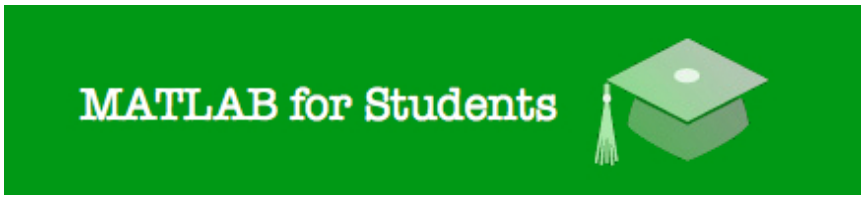
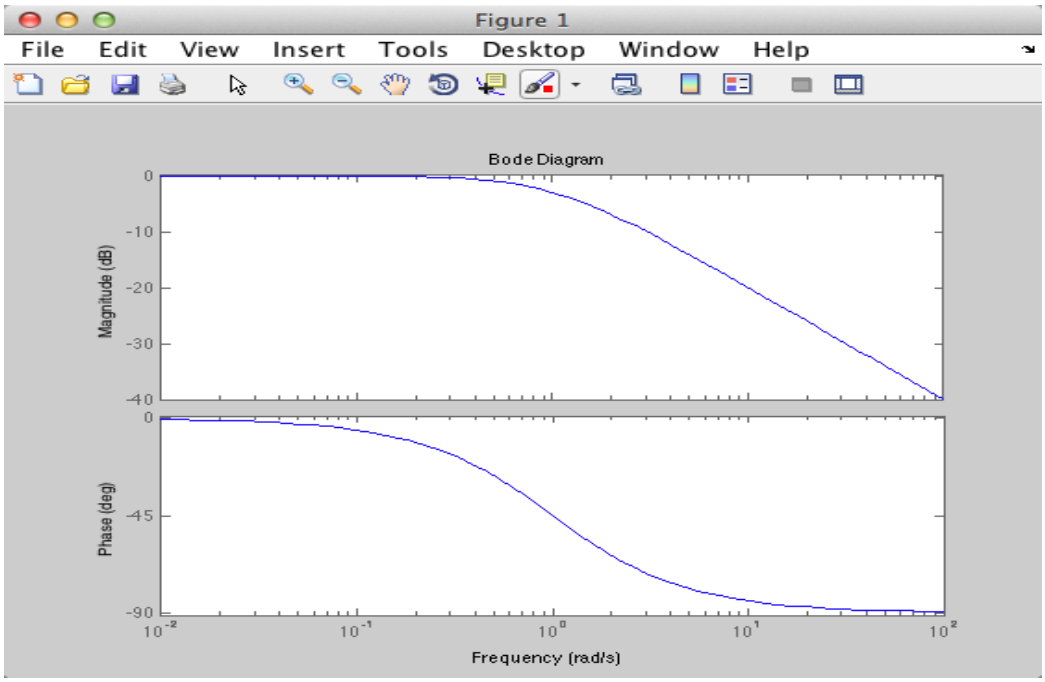
University College of Southeast Norway



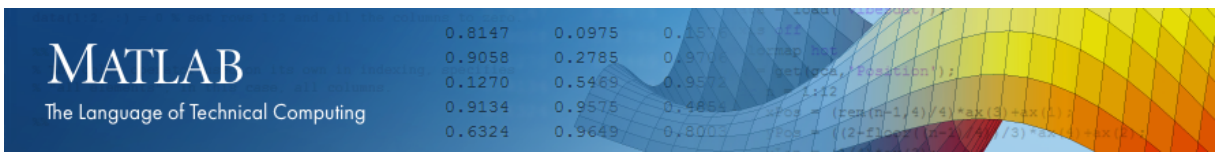
MATLAB

Part II: Modelling, Simulation & Control

Hans-Petter Halvorsen, 2016.10.04



http://home.hit.no/~hansha



Preface

In this MATLAB Course you will learn basic MATLAB and how to use MATLAB in Control and Simulation applications. An introduction to Simulink and other Tools will also be given.

MATLAB is a tool for technical computing, computation and visualization in an integrated environment. MATLAB is an abbreviation for MATrix LABoratory, so it is well suited for matrix manipulation and problem solving related to Linear Algebra, Modelling, Simulation and Control applications.

This is a self-paced course based on this document and some short videos on the way. This document contains lots of examples and self-paced tasks that the users will go through and solve on their own. The user may go through the tasks in this document in their own pace and the instructor will be available for guidance throughout the course.

The MATLAB Course consists of 3 parts:

- MATLAB Course – Part I: Introduction to MATLAB
- MATLAB Course – Part II: Modelling, Simulation and Control
- MATLAB Course – Part III: Simulink and Advanced Topics

In Part II of the course (Part II: Modelling, Control and Simulation) you will learn how to use MATLAB in Modelling, Control and Simulation.

You must go through MATLAB Course – Part I: Introduction to MATLAB before you start.

The course consists of lots of Tasks you should solve while reading this course manual and watching the videos referred to in the text.


Make sure to bring your **headphones** for the videos in this course. The course consists of several short videos that will give you an introduction to the different topics in the course.

