensation Property** is good if the sensitivity function  $S$  has a small value close to zero:

$$|S| \approx 0 \text{ or } |S| \ll 1$$

Note!

$$T(s) + S(s) = \frac{L(s)}{1 + L(s)} + \frac{1}{1 + L(s)} \equiv 1$$

### 6.3.4 Characteristic Polynomial

We have that:

$$L(s) = \frac{n_L(s)}{d_L(s)}$$

And:

$$T(s) = \frac{y(s)}{r(s)} = \frac{L(s)}{1 + L(s)} = \frac{\frac{n_L(s)}{d_L(s)}}{1 + \frac{n_L(s)}{d_L(s)}} = \frac{n_L(s)}{d_L(s) + n_L(s)}$$

Where  $n_L(s)$  and  $d_L(s)$  numerator and the denominator of the Loop transfer function  $L(s)$ .

The Characteristic Polynomial for the control system then becomes:

$$a(s) = d_L(s) + n_L(s)$$

# 7 Frequency Response

## 7.1 Introduction

The frequency response of a system is a frequency dependent function which expresses how a sinusoidal signal of a given frequency on the system input is transferred through the system. Each frequency component is a sinusoidal signal having a certain amplitude and a certain frequency.

The frequency response is an important tool for analysis and design of signal filters and for analysis and design of control systems. The frequency response can be found experimentally or from a transfer function model.

We can find the frequency response of a system by exciting the system with a sinusoidal signal of amplitude  $A$  and frequency  $\omega$  [rad/s] (Note:  $\omega = 2\pi f$ ) and observing the response in the output variable of the system.

The frequency response of a system is defined as the steady-state response of the system to a sinusoidal input signal. When the system is in steady-state it differs from the input signal only in amplitude/gain ( $A$ ) and phase lag ( $\phi$ ).

If we have the input signal:

$$u(t) = U \sin \omega t$$

The steady-state output signal will be:

$$y(t) = \underbrace{UA}_Y \sin (\omega t + \phi)$$

Where  $A = \frac{Y}{U}$  is the ratio between the amplitudes of the output signal and the input signal (in steady-state).

$A$  and  $\phi$  is a function of the frequency  $\omega$  so we may write  $A = A(\omega)$ ,  $\phi = \phi(\omega)$

For a transfer function

$$H(S) = \frac{y(s)}{u(s)}$$

We have that:

$$\boxed{H(j\omega) = |H(j\omega)|e^{j\angle H(j\omega)}}$$



Where  $H(j\omega)$  is the frequency response of the system, i.e., we may find the frequency response by setting  $s = j\omega$  in the transfer function. Bode diagrams are useful in frequency response analysis. The Bode diagram consists of 2 diagrams, the Bode magnitude diagram,  $A(\omega)$  and the Bode phase diagram,  $\phi(\omega)$ .

The **Gain** function:

$$A(\omega) = |H(j\omega)|$$

The **Phase** function:

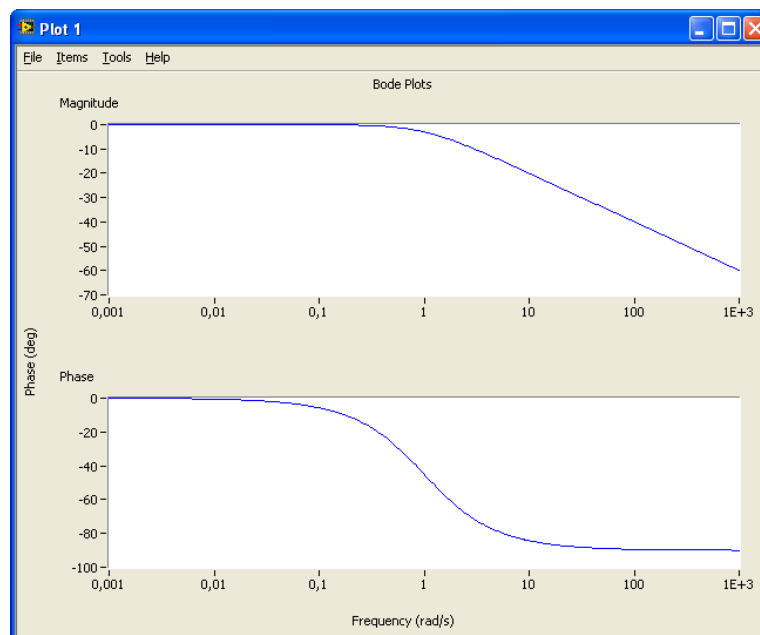
$$\phi(\omega) = \angle H(j\omega)$$

The  $A(\omega)$ -axis is in decibel (dB), where the decibel value of  $x$  is calculated as:  $x[\text{dB}] = 20\log_{10}x$

The  $\phi(\omega)$ -axis is in degrees (not radians!)

Here you will learn to plot the frequency response in a Bode diagram.

Below we see an example of a Bode plot created in MathScript:



## 7.2 MathScript

MathScript has several functions for Frequency responses:

Function	Description	Example
<b>bode</b>	Creates the Bode magnitude and Bode phase plots of a system model. You also can use this function to return the magnitude and phase values of a model at frequencies you specify. If you	<pre>&gt;num=[4]; &gt;den=[2, 1]; &gt;H = tf(num, den) &gt;bode(H)</pre>

	do not specify an output, this function creates a plot.	
<b>bodemag</b>	Creates the Bode magnitude plot of a system model. If you do not specify an output, this function creates a plot.	<pre>&gt;[mag, wout] = bodemag(SysIn) &gt;[mag, wout] = bodemag(SysIn, [wmin wmax]) &gt;[mag, wout] = bodemag(SysIn, wlist)</pre>
<b>margin</b>	Calculates and/or plots the smallest gain and phase margins of a single-input single-output (SISO) system model. The gain margin indicates where the frequency response crosses at 0 decibels. The phase margin indicates where the frequency response crosses -180 degrees. Use the margins function to return all gain and phase margins of a SISO model.	<pre>&gt;num = [1] &gt;den = [1, 5, 6] &gt;H = tf(num, den) margin(H)</pre>
<b>margins</b>	Calculates all gain and phase margins of a single-input single-output (SISO) system model. The gain margins indicate where the frequency response crosses at 0 decibels. The phase margins indicate where the frequency response crosses -180 degrees. Use the margin function to return only the smallest gain and phase margins of a SISO model.	<pre>&gt;[gmf, gm, pmf, pm] = margins(H)</pre>

## 7.3 Examples

### Example:

We have the following transfer function

$$H(s) = \frac{y(s)}{u(s)} = \frac{1}{s + 1}$$

Below we see the script for creating the **frequency response** of the system in a bode plot using the **bode** function in MathScript. Use the **grid** function to apply a grid to the plot.

```
% Transfer function H=1/(s+1)
num=[1];
den=[1, 1];
H = tf(num, den)
bode (H);
```

[End of Example]

### Example:

In this example we will use 3 different methods to find  $A$  and  $\phi$  for a given frequency  $\omega$ .

Given the following system:

$$H = \frac{K}{Ts + 1}$$

Set  $K = 1$ ,  $T = 1$

The input signal is given by:

$$u(t) = U \sin \omega t$$

The steady-state output signal will then be:

$$y(t) = \underbrace{UA}_Y \sin(\omega t + \phi)$$

The gain is given by:

$$A = \frac{Y}{U}$$

The phase lag is given by:

$$\phi = -\omega\Delta t \text{ [rad/s]}$$

**Method 1:** We create a MathScript program where we define the transfer function and define the input signal and plot it.

We will then use the **lsim** function in MathScript to plot the output signal for a given frequency,  $\omega = 1$ , in the same plot. We set  $U = 1$  in this example.

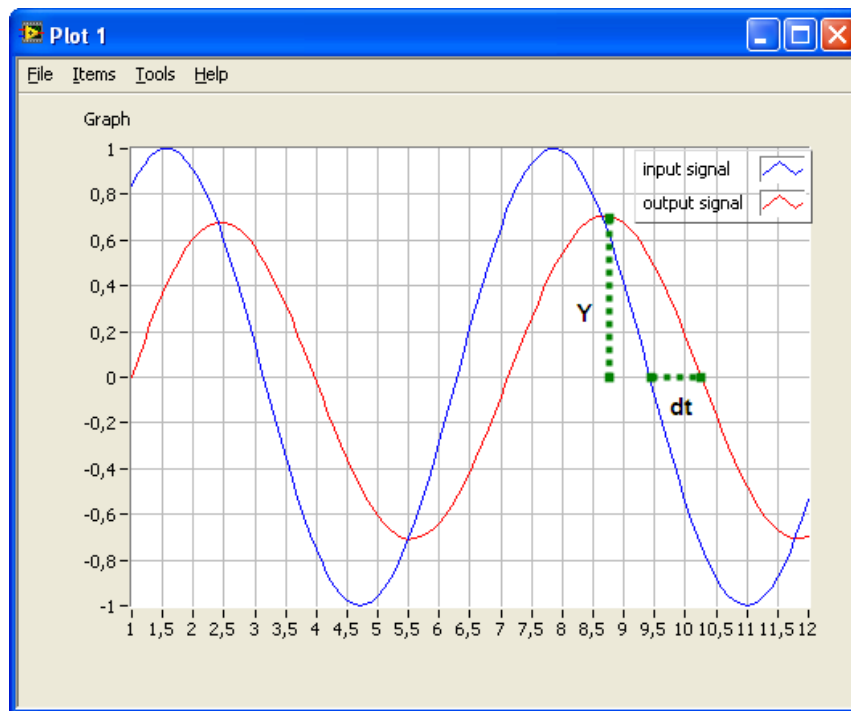
The following code will do this:

```
% Define Transfer function
K = 1;
T = 1;
num = [K];
den = [T, 1];
H = tf(num, den);

% Define input signal
t = [1: 0.1 : 12];
w = 1;
U = 1;
u = U*sin(w*t);
figure(1)
plot(t, u)

% Output signal
hold on
lsim(H, 'r', u, t)
grid on
hold off
legend('input signal', 'output signal')
```

This gives the following plot



From the plot above we get the following values:

$$Y = 0.68$$

$$\Delta t = 0.8$$

We use the following Script to calculate  $A_{dB}$  and  $\phi_{degrees}$ :

```
% Values found from plot1 for w=1
Y = 0.68;
A = Y/U;
AdB = 20*log10(A)

dt = 0.8;
phi = -w*dt; %[rad]
phi_degrees = phi*180/pi %[degrees]
```

This gives:

$$A = 0.68, \quad A_{dB} = -3.35[dB]$$

$$\phi = -0.8 \text{ rad}, \quad \phi_{deg} = -45.9 \text{ degrees}$$

**Method 2:** Next we will use the `bode` function to plot the frequency response/Bode plot to see if we get the same results.