Prerequisites

You should be familiar with undergraduate-level mathematics and have experience with basic computer operations.

What is MATLAB**?** MATLAB is a tool for technical computing, computation and visualization in an integrated environment. MATLAB is an abbreviation for MATrix LABoratory, so it is well suited for matrix manipulation and problem solving related to Linear Algebra.

MATLAB is developed by The MathWorks. MATLAB is a short-term for MATrix LABoratory. MATLAB is in use world-wide by researchers and universities. For more information, see www.mathworks.com

For more information about MATLAB, etc., please visit http://home.hit.no/~hansha/

Online MATLAB Resources:

MATLAB Basics:

http://home.hit.no/~hansha/video/matlab_basics.php

Modelling, Simulation and Control with MATLAB:

http://home.hit.no/~hansha/video/matlab_mic.php

MATLAB Training:

http://home.hit.no/~hansha/documents/lab/Lab%20Work/matlabtraining.htm

MATLAB for Students:

http://home.hit.no/~hansha/documents/lab/Lab%20Work/matlab.htm

On these web pages you find video solutions, complete step by step solutions, downloadable MATLAB code, additional resources, etc.

Table of Contents

1 Introduction

Part 2: “Modelling, Simulation and Control” consists of the following topics:

- Differential Equations and ODE Solvers
- Discrete Systems
- Numerical Techniques
    - Interpolation
    - Curve Fitting
    - Numerical Differentiation
    - Numerical Integration
- Optimization
- Control System Toolbox
- Transfer functions
- State-space models
- Frequency Response

2 Differential Equations and ODE Solvers

MATLAB have lots of built-in functionality for solving differential equations. MATLAB includes functions that solve ordinary differential equations (ODE) of the form:

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_0$$

MATLAB can solve these equations numerically.

Higher order differential equations must be reformulated into a system of first order differential equations.

Note! Different notation is used:

$$\frac{dy}{dt} = y' = \dot{y}$$

This document will use these different notations interchangeably.

Not all differential equations can be solved by the same technique, so MATLAB offers lots of different ODE solvers for solving differential equations, such as **ode45**, **ode23**, **ode113**, etc.

Example:

Given the following differential equation:

$$\dot{x} = ax$$

where $a = -\frac{1}{T}$,where T is the time constant

Note! $\dot{x} = \frac{dx}{dt}$

The solution for the differential equation is found to be:

$$x(t) = e^{at}x_0$$

We shall plot the solution for this differential equation using MATLAB.

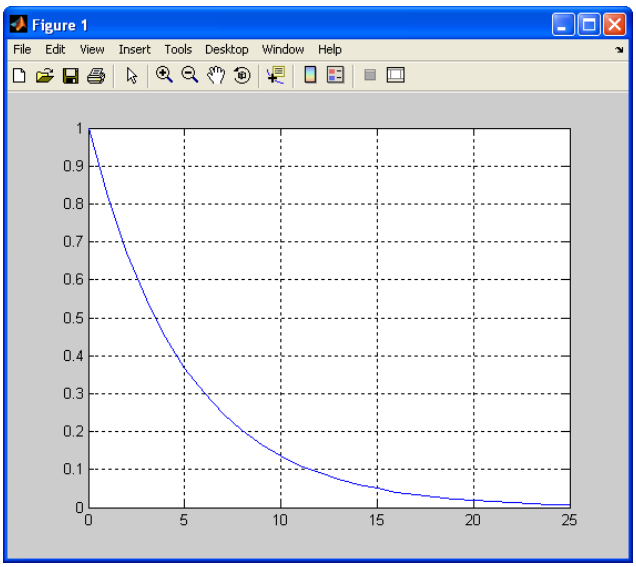
Set $T = 5$ and the initial condition $x(0) = 1$.

We will create a script in MATLAB (.m file) where we plot the solution $x(t)$ in the time interval $0 \leq t \leq 25$

The Code is as follows:

```
T = 5;
a = -1/T;
x0 = 1;
t = [0:1:25]

x = exp(a*t)*x0;

plot(t,x);
grid
```

This gives the following Results:



[End of Example]

This works fine, but the problem is that we first have to find the solution to the differential equation – instead we can use one of the built-in solvers for Ordinary Differential Equations (ODE) in MATLAB.

There are different functions, such as **ode23** and **ode45**.

Example:

We use the ode23 solver in MATLAB for solving the differential equation ("runmydiff.m"):

```
tspan = [0 25];
x0 = 1;
```

```
[t,x] = ode23(@mydiff,tspan,x0);
plot(t,x)
```

Where `@mydiff` is defined as a function like this ("mydiff.m"):

```
function dx = mydiff(t,x)
a = -1/5;
dx = a*x;
```

This gives the same results as shown in the plot above and MATLAB have solved the differential equation for us (numerically).

Note! You have to implement it in 2 different m. files, one m. file where you define the differential equation you are solving, and another .m file where you solve the equation using the ode23 solver.

[End of Example]

2.1 ODE Solvers in MATLAB

All of the ODE solver functions share a syntax that makes it easy to try any of the different numerical methods, if it is not apparent which is the most appropriate. To apply a different method to the same problem, simply change the ODE solver function name. The simplest syntax, common to all the solver functions, is:

```
[t,y] = solver(odefun,tspan,y0,options,…)
```

where "solver" is one of the ODE solver functions (ode23, ode45, etc.).

Note! If you don't specify the resulting array [t, y], the function create a plot of the result.

'odefun' is the function handler, which is a "nickname" for your function that contains the differential equations.

Example:

Given the differential equations:

$$\frac{dy}{dt} = x$$

$$\frac{dx}{dt} = -y$$

In MATLAB you define a function for these differential equations:

```
function  dy=mydiff(t,y)
```

```
dy(1) = y(2);
dy(2) = -y(1);

dy = [dy(1); dy(2)];
```

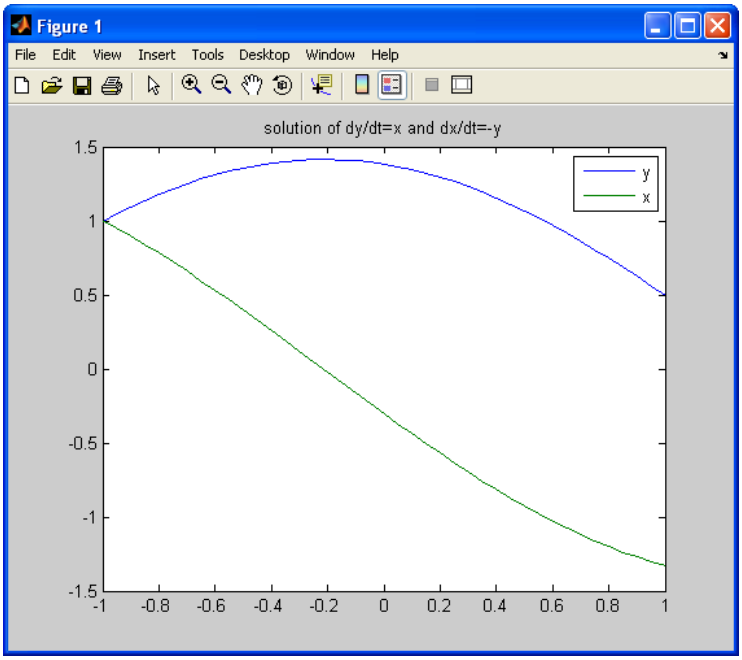
Note! Since numbers of equations is more than one, we need to use vectors!!

Using the ode45 function gives the following code:

```
[t,y] = ode45(@mydiff, [-1,1], [1,1]);

plot(t,y)
title('solution of dy/dt=x and dx/dt=-y')
legend('y', 'x')
```

The equations are solved in the time span $[-1\ 1]$ with initial values $[1, 1]$.

This gives the following plot:



To make it more clearly, we can rewrite the equations (setting $x_1 = x, x_2 = y$):

$$\frac{dx_1}{dt} = -x_2$$

$$\frac{dx_2}{dt} = x_1$$

The code for the differential equations ("mydiff.m"):

```
function dxdt = mydiff(t,x)

dxdt(1) = -x(2);
dxdt(2) = x(1);
```

```
dxdt = dxdt';
```

Note! The function mydiff must return a column vector, that's why we need to transpose it.

Then we use the ode solver to solve the differential equations ("run_mydiff.m"):

```
tspan = [-1,1];
x0 = [1,1];

[t,x] = ode45(@mydiff, tspan, x0);

plot (t,x)

legend('x1', 'x2')
```

The solution will be the same.

[End of Example]

Task 1: Bacteria Population

In this task we will simulate a simple model of a bacteria population in a jar.

The model is as follows:

$$birth\ rate = bx$$

$$death\ rate = px^2$$

Then the total rate of change of bacteria population is:

$$\boxed{\dot{x} = bx - px^2}$$

Set ***b=1***/hour and ***p=0.5*** bacteria-hour

Note! $\dot{x} = \frac{dx}{dt}$

→ Simulate (i.e., create a plot) the number of bacteria in the jar after **1 hour**, assuming that initially there are **100 bacteria** present.

How many bacteria are present after 1 hour?

[End of Task]

Task 2: Passing Parameters to the model

Given the following system:

$$\dot{x} = ax + b$$

where $a = -\frac{1}{T}$,where T is the time constant

In this case we want to pass a and b as parameters, to make it easy to be able to change values for these parameters.

We set initial condition $x(0) = 1$ and $T = 5$.