The code for this is:

```
% Define Transfer function
K = 1;
T = 1;
```

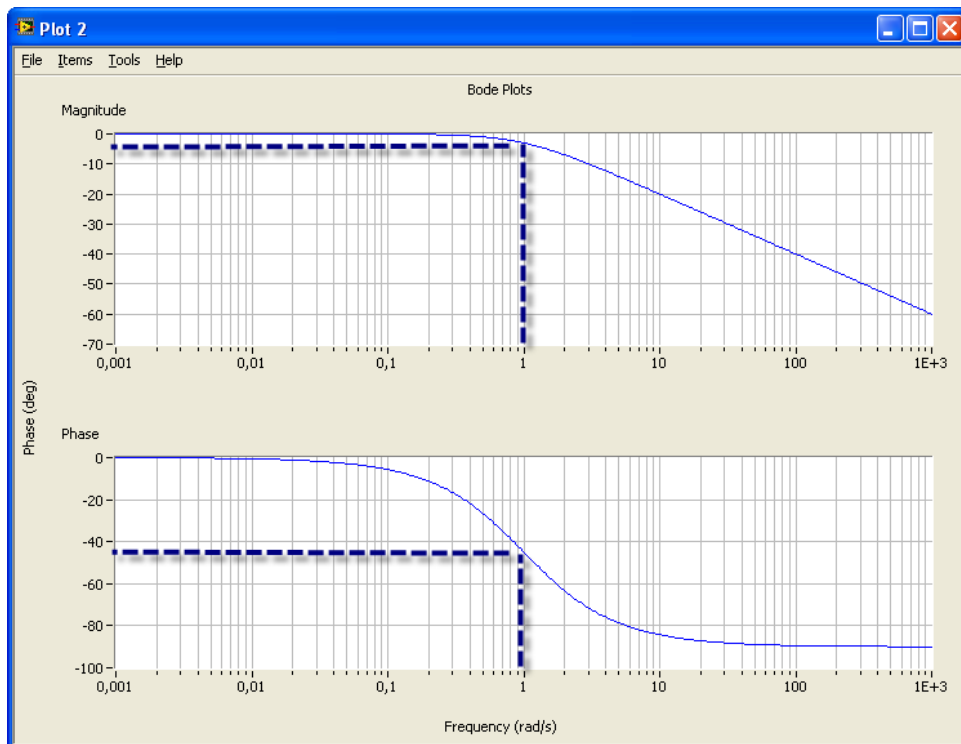
```

num = [K];
den = [T, 1];
H = tf(num, den);

%Bode plot
figure(2)
bode(H)
subplot(2,1,1)
grid on
subplot(2,1,2)
grid on

```

This gives:



We use the Bode plot to find  $A_{dB}$  and  $\phi_{degrees}$  for  $\omega = 1$

→ As you can see from the plot above we get the same results.

**Method 3:** Here we will use the `bode` function to calculate the exact values and compare with the other methods.

The MathScript becomes:

```

% Define Transfer function
K = 1;
T = 1;
num = [K];
den = [T, 1];
H = tf(num, den);

```

```
%Calculated magnitude and phase values for some given frequencies
wlist = [0.001, 0.01, 0.1, 1, 3, 5, 10, 100];
[mag, phase, wout] = bode(H, wlist);
magdB = 20*log10(mag)
phase
```

This gives:

```
magdB =
    -4.3429e-006
    -0.0004
    -0.0432
    -3.0103
    -10
    -14.1497
    -20.0432
    -40.0004

phase =

    -0.0573
    -0.5729
    -5.7106
    -45
    -71.5651
    -78.6901
    -84.2894
    -89.4271
```

→ we get the same results here also (as expected).

[End of Example]

### Example:

We have the following transfer function:

$$H(S) = \frac{4}{2s + 1}$$

**Break frequency:**

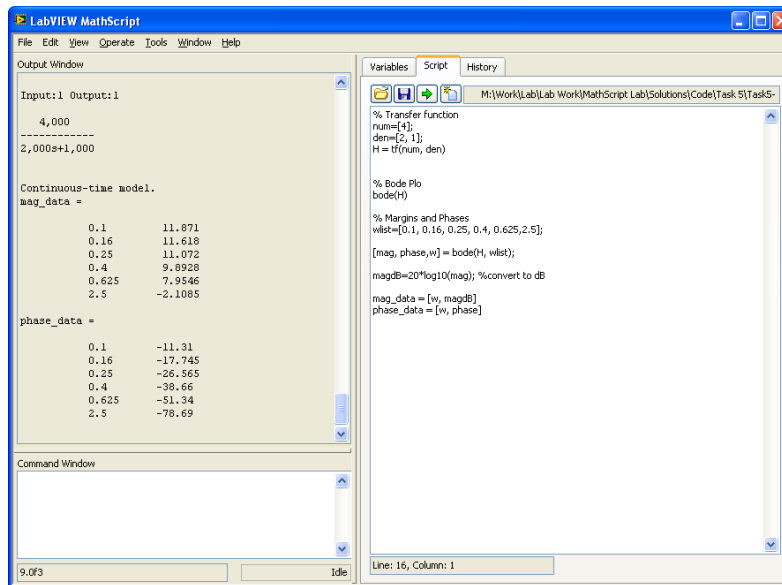
$$\omega = \frac{1}{T} = \frac{1}{2} = \underline{0.5}$$

The mathematical expressions for  $A(\omega)$  and  $\phi(\omega)$ :

$$|H(j\omega)|_{dB} = \underline{20\log 4 - 20\log\sqrt{(2\omega)^2 + 1}}$$

$$\angle H(j\omega) = \underline{-\arctan(2\omega)}$$

Frequency response of the system in a bode plot using the `bode` function in MathScript:

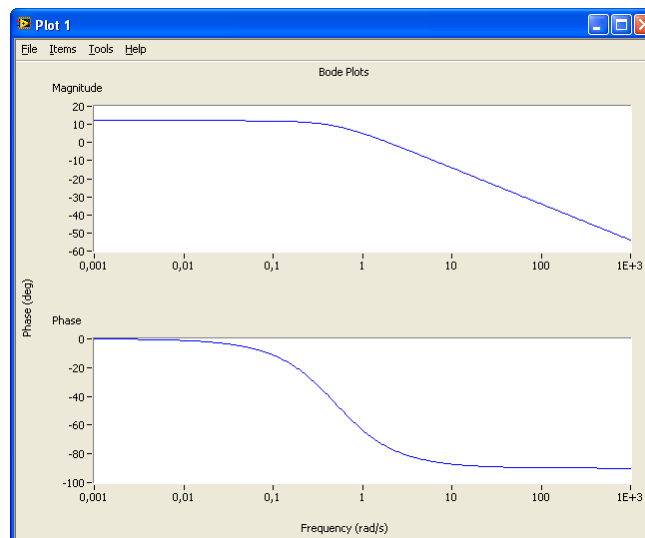


**MathScript Code:**

```

% Transfer function
num=[4];
den=[2, 1];
H = tf(num, den)
% Bode Plot
bode(H)
% Margins and Phases
wlist=[0.1, 0.16, 0.25, 0.4, 0.625,2.5];
[mag, phase,w] = bode(H, wlist);
magdB=20*log10(mag); %convert to dB
mag_data = [w, magdB]
phase_data = [w, phase]
  
```

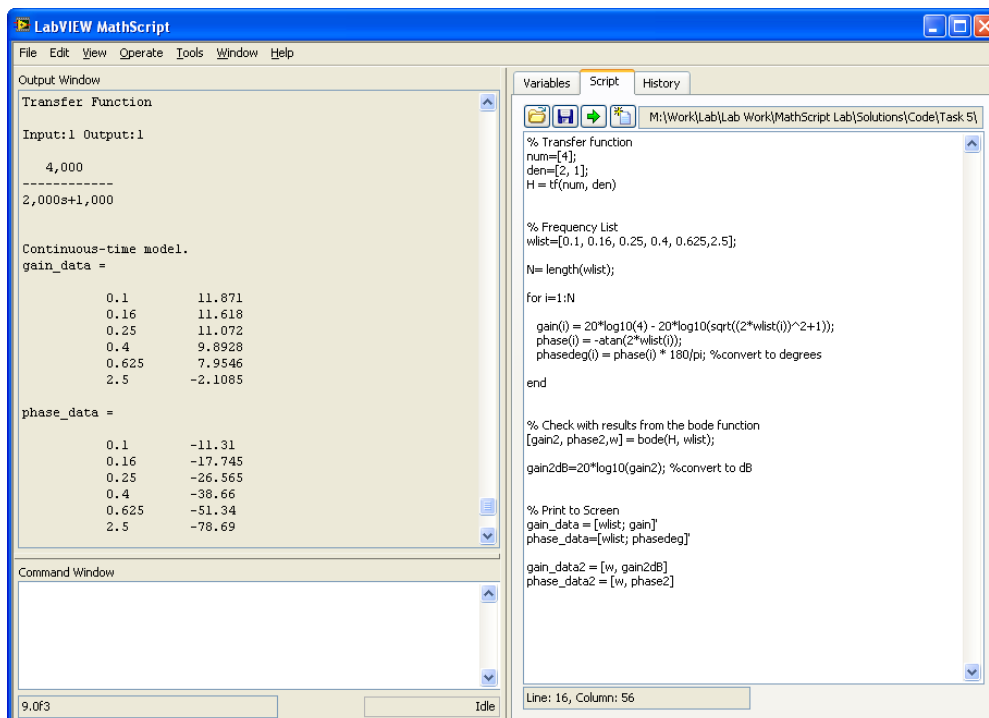
This gives:



From the code above we get  $A(\omega)$  and  $\phi(\omega)$  for the following frequencies using MathScript code:

$\omega$	$A(\omega)$	$\phi(\omega)$
0.1	11.9	-11.3
0.16	11.6	-17.7
0.25	11.1	-26.5
0.4	9.9	-38.7
0.625	7.8	-51.3
2.5	-2.1	-78.6

We find  $A(\omega)$  and  $\phi(\omega)$  for the same frequencies above using the mathematical expressions for  $A(\omega)$  and  $\phi(\omega)$  and a For Loop in MathScript. We define a vector  $w=[0.1, 0.16, 0.25, 0.4, 0.625, 2.5]$ .



```

File Edit View Operate Tools Window Help
Output Window
Transfer Function
Input:1 Output:1
-----
4,000
-----
2,000s+1,000

Continuous-time model.
gain_data =

    0.1    11.871
    0.16   11.618
    0.25   11.072
    0.4    9.8928
    0.625  7.9546
    2.5   -2.1085

phase_data =

    0.1    -11.31
    0.16   -17.745
    0.25   -26.565
    0.4    -38.66
    0.625  -51.34
    2.5   -78.69

Command Window
9,0f3 Idle

Variables Script History
M:\Work\Lab\Lab Work\MathScript Lab\Solutions\Code\Task 5\
% Transfer function
num=[4];
den=[2, 1];
H = tf(num, den)

% Frequency List
wlist=[0.1, 0.16, 0.25, 0.4, 0.625,2.5];

N=length(wlist);

for i=1:N

    gain(i) = 20*log10(4) - 20*log10(sqrt((2*wlist(i))^2+1));
    phase(i) = -atan(2*wlist(i));
    phasedeg(i) = phase(i) * 180/pi; %convert to degrees

end

% Check with results from the bode function
[gain2, phase2,w] = bode(H, wlist);

gain2dB=20*log10(gain2); %convert to dB

% Print to Screen
gain_data = [wlist; gain]
phase_data=[wlist; phasedeg]

gain_data2 = [w, gain2dB]
phase_data2 = [w, phase2]

Line: 16, Column: 56

```

→ We see the results are the same as the result found using the **bode** function.

[End of Example]



## 7.4 Standard Transfer functions

Here we will find the frequency response for the following transfer functions:

- Amplifier
- Integrator
- Derivator
- 1.order system
- 2.order system
- Zero-part
- Time delay

### 7.4.1 Amplifier (Norwegian: “Forsterker”):

The transfer function for an Amplifier is as follows:

$$H(s) = K$$

Where

$K$  is the gain

The **mathematical** expressions for  $A(\omega)$  and  $\phi(\omega)$  is as follows:

Gain:

$$A(\omega) = |H(j\omega)| = K$$

or in dB:

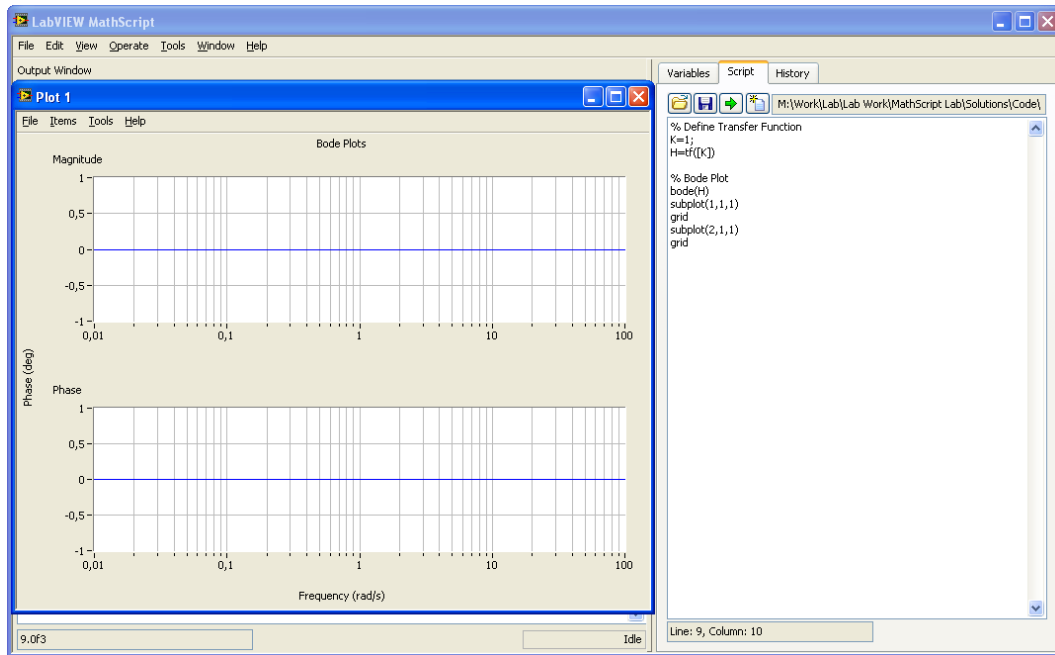
$$|H(j\omega)|_{dB} = 20\log K$$

Phase:

$$\phi(\omega) = \angle H(j\omega) = 0$$

#### Example:

We plot the **Bode** plot for the Amplifier using the bode function in MathScript (K=1):



→ We see that both  $A(\omega)$  and  $\phi(\omega)$  are independent of the frequency  $\omega$ .

[End of Example]

## 7.4.2 Integrator

The transfer function for an Integrator is as follows:

$$H(s) = \frac{K}{s}$$

Where

$K$  is the gain

The **mathematical** expressions for  $A(\omega)$  and  $\phi(\omega)$  is as follows:

Gain:

$$A(\omega) = |H(j\omega)| = \frac{K}{\omega}$$

or in dB:

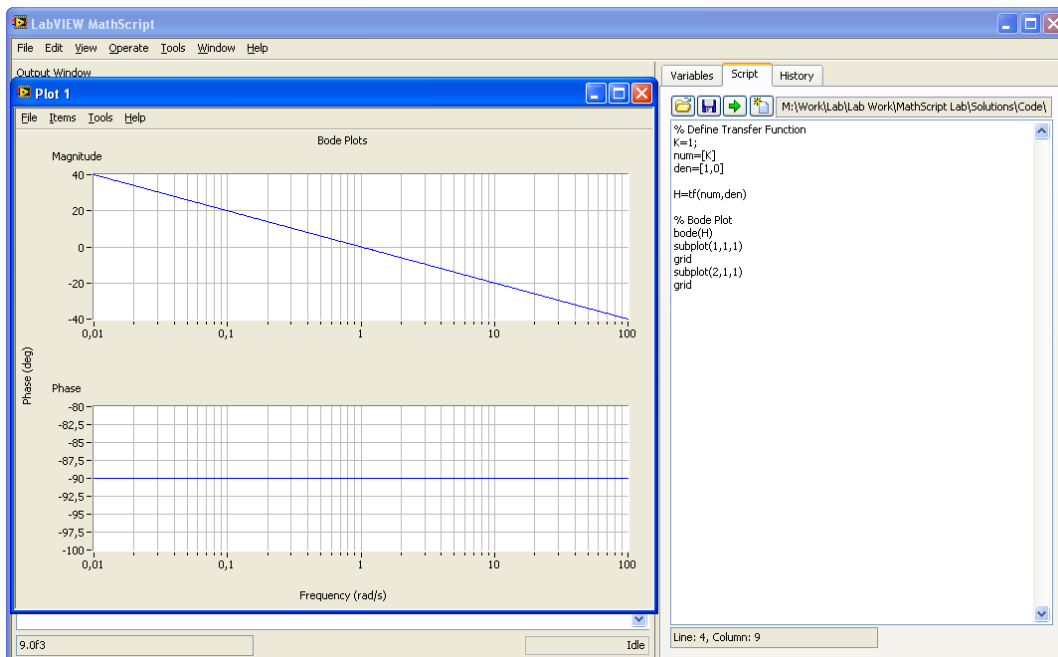
$$|H(j\omega)|_{dB} = 20 \log \frac{K}{\omega}$$

Phase:

$$\phi(\omega) = \angle H(j\omega) = -\frac{\pi}{2} \text{ rad} = -90^\circ$$

### Example:

We plot the **Bode** plot for the Integrator using the bode function in MathScript:



[End of Example]

## 7.4.3 Derivator

The transfer function for an Derivator is as follows:

$$H(s) = Ks$$

Where

$K$  is the gain

The **mathematical** expressions for  $A(\omega)$  and  $\phi(\omega)$  is as follows:

Gain:

$$A(\omega) = |H(j\omega)| = K\omega$$

or in dB:

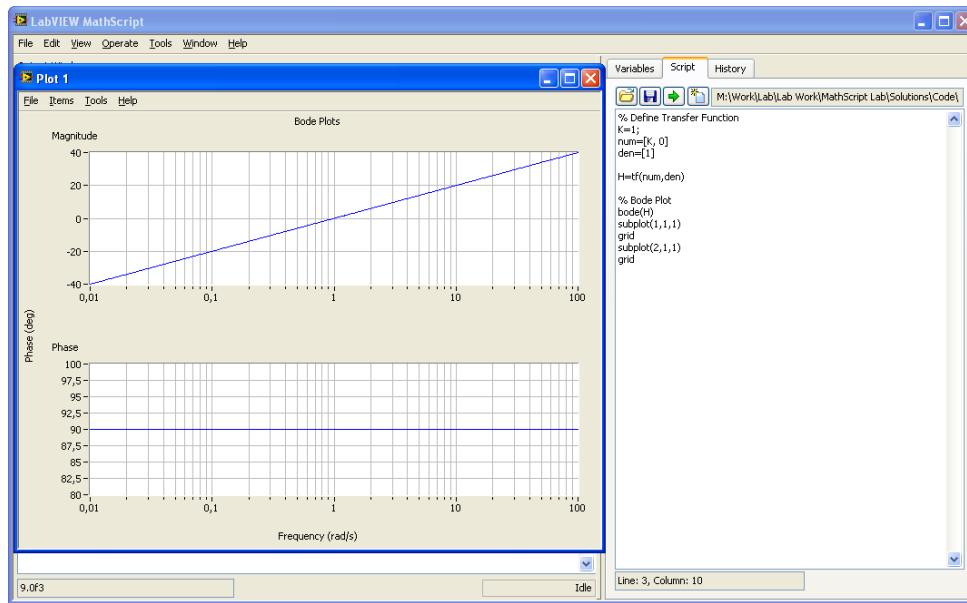
$$|H(j\omega)|_{dB} = 20 \log K\omega$$

Phase:

$$\phi(\omega) = \angle H(j\omega) = +\frac{\pi}{2} \text{ rad} = +90^\circ$$

### Example:

We plot the **Bode** plot for the Derivator using the bode function in MathScript:



[End of Example]

## 7.4.4 1. Order system

The transfer function for a 1.order system is as follows:

$$H(s) = \frac{K}{Ts + 1}$$

Where

$K$  is the gain

$T$  is the Time constant

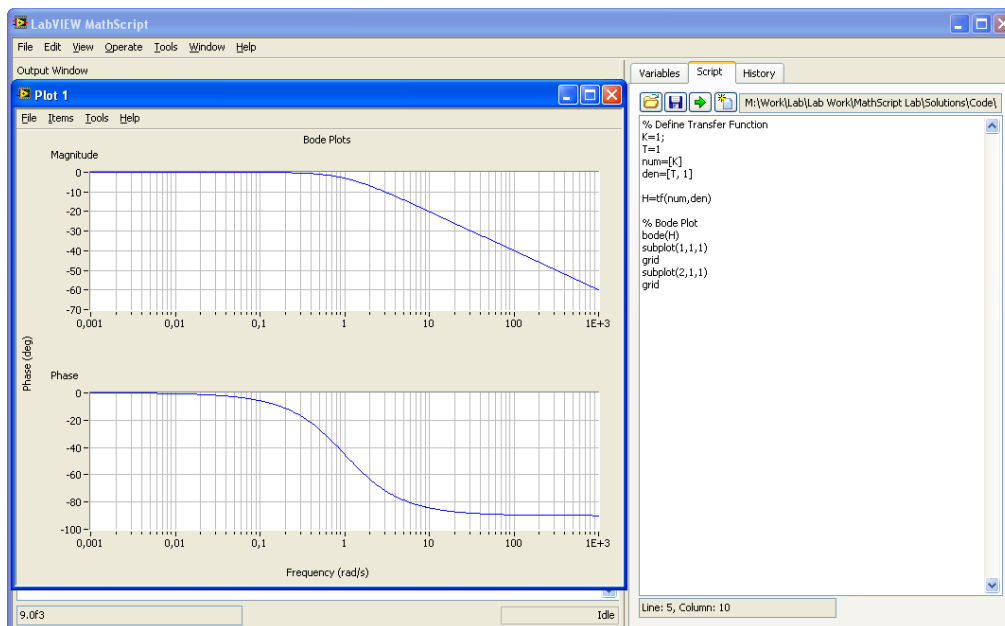
The **mathematical** expressions for  $A(\omega)$  and  $\phi(\omega)$  is as follows:

$$A(\omega) = |H(j\omega)| = \frac{K}{\omega^2 T^2 + 1}$$

$$\phi(\omega) = \angle H(j\omega) = -\arctan(\omega T)$$

### Example:

We plot the **Bode** plot for the 1.order system using the bode function in MathScript:



[End of Example]

## 7.4.5 2. Order system

The transfer function for a 2.order system is as follows:

$$H(s) = \frac{K\omega_0^2}{s^2 + 2\zeta\omega_0s + \omega_0^2} = \frac{K}{\left(\frac{s}{\omega_0}\right)^2 + 2\zeta\frac{s}{\omega_0} + 1}$$

Where

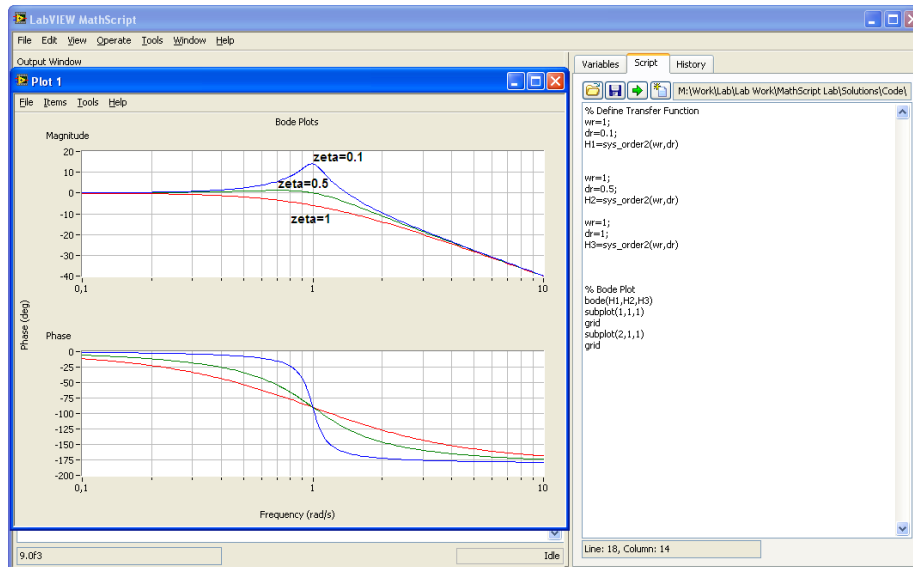
$K$  is the gain

$\zeta$  zeta is the relative damping factor

$\omega_0$ [rad/s] is the undamped resonance frequency.

### Example:

We plot the **Bode** plot for the 2.order system using the bode function in MathScript:



[End of Example]

## 7.4.6 Zero part (Norwegian: “Nullpunktsledd”)

The transfer function for a Zero part system is as follows:

$$H(s) = K(Ts + 1)$$

Where

$K$  is the gain

$T$  is the Time constant

The **mathematical** expressions for  $A(\omega)$  and  $\phi(\omega)$  is as follows:

Gain:

$$A(\omega) = |H(j\omega)| = K\sqrt{(\omega T)^2 + 1}$$

or in dB:

$$|H(j\omega)|_{dB} = 20\log K\sqrt{(\omega T)^2 + 1}$$

Phase:

$$\phi(\omega) = \angle H(j\omega) = +\arctan(\omega T)$$

## 7.4.7 Time delay (Norwegian: “Tidsforsinkelse”)

The transfer function for a Time Delay is as follows:

$$H(s) = Ke^{-\tau s}$$

Where

$K$  is the gain

$\tau$  is the time-delay

The **mathematical** expressions for  $A(\omega)$  and  $\phi(\omega)$  is as follows:

Gain:

$$A(\omega) = |H(j\omega)| = K$$

Phase:

$$\phi(\omega) = \angle H(j\omega) = -\omega\tau \text{ rad} = -\omega\tau \frac{180}{\pi} \text{ degrees}$$

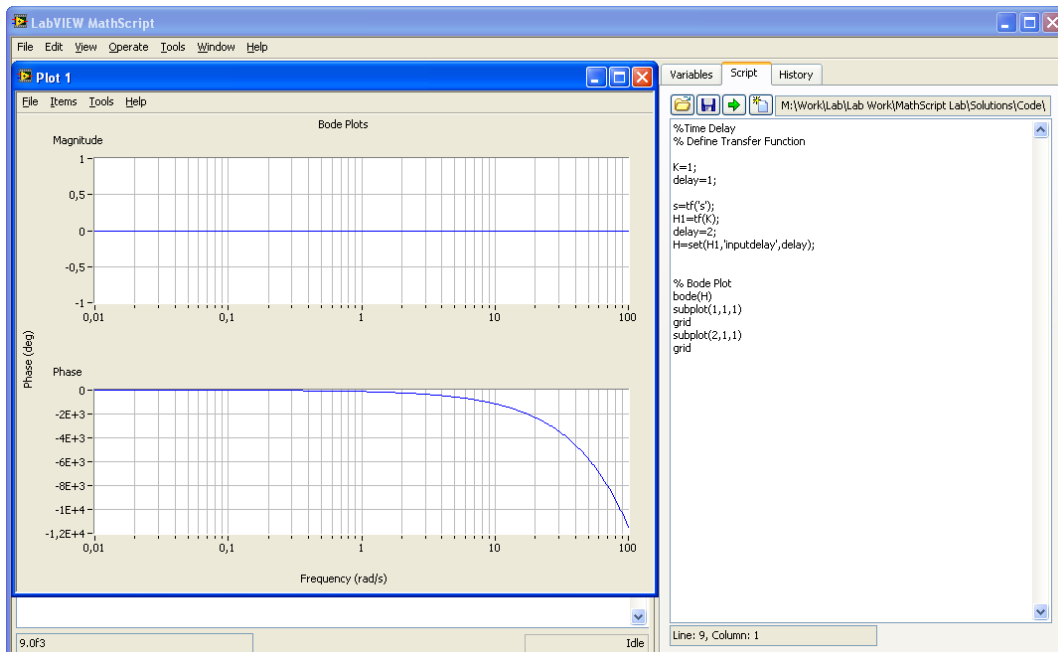
**Note!**

$$2\pi \text{ rad} = 360^\circ$$

$$\pi \text{ rad} = 180^\circ$$

### Example:

We plot the **Bode** plot for the Time delay using the bode function in MathScript:

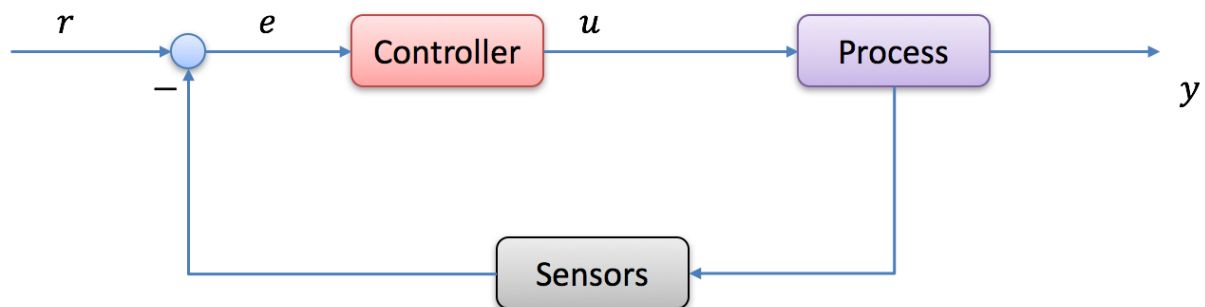


[End of Example]

# 8 Frequency response Analysis

## 8.1 Introduction

Here are some important transfer functions to determine the stability of a feedback system. Below we see a typical feedback system.



The **Loop transfer function**  $L(s)$  (Norwegian: “Sløyfetransferfunksjonen”) is defined as follows:

$$L(s) = H_c H_p H_m$$

Where

$H_c$  is the Controller transfer function

$H_p$  is the Process transfer function

$H_m$  is the Measurement (sensor) transfer function

Note! Another notation for  $L$  is  $H_0$

The **Tracking transfer function**  $T(s)$  (Norwegian: “Følgeforholdet”) is defined as follows:

$$T(s) = \frac{y(s)}{r(s)} = \frac{H_c H_p H_m}{1 + H_c H_p H_m} = \frac{L(s)}{1 + L(s)} = 1 - S(s)$$

The **Tracking Property** (Norwegian: “følgeegenskaper”) is good if the tracking function  $T$  has value equal to or close to 1:



$$|T| \approx 1$$

The **Sensitivity transfer function**  $S(s)$  (Norwegian: “Sensitivitetsfunksjonen/avviksforholdet”) is defined as follows:

$$S(s) = \frac{e(s)}{r(s)} = \frac{1}{1 + L(s)} = 1 - T(s)$$

The **Compensation Property** is good if the sensitivity function  $S$  has a small value close to zero:

$$|S| \approx 0 \text{ or } |S| \ll 1$$

Note!

$$T(s) + S(s) = \frac{L(s)}{1 + L(s)} + \frac{1}{1 + L(s)} \equiv 1$$

Frequency Response Analysis of the **Tracking Property**:

From the equations above we find:

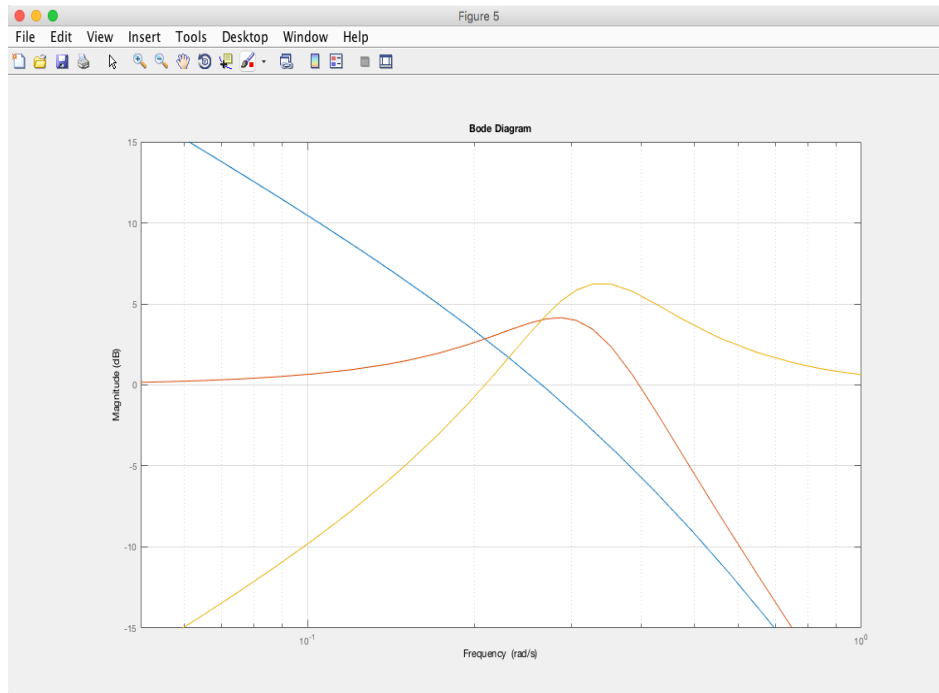
The **Tracking Property** (Norwegian: “følgeegenskaper”) is **good** if:

$$|L(j\omega)| \gg 1$$

The **Tracking Property** (Norwegian: “følgeegenskaper”) is **poor** if:

$$|L(j\omega)| \ll 1$$

**Bandwidths**  $\omega_t, \omega_c, \omega_s$  (see the sketch below)



$\omega_c$  – crossover-frequency – the frequency where the gain of the Loop transfer function  $L(j\omega)$  has the value:

$$1 = \underline{0dB}$$

$\omega_t$  – the frequency where the gain of the Tracking function  $T(j\omega)$  has the value:

$$\frac{1}{\sqrt{2}} \approx 0.71 = \underline{-3dB}$$

$\omega_s$  - the frequency where the gain of the Sensitivity transfer function  $S(j\omega)$  has the value:

$$1 - \frac{1}{\sqrt{2}} \approx 0.29 = \underline{-11dB}$$

## 8.2 MathScript

MathScript has several functions for frequency response analysis:

Function	Description	Example
<b>tf</b>	Creates system model in transfer function form. You also can use this function to state-space models to transfer function form.	<pre>&gt;num=[1]; &gt;den=[1, 1, 1]; &gt;H = <b>tf</b>(num, den)</pre>
<b>poles</b>	Returns the locations of the closed-loop poles of a system model.	<pre>&gt;num=[1] &gt;den=[1,1] &gt;H=<b>tf</b>(num,den) &gt;<b>poles</b>(H)</pre>
<b>tfinfo</b>	Returns information about a transfer function system model.	<pre>&gt;[num, den, delay, Ts] = <b>tfinfo</b>(SysInTF)</pre>
<b>series</b>	Connects two system models in series to produce a model SysSer with input and output connections you specify	<pre>&gt;Hseries = <b>series</b>(H1,H2)</pre>
<b>feedback</b>	Connects two system models together to produce a closed-loop model using negative or positive feedback connections	<pre>&gt;SysClosed = <b>feedback</b>(SysIn_1, SysIn_2)</pre>

<b>bode</b>	Creates the Bode magnitude and Bode phase plots of a system model. You also can use this function to return the magnitude and phase values of a model at frequencies you specify. If you do not specify an output, this function creates a plot.	<pre>&gt;num=[4]; &gt;den=[2, 1]; &gt;H = tf(num, den) &gt;bode(H)</pre>
<b>bodemag</b>	Creates the Bode magnitude plot of a system model. If you do not specify an output, this function creates a plot.	<pre>&gt;[mag, wout] = bodemag(SysIn) &gt;[mag, wout] = bodemag(SysIn, [wmin wmax]) &gt;[mag, wout] = bodemag(SysIn, wlist)</pre>
<b>margin</b>	Calculates and/or plots the smallest gain and phase margins of a single-input single-output (SISO) system model. The gain margin indicates where the frequency response crosses at 0 decibels. The phase margin indicates where the frequency response crosses -180 degrees. Use the margins function to return all gain and phase margins of a SISO model.	<pre>&gt;num = [1] &gt;den = [1, 5, 6] &gt;H = tf(num, den) margin(H)</pre>
<b>margins</b>	Calculates all gain and phase margins of a single-input single-output (SISO) system model. The gain margins indicate where the frequency response crosses at 0 decibels. The phase margins indicate where the frequency response crosses -180 degrees. Use the margin function to return only the smallest gain and phase margins of a SISO model.	<pre>&gt;[gmf, gm, pmf, pm] = margins(H)</pre>

### Example:

Given the following system:

**Process transfer function:**

$$H_p = \frac{K}{s} e^{-\tau s}$$

Where  $K = \frac{K_s}{\rho A}$ , where  $K_s = 0,556$ ,  $A = 13,4$ ,  $\rho = 145$  and  $\tau = 250$

**Measurement (sensor) transfer function:**

$$H_m = K_m$$

Where  $K_m = 6,67 \text{ %/m}$ .