The function for the differential equation is:

```
function dx = mysimplediff(t,x,param)
% My Simple Differential Equation

a = param(1);
b = param(2);

dx = a*x+b;
```

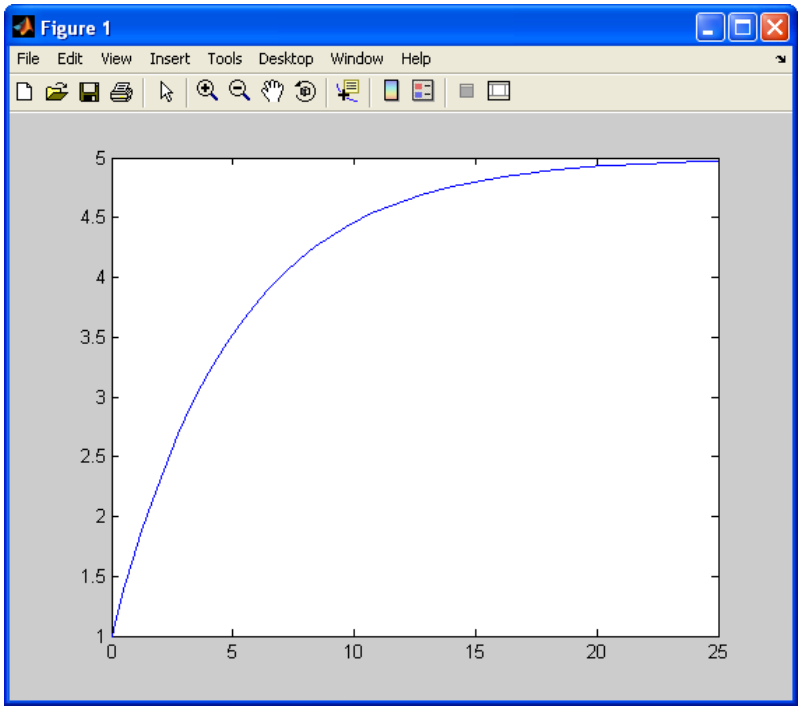
Then we solve and plot the equation using this code:

```
tspan = [0 25];
x0 = 1;
a = -1/5;
b = 1;
param = [a b];

[t,y] = ode45(@mysimplediff, tspan, x0,[], param);
plot(t,y)
```

By doing this, it is very easy to changes values for the parameters a and b without changing the code for the differential equation.

Note! We need to use the 5.argument in the ODE solver function for this. The 4.argument is for special options and is normally set to "[]", i.e., no options.

The result from the simulation is:



→ Write the code above

Read more about the different solvers that exists in the Help system in MATLAB

[End of Task]

Task 3: ODE Solvers

Use the ode23 function to solve and plot the results of the following differential equation in the interval $[t_0, t_f]$:

$$\boldsymbol{w' + (1.2 + sin10t)w = 0},\ t_0 = 0, t_f = 5, w(t_0) = 1$$

Note! $w' = \frac{dw}{dt}$

[End of Task]

Task 4: 2. order differential equation

Use the ode23/ode45 function to solve and plot the results of the following differential equation in the interval $[t_0, t_f]$:

$$\boldsymbol{(1 + t^2)\ddot{w} + 2t\dot{w} + 3w = 2},\ t_0 = 0, t_f = 5, w(t_0) = 0, \dot{w}(t_0) = 1$$

Note! $\ddot{w} = \frac{d^2w}{dt^2}$

Note! Higher order differential equations must be reformulated into a system of first order differential equations.

Tip 1: Reformulate the differential equation so $\ddot{w}$ is alone on the left side.

Tip 2: Set:

$$w = x_1$$

$$\dot{w} = x_2$$

[End of Task]

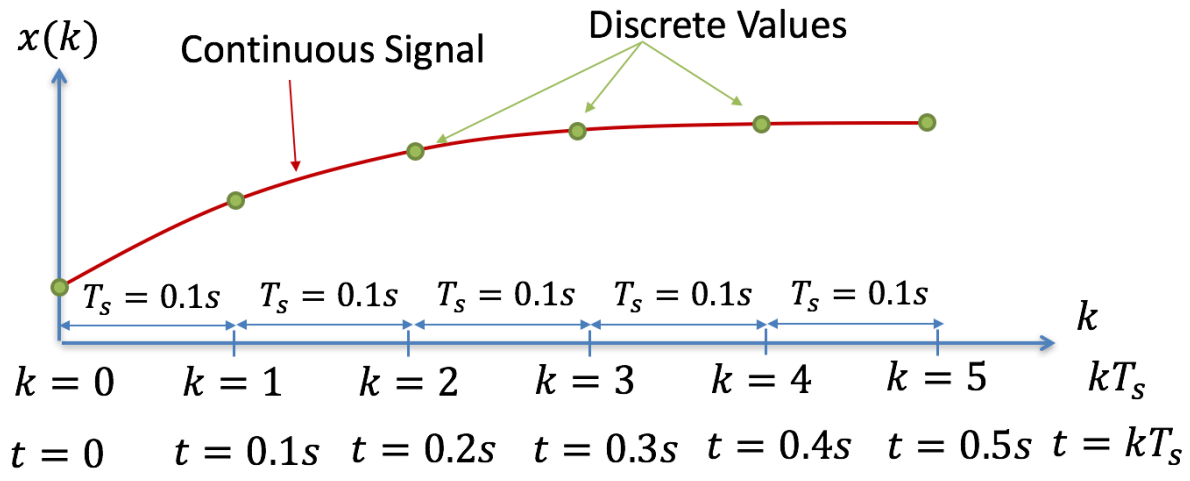
3 Discrete Systems

MATLAB has built-in powerful features for simulation of continuous differential equations and dynamic systems.