**Controller transfer function (PI Controller):**

$$H_c = K_p + \frac{K_p}{T_i s}$$

Set  $K_p = 1,5$  og  $T_i = 1000 \text{ sec}$ .

We shall find the **Loop transfer function**  $(s)$ , **Sensitivity transfer function**  $S(s)$ , **Tracking transfer function**  $T(s)$  using the **series** and **feedback** functions in MathScript.

MathScript Code:

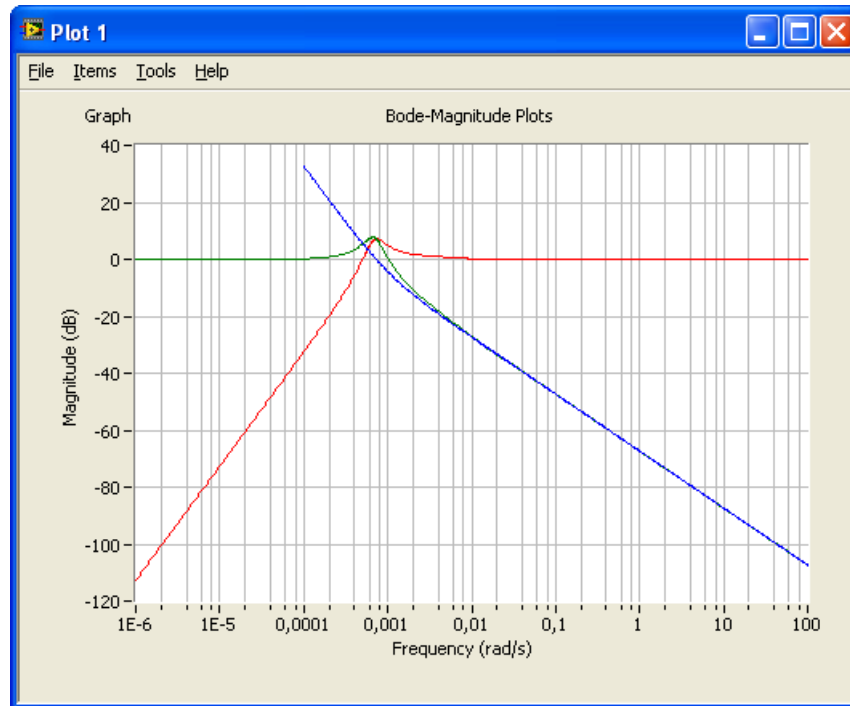
```
%Calculating control system transfer functions:
L=series(Hc,series(Hp,Hs)); %Calculating loop tranfer function
T=feedback(L,1); %Calculating tracking transfer function
S=1-T; %Calculating sensitivity transfer function
```

We plot the Bode plot for L, T and S and find the Bandwidths  $\omega_t, \omega_c, \omega_s$ :

MathScript Code:

```
bodemag(L,T,S), grid %Plots magnitude of L, T, and S in Bode diagram
```

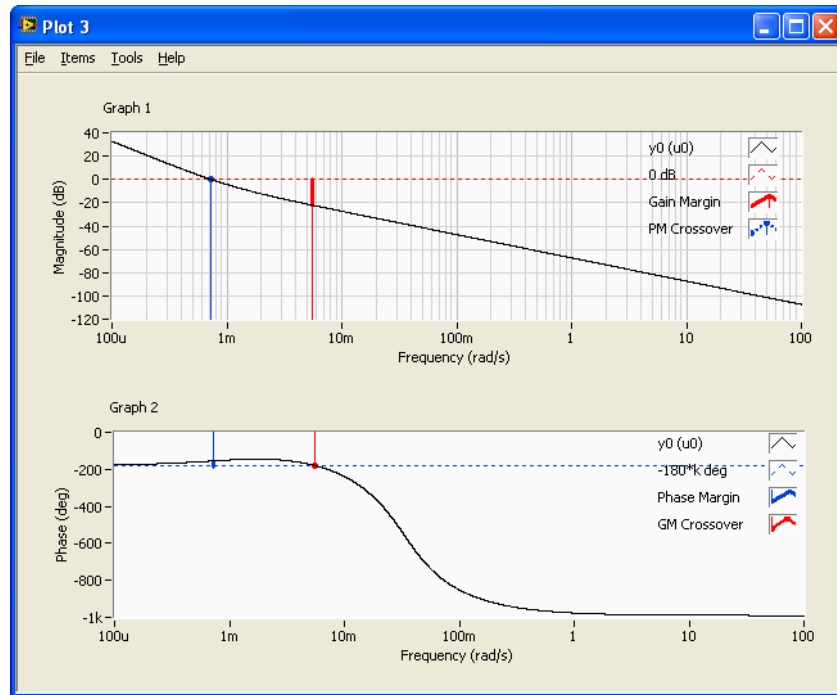
Bode plot (Magnitude only) of L, T and S:



We find the stability margins (GM, PM) of the system ( $L(s)$ ):

```
margin(L), grid %Plotting L and stability margins and crossover frequencies in Bode diagram
```

Bode plot with the stability margins (GM, PM) marked on the plot:



[End of Example]

# 9 Stability Analysis in the Frequency Domain

## 9.1 Introduction

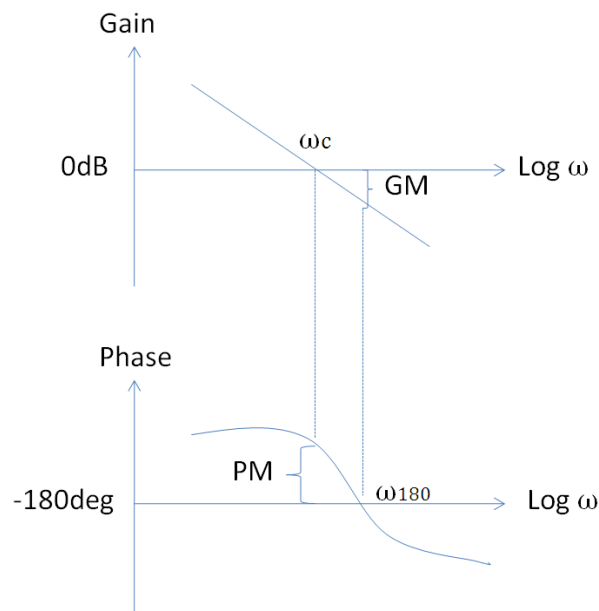
Gain Margin (GM) and Phase Margin (PM) are important design criteria for analysis of feedback control systems.

A dynamic system has one of the following stability properties:

- Asymptotically stable system
- Marginally stable system
- Unstable system

**The Gain Margin – GM ( $\Delta K$ )** is how much the loop gain can increase before the system become unstable.

The **Phase Margin - PM ( $\varphi$ )** is how much the phase lag function of the loop can be reduced before the loop becomes unstable.



Where:

- $\omega_{180}$  (gain margin frequency - gmf) is the gain margin frequency/frequencies, in radians/second. A gain margin frequency indicates where the model phase crosses -180 degrees.
- **GM** ( $\Delta K$ ) is the gain margin(s) of the system.
- $\omega_c$  (phase margin frequency - pmf) returns the phase margin frequency/frequencies, in radians/second. A phase margin frequency indicates where the model magnitude crosses 0 decibels.
- **PM** ( $\varphi$ ) is the phase margin(s) of the system.

**Note!**  $\omega_{180}$  and  $\omega_c$  are called the crossover-frequencies

The definitions are as follows:

**Gain Crossover-frequency -  $\omega_c$ :**

$$|L(j\omega_c)| = 1 = 0dB$$

**Phase Crossover-frequency -  $\omega_{180}$ :**

$$\angle L(j\omega_{180}) = -180^\circ$$

**Gain Margin - GM ( $\Delta K$ ):**

$$GM = \frac{1}{|L(j\omega_{180})|}$$

or:

$$GM [dB] = -|L(j\omega_{180})| [dB]$$

**Phase margin PM ( $\varphi$ ):**

$$PM = 180^\circ + \angle L(j\omega_c)$$

We have that:

- **Asymptotically stable system:**  $\omega_c < \omega_{180}$
- **Marginally stable system:**  $\omega_c = \omega_{180}$
- **Unstable system:**  $\omega_c > \omega_{180}$

## 9.2 MathScript

MathScript has several functions for stability analysis:

Function	Description	Example
<b>bode</b>	Creates the Bode magnitude and Bode phase plots of a system model. You also can use this function to return the magnitude and phase values of a model at frequencies you specify. If you do not specify an output, this function creates a plot.	<pre>&gt;num=[4]; &gt;den=[2, 1]; &gt;H = tf(num, den) &gt;bode(H)</pre>
<b>bodemag</b>	Creates the Bode magnitude plot of a system model. If you do	<pre>&gt;[mag, wout] = bodemag(SysIn)</pre>

not specify an output, this function creates a plot.

```
>[mag, wout] = bodemag(SysIn, [wmin
wmax])
>[mag, wout] = bodemag(SysIn,
wlist)
>num = [1]
>den = [1, 5, 6]
>H = tf(num, den)
margin(H)
```

### margin

Calculates and/or plots the smallest gain and phase margins of a single-input single-output (SISO) system model. The gain margin indicates where the frequency response crosses at 0 decibels. The phase margin indicates where the frequency response crosses -180 degrees. Use the margins function to return all gain and phase margins of a SISO model.

### margins

Calculates all gain and phase margins of a single-input single-output (SISO) system model. The gain margins indicate where the frequency response crosses at 0 decibels. The phase margins indicate where the frequency response crosses -180 degrees. Use the margin function to return only the smallest gain and phase margins of a SISO model.

```
>[gmf, gm, pmf, pm] = margins(H)
```

## Example:

Given the following system:

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

We will find the **crossover-frequencies** for the system using MathScript. We will also find also the **gain margins** and **phase margins** for the system.

We get:

The screenshot shows the LabVIEW MathScript environment. The Output Window displays the results of the margin calculations:

```
phase_data =
    0.01    -91.146
    0.1     -101.42
    0.2     -112.62
    0.5     -143.13
    1       -180
    10     -258.58
    100    -268.85

gmf =
    1.9972

gm =
    0.99931

pmf =
    21.386

pm =
    0.68233
```

The Command Window shows the status: 9.0F3 Idle.

The Script Window contains the following MathScript code:

```
% Transfer function
num=[1];
den1=[1,0];
den2=[1,1]
den3=[1,1]

den = conv(den1,conv(den2,den3));
H = tf(num, den)

% Bode Plot
bode(H)

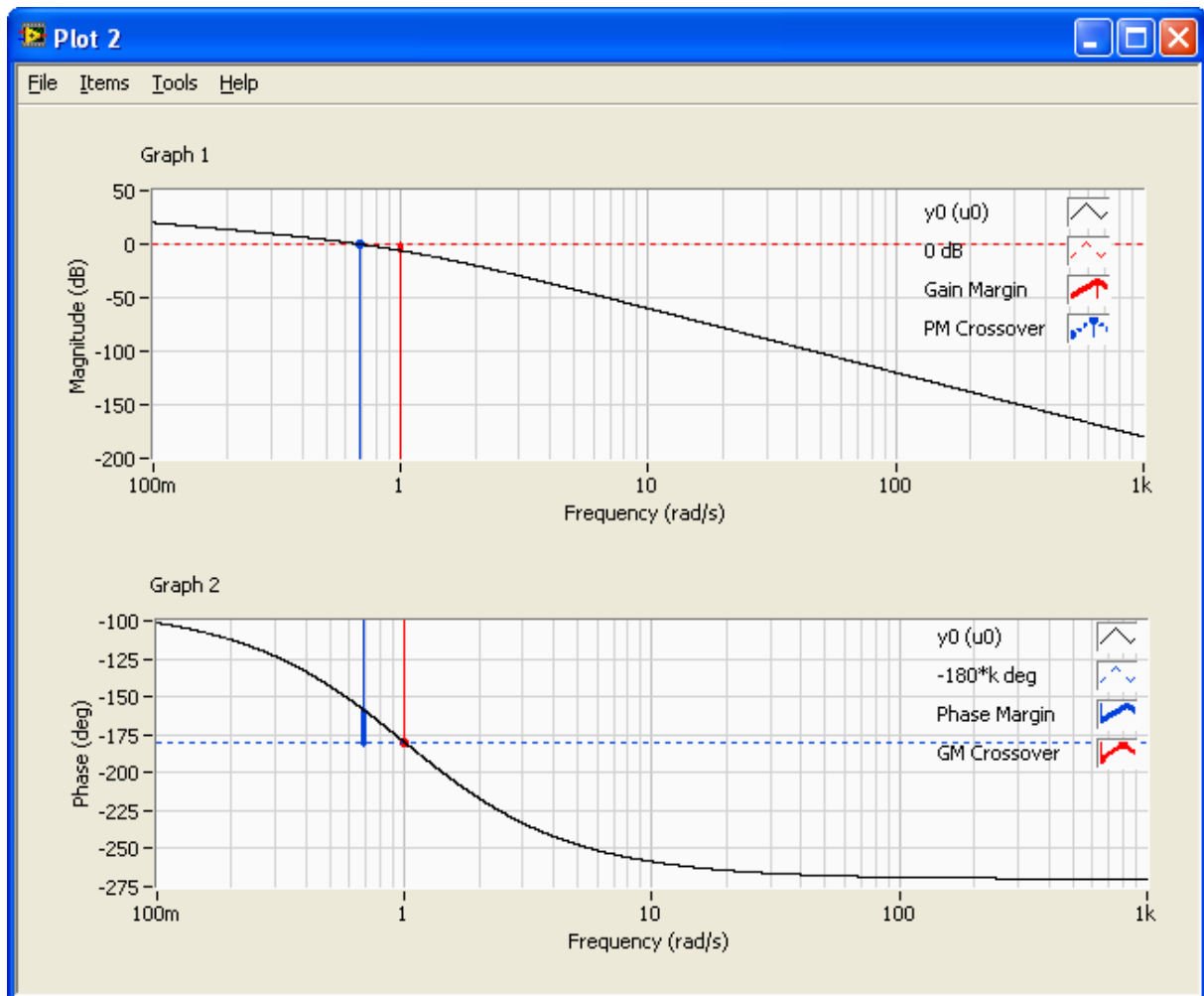
% Margins and Phases
wlist=[0.01, 0.1, 0.2, 0.5, 1, 10, 100];
[mag, phase,w] = bode(H, wlist);
magdB=20*log10(mag); %convert to dB

% [mag, phase,w] = bode(H);
mag_data = [w, magdB]
phase_data = [w, phase]

% Crossover Frequency-----
[gmf, gm, pmf, pm] = margins(H)
margin(H)
```



Below we see the Bode diagram with the crossover-frequency and the gain margin and phase margin for the system plotted in:



[End of Example]

### Example:

Given the following system:

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

→ The system is unstable and Frequency Response gives meaning only for stable systems.

**Note!** The frequency response of a system is defined as the steady-state response of the system to a sinusoidal input signal.

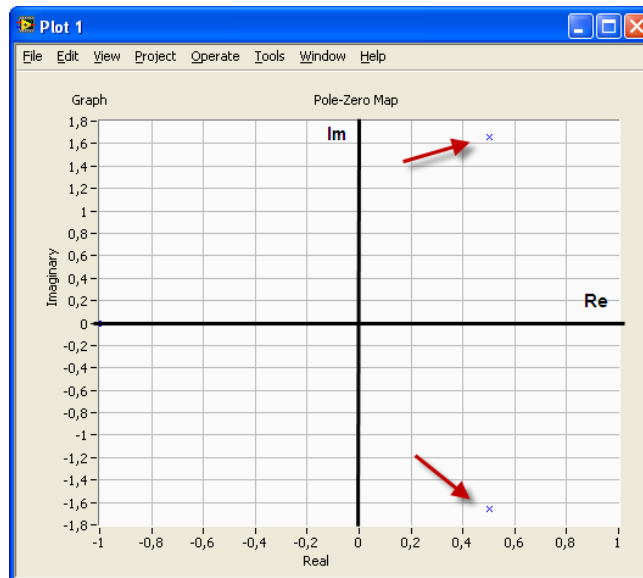
The Bode diagram for unstable systems don't show what happens with the sinusoidal signal of a given frequency when the system input is transferred through the system because it never reach steady state.

We see that the system is unstable because some of the coefficients in the denominator polynomial  $s^2 - s + 3$  are negative.

We confirm this by some simulations and finding the poles for the system:

```
poles(H)
pzgraph(H)
```

This gives:



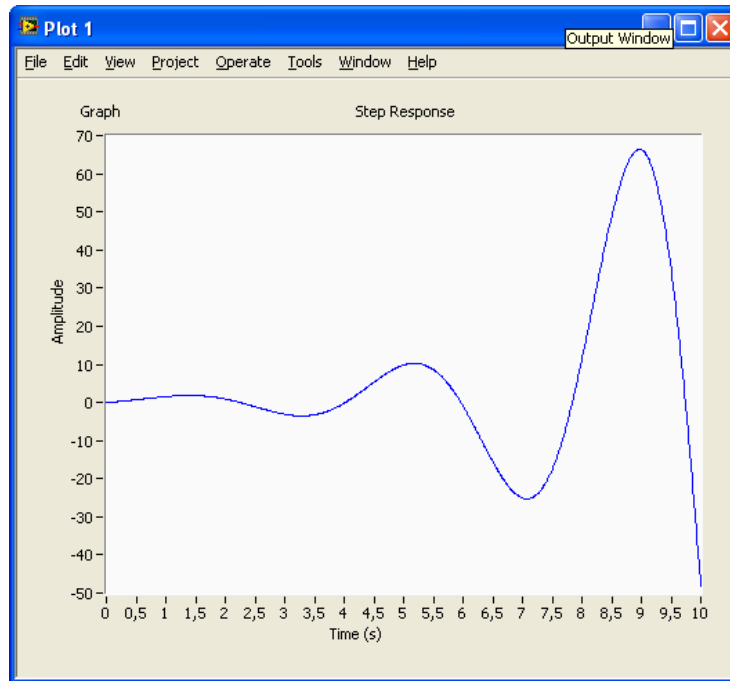
→ We see the poles are complex conjugate and that they lies in the right half-plane.

```
p =
    0.5 + 1.6583i
    0.5 - 1.6583i
```

We plot the step response for the transfer function using the `step` function:

```
num=[1,1];
den=[1,-1,3];
H=tf(num,den);
t=[0:0.01:10];
step(H,t);
```

This gives the following plot:



→ We see the system is unstable

[End of Example]

# Appendix A – MathScript Functions

## Basic Functions

Here are some descriptions for the most used basic MathScript functions.