Sometimes we want to or need to discretize a continuous system and then simulate it in MATLAB.

When dealing with computer simulation, we need to create a discrete version of our system. This means we need to make a discrete version of our continuous differential equations. Actually, the built-in ODE solvers in MATLAB use different discretization methods. Interpolation, Curve Fitting, etc. is also based on a set of discrete values (data points or measurements). The same with Numerical Differentiation and Numerical Integration, etc.

Below we see a continuous signal vs the discrete signal for a given system with discrete time interval $T_s = 0.1s$.



3.1 Discretization

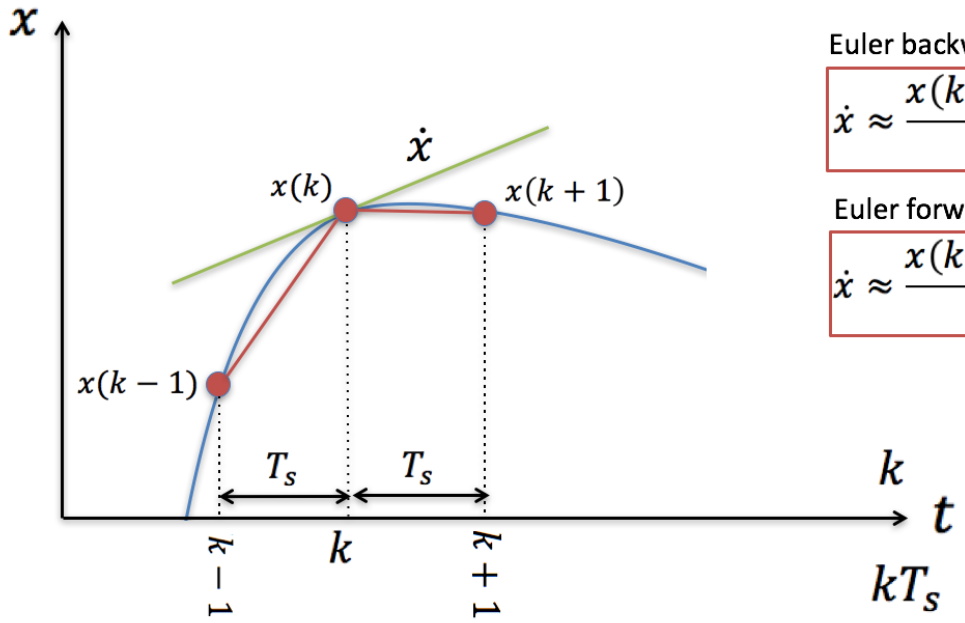
In order to discretize a continuous model there are lots of different methods to use. One of the simplest is Euler Forward method:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s}$$

where T_s is the sampling time.

Lots of other discretization methods do exists, such as “Euler backward”, Zero Order Hold (ZOH), Tustin’s method, etc.

As shown in a previous chapter, MATLAB have lots of built-in functions for solving differential equations numerically, but here we will create our own discrete model.



Euler backward method:

$$\dot{x} \approx \frac{x(k) - x(k-1)}{T_s}$$

Euler forward method:

$$\dot{x} \approx \frac{x(k+1) - x(k)}{T_s}$$

Example:

Given the following differential equation:

$$\dot{x} = -ax + bu$$

Note!

$\dot{x}$ is the same as $\frac{dx}{dt}$

Where:

x - Process variable, e.g., Level, Pressure, Temperature, etc.

u - Input variable, e.g., Control Signal from the Controller

$a,\ b$ - Constants

We start with finding the discrete differential equation.

We can use e.g., the **Euler Approximation**:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s}$$

T_s - Sampling Interval

Then we get:

$$\frac{x_{k+1} - x_k}{T_s} = -ax_k + bu_k$$

This gives the following discrete differential equation:

$$x_{k+1} = (1 - T_s a)x_k + T_s b u_k$$

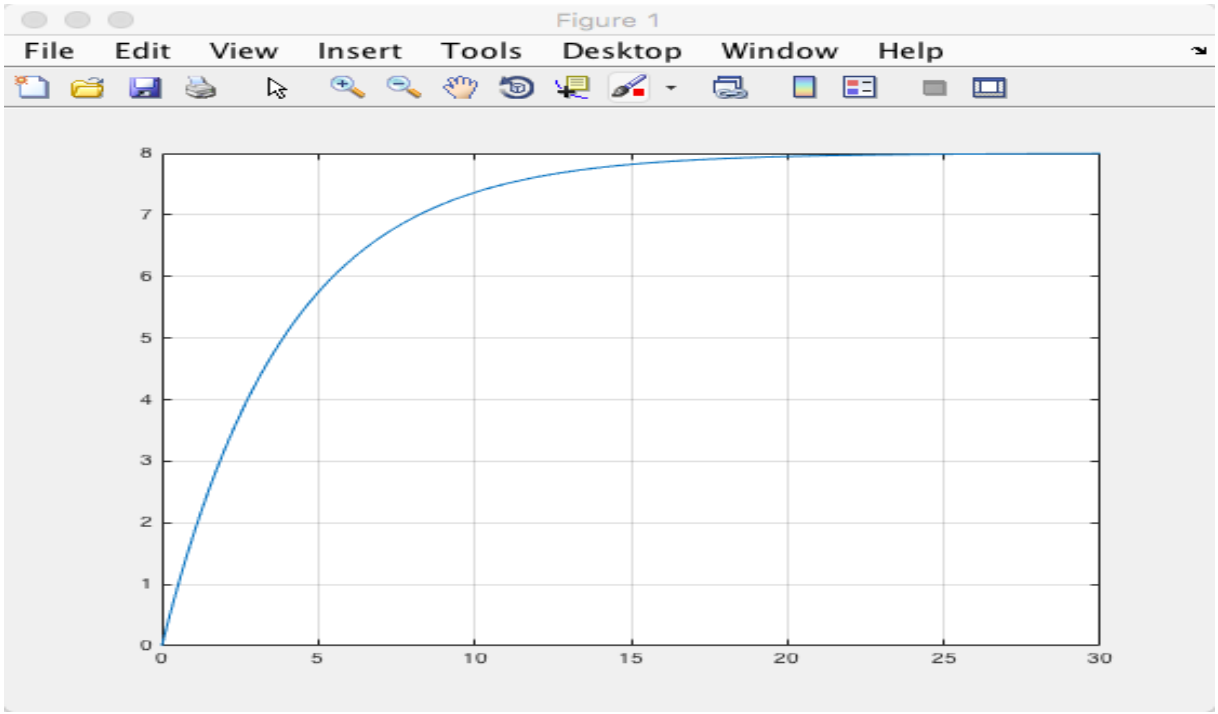
Now we are ready to simulate the system

We set $a = 0.25$, $b = 2$ and $u = 1$ (You can explore with other values on your own)

The Code can be written as follows:

```
% Simulation of discrete model
clear, clc

% Model Parameters
a = 0.25;b = 2;

% Simulation Parameters
Ts = 0.1; %s
Tstop = 30; %s
uk = 1; % Step Response
x(1) = 0;

% Simulation
for k=1:(Tstop/Ts)
   x(k+1) = (1-a*Ts).*x(k) + Ts*b*uk;
end

% Plot the Simulation Results
k=0:Ts:Tstop;
plot(k,x)
grid on
```

This gives the following Results:



[End of Example]

Task 5: Discrete Simulation

Given the following differential equation:

$$\dot{x} = ax$$

where $a = -\frac{1}{T}$, where T is the time constant

Note! $\dot{x} = \frac{dx}{dt}$

Find the discrete differential equation and plot the solution for this system using MATLAB.

Set $T = 5$ and the initial condition $x(0) = 1$.

Create a script in MATLAB (.m file) where we plot the solution $x(k)$.

[End of Task]

Task 6: Discrete Simulation – Bacteria Population

In this task we will simulate a simple model of a bacteria population in a jar.

The model is as follows:

$$birth\ rate = bx$$

$$death\ rate = px^2$$

Then the total rate of change of bacteria population is:

$$\boxed{\dot{x} = bx - px^2}$$

Set ***b=1***/hour and ***p=0.5*** bacteria-hour

We will simulate the number of bacteria in the jar after **1 hour**, assuming that initially there are **100 bacteria** present.

→ Find the discrete model using the Euler Forward method by hand and implement and simulate the system in MATLAB using a For Loop.

[End of Task]

Task 7: Simulation with 2 variables

Given the following system

$$\frac{dx_1}{dt} = -x_2$$

$$\frac{dx_2}{dt} = x_1$$

Find the discrete system and simulate the discrete system in MATLAB.

Solve the equations, e.g., in the time span $[-1\ 1]$ with initial values $[1,\ 1]$.

[End of Task]

4 Numerical Techniques

In the previous chapter we investigated how to solve differential equations numerically, in this chapter we will take a closer look at some other numerical techniques offered by MATLAB, such as interpolation, curve-fitting, numerical differentiations and integrations.

4.1 Interpolation

Interpolation is used to estimate data points between two known points. The most common interpolation technique is Linear Interpolation.

In MATLAB we can use the **interp1** function.

Example:

Given the following data:

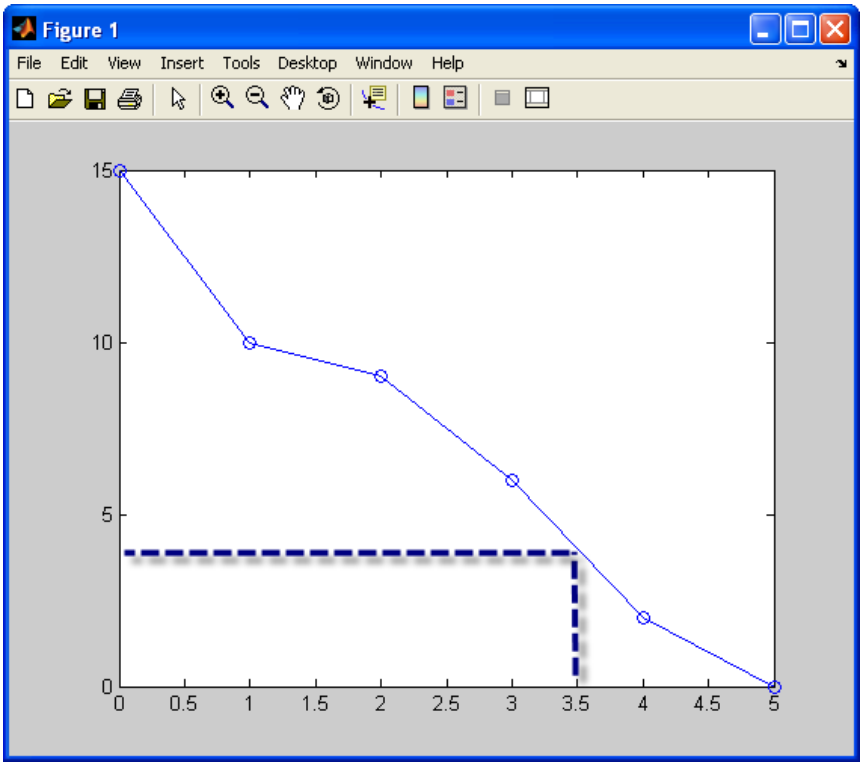
x	y
0	15
1	10
2	9
3	6
4	2
5	0

We will find the interpolated value for $x = 3.5$.

The following MATLAB code will do this:

```
x=0:5;
y=[15, 10, 9, 6, 2, 0];

plot(x,y ,'-o')

% Find interpolated value for x=3.5
new_x=3.5;
new_y = interp1(x,y,new_x)
```

The answer is 4, from the plot below we see this is a good guess:



[End of Example]

The default is linear interpolation, but there are other types available, such as:

- linear
- nearest
- spline
- cubic
- etc.

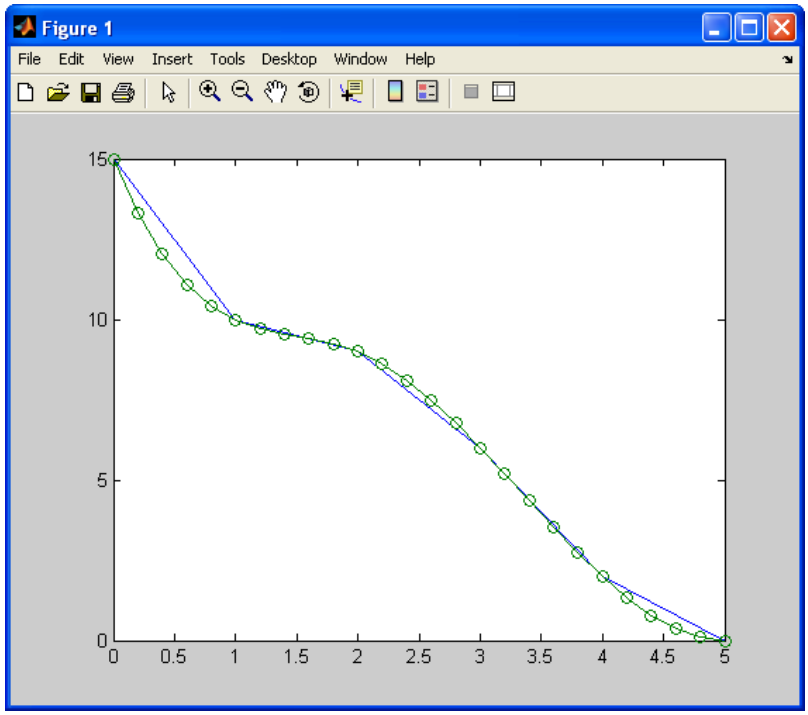
Type “help interp1” in order to read more about the different options.

Example:

In this example we will use a spline interpolation on the same data as in the example above.

```
x=0:5;
y=[15, 10, 9,6, 2, 0];

new_x=0:0.2:5;
new_y=interp1(x,y,new_x, 'spline')

plot(x,y, new_x, new_y, '-o')
```

The result is as we plot both the original point and the interpolated points in the same graph:



We see this result in 2 different lines.

[End of Example]

Task 8: Interpolation

Given the following data:

Temperature, T [°C]	Energy, u [KJ/kg]
100	2506.7
150	2582.8
200	2658.1
250	2733.7
300	2810.4
400	2967.9
500	3131.6

Plot u versus T. Find the interpolated data and plot it in the same graph. Test out different interpolation types. Discuss the results. What kind of interpolation is best in this case?

What is the interpolated value for u=2680.78 KJ/kg?

[End of Task]

4.2 Curve Fitting

In the previous section we found interpolated points, i.e., we found values between the measured points using the interpolation technique. It would be more convenient to model the data as mathematical function $y = f(x)$. Then we could easily calculate any data we want based on this model.

MATLAB has built-in curve fitting functions that allows us to create empiric data model. It is important to have in mind that these models are good only in the region we have collected data.

Here are some of the functions available in MATLAB used for curve fitting:

Function	Description	Example
polyfit	P = POLYFIT(X,Y,N) finds the coefficients of a polynomial P(X) of degree N that fits the data Y best in a least-squares sense. P is a row vector of length N+1 containing the polynomial coefficients in descending powers, P(1)*X^N + P(2)*X^(N-1) +...+ P(N)*X + P(N+1).	`>>polyfit(x,y,1)`
polyval	Evaluate polynomial. Y = POLYVAL(P,X) returns the value of a polynomial P evaluated at X. P is a vector of length N+1 whose elements are the coefficients of the polynomial in descending powers. Y = P(1)*X^N + P(2)*X^(N-1) + ... + P(N)*X + P(N+1)	

These techniques use a polynomial of degree N that fits the data Y best in a least-squares sense.

A polynomial is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

where $p_1, p_2, p_3, \ldots$ are the coefficients of the polynomial.

MATLAB represents polynomials as row arrays containing coefficients ordered by descending powers.

4.2.1 Linear Regression

Here we will create a linear model of our data on the form:

$$y = ax + b$$

This is actually a polynomial of 1.order

Example:

Given the following data:

x	y
0	15
1	10
2	9
3	6
4	2
5	0

We will find the model on the form:

$$y = ax + b$$

We will use the **polyfit** function in MATLAB.

The following code will solve it:

```
x=[0, 1, 2, 3, 4 ,5];
y=[15, 10, 9, 6, 2 ,0];
n=1; % 1.order polynomial
p = polyfit(x,y,n)
```

The answer is:

ans =

    -2.9143   14.2857

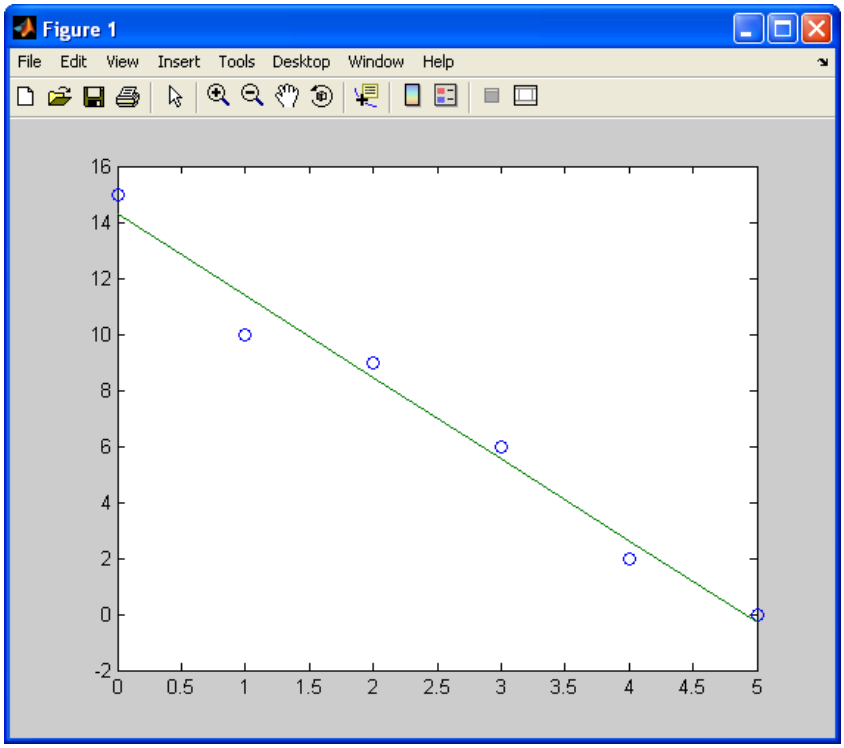
This gives the following model:

$$y = -2.9143x + 14.2857$$

We can also plot the measured data and the model in the same plot:

```
x=[0, 1, 2, 3, 4 ,5];
y=[15, 10, 9, 6, 2 ,0];
n=1; % 1.order polynomial
p=polyfit(x,y,n);

a=p(1);
b=p(2);

ymodel=a*x+b;

plot(x,y,'o',x,ymodel)
```

This gives the following plot:



We see this gives a good model based on the data available.

[End of Example]

Task 9: Linear Regression

Given the following data:

Temperature, T [°C]	Energy, u [KJ/kg]
100	2506.7
150	2582.8
200	2658.1
250	2733.7
300	2810.4
400	2967.9
500	3131.6

Plot u versus T.

Find the linear regression model from the data

$$y = ax + b$$

Plot it in the same graph.

[End of Task]

4.2.2 Polynomial Regression

In the previous section we used linear regression which is a 1.order polynomial. In this section we will study higher order polynomials.

In polynomial regression we will find the following model:

$$y(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$$

Example:

Given the following data:

x	y
0	15
1	10
2	9
3	6
4	2
5	0

We will found the model of the form:

$$y(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$$

We will use the **polyfit** and **polyval** functions in MATLAB and compare the models using different orders of the polynomial.

We will investigate models of 2.order, 3.order, 4.order and 5.order. We have only 6 data points, so a model with order higher than 5 will make no sense.

We use a For loop in order to create models of 2, 3, 4 and 5.order.

The code is as follows:

```
x=[0, 1, 2, 3, 4 ,5];
y=[15, 10, 