Function	Description	Example
<b>help</b>	MathScript displays the help information available	>> <b>help</b>
<b>help</b> <b>&lt;function&gt;</b>	Display help about a specific function	>> <b>help</b> plot
<b>who, whos</b>	who lists in alphabetical order all variables in the currently active workspace.	>> <b>who</b> >> <b>whos</b>
<b>clear</b>	Clear variables and functions from memory.	>> <b>clear</b> >> <b>clear</b> x
<b>size</b>	Size of arrays, matrices	>>x=[1 2 ; 3 4]; >> <b>size</b> (A)
<b>length</b>	Length of a vector	>>x=[1:1:10]; >> <b>length</b> (x)
<b>format</b>	Set output format	
<b>disp</b>	Display text or array	>>A=[1 2;3 4]; >> <b>disp</b> (A)
<b>plot</b>	This function is used to create a plot	>>x=[1:1:10]; >> <b>plot</b> (x) >>y=sin(x); >> <b>plot</b> (x,y)
<b>clc</b>	Clear the Command window	>> <b>clc</b>
<b>rand</b>	Creates a random number, vector or matrix	>> <b>rand</b> >> <b>rand</b> (2,1)
<b>max</b>	Find the largest number in a vector	>>x=[1:1:10] >> <b>max</b> (x)
<b>min</b>	Find the smallest number in a vector	>>x=[1:1:10] >> <b>min</b> (x)
<b>mean</b>	Average or mean value	>>x=[1:1:10] >> <b>mean</b> (x)
<b>std</b>	Standard deviation	>>x=[1:1:10] >> <b>std</b> (x)

## Basic Plotting Functions

Function	Description	Example
<b>plot</b>	Generates a plot. plot(y) plots the columns of y against the indexes of the columns.	>>X = [0:0.01:1]; >Y = X.*X; > <b>plot</b> (X, Y)
<b>figure</b>	Create a new figure window	>> <b>figure</b> >> <b>figure</b> (1)
<b>subplot</b>	Create subplots in a Figure. subplot(m,n,p) or subplot(mnp), breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot. The axes are counted along the top row of the Figure window, then the second row, etc.	>> <b>subplot</b> (2,2,1)

<b>grid</b>	Creates grid lines in a plot. “grid on” adds major grid lines to the current plot. “grid off” removes major and minor grid lines from the current plot.	<pre>&gt;&gt;grid &gt;&gt;grid on &gt;&gt;grid off</pre>
<b>axis</b>	Control axis scaling and appearance. “axis([xmin xmax ymin ymax])” sets the limits for the x- and y-axis of the current axes.	<pre>&gt;&gt;axis([xmin xmax ymin ymax]) &gt;&gt;axis off &gt;&gt;axis on</pre>
<b>title</b>	Add title to current plot title('string')	<pre>&gt;&gt;title('this is a title')</pre>
<b>xlabel</b>	Add xlabel to current plot xlabel('string')	<pre>&gt;&gt; xlabel('time')</pre>
<b>ylabel</b>	Add ylabel to current plot ylabel('string')	<pre>&gt;&gt; ylabel('temperature')</pre>
<b>legend</b>	Creates a legend in the corner (or at a specified position) of the plot	<pre>&gt;&gt; legend('temperature')</pre>
<b>hold</b>	Freezes the current plot, so that additional plots can be overlaid	<pre>&gt;&gt;hold on &gt;&gt;hold off</pre>

For more information about the plots function, type “[help plots](#)”.

## Functions used for Control and Simulation

Function	Description	Example
<b>plot</b>	Generates a plot. plot(y) plots the columns of y against the indexes of the columns.	<pre>&gt;X = [0:0.01:1]; &gt;Y = X.*X; &gt;plot(X, Y)</pre>
<b>tf</b>	Creates system model in transfer function form. You also can use this function to state-space models to transfer function form.	<pre>&gt;num=[1]; &gt;den=[1, 1, 1]; &gt;H = <b>tf</b>(num, den)</pre>
<b>poles</b>	Returns the locations of the closed-loop poles of a system model.	<pre>&gt;num=[1] &gt;den=[1,1] &gt;H=<b>tf</b>(num,den) &gt;<b>poles</b>(H)</pre>
<b>tfinfo</b>	Returns information about a transfer function system model.	<pre>&gt;[num, den, delay, Ts] = <b>tfinfo</b>(SysInTF)</pre>
<b>step</b>	Creates a step response plot of the system model. You also can use this function to return the step response of the model outputs. If the model is in state-space form, you also can use this function to return the step response of the model states. This function assumes the initial model states are zero. If you do not specify an output, this function creates a plot.	<pre>&gt;num=[1,1]; &gt;den=[1,-1,3]; &gt;H=<b>tf</b>(num,den); &gt;t=[0:0.01:10]; &gt;<b>step</b>(H,t);</pre>
<b>lsim</b>	Creates the linear simulation plot of a system model. This function calculates the output of a system model when a set of inputs excite the model, using discrete simulation. If you do not specify an output, this function creates a plot.	<pre>&gt;t = [0:0.1:10] &gt;u = <b>sin</b>(0.1*pi*t)' &gt;<b>lsim</b>(SysIn, u, t)</pre>
<b>Sys_order1</b>	Constructs the components of a first-order system model based on a gain, time constant, and delay that you specify. You can use this function to create either a state-space model or a transfer function model, depending on the output parameters you specify.	<pre>&gt;K = 1; &gt;tau = 1; &gt;H = <b>sys_order1</b>(K, tau)</pre>
<b>Sys_order2</b>	Constructs the components of a second-order system model based on a damping ratio and natural frequency you specify. You can use this function to create either a state-space model or a transfer function model, depending on the output parameters you specify.	<pre>&gt;dr = 0.5 &gt;wn = 20 &gt;[num, den] = <b>sys_order2</b>(wn, dr) &gt;SysTF = <b>tf</b>(num, den) &gt;[A, B, C, D] = <b>sys_order2</b>(wn, dr) &gt;SysSS = <b>ss</b>(A, B, C, D)</pre>
<b>damp</b>	Returns the damping ratios and natural frequencies of the poles of a system model.	<pre>&gt;[dr, wn, p] = <b>damp</b>(SysIn)</pre>
<b>pid</b>	Constructs a proportional-integral-derivative (PID) controller model in either parallel, series, or academic form. Refer to the LabVIEW Control Design User Manual for information about these three forms.	<pre>&gt;Kc = 0.5; &gt;Ti = 0.25; &gt;SysOutTF = <b>pid</b>(Kc, Ti, 'academic');</pre>
<b>conv</b>	Computes the convolution of two vectors or matrices.	<pre>&gt;C1 = [1, 2, 3]; &gt;C2 = [3, 4]; &gt;C = <b>conv</b>(C1, C2)</pre>

<b>series</b>	Connects two system models in series to produce a model SysSer with input and output connections you specify	<code>&gt;Hseries = <b>series</b>(H1,H2)</code>
<b>feedback</b>	Connects two system models together to produce a closed-loop model using negative or positive feedback connections	<code>&gt;SysClosed = <b>feedback</b>(SysIn_1, SysIn_2)</code>
<b>ss</b>	Constructs a model in state-space form. You also can use this function to convert transfer function models to state-space form.	<code>&gt;A = eye(2) &gt;B = [0; 1] &gt;C = B' &gt;SysOutSS = <b>ss</b>(A, B, C)</code>
<b>ssinfo</b>	Returns information about a state-space system model.	<code>&gt;A = [1, 1; -1, 2] &gt;B = [1, 2]' &gt;C = [2, 1] &gt;D = 0 &gt;SysInSS = <b>ss</b>(A, B, C, D) &gt;[A, B, C, D, Ts] = <b>ssinfo</b>(SysInSS) &gt;[num, den] = <b>pade</b>(delay, order) &gt;[A, B, C, D] = <b>pade</b>(delay, order)</code>
<b>pade</b>	Incorporates time delays into a system model using the Pade approximation method, which converts all residuals. You must specify the delay using the set function. You also can use this function to calculate coefficients of numerator and denominator polynomial functions with a specified delay.	
<b>bode</b>	Creates the Bode magnitude and Bode phase plots of a system model. You also can use this function to return the magnitude and phase values of a model at frequencies you specify. If you do not specify an output, this function creates a plot.	<code>&gt;num=[4]; &gt;den=[2, 1]; &gt;H = <b>tf</b>(num, den) &gt;<b>bode</b>(H)</code>
<b>bodemag</b>	Creates the Bode magnitude plot of a system model. If you do not specify an output, this function creates a plot.	<code>&gt;[mag, wout] = <b>bodemag</b>(SysIn) &gt;[mag, wout] = <b>bodemag</b>(SysIn, [wmin wmax]) &gt;[mag, wout] = <b>bodemag</b>(SysIn, wlist)</code>
<b>margin</b>	Calculates and/or plots the smallest gain and phase margins of a single-input single-output (SISO) system model. The gain margin indicates where the frequency response crosses at 0 decibels. The phase margin indicates where the frequency response crosses -180 degrees. Use the margins function to return all gain and phase margins of a SISO model.	<code>&gt;num = [1] &gt;den = [1, 5, 6] &gt;H = <b>tf</b>(num, den) <b>margin</b>(H)</code>
<b>margins</b>	Calculates all gain and phase margins of a single-input single-output (SISO) system model. The gain margins indicate where the frequency response crosses at 0 decibels. The phase margins indicate where the frequency response crosses -180 degrees. Use the margin function to return only the smallest gain and phase margins of a SISO model.	<code>&gt;[gmf, gm, pmf, pm] = <b>margins</b>(H)</code>

For more details about these functions, type “[help cdt](#)” to get an overview of all the functions used for Control Design and Simulation. For detailed help about one specific function, type “[help <function\\_name>](#)”.



Hans-Petter Halvorsen, M.Sc.

E-mail: [hans.p.halvorsen@hit.no](mailto:hans.p.halvorsen@hit.no)

Blog: <http://home.hit.no/~hansha/>



---

University College of Southeast Norway

[www.usn.no](http://www.usn.no)

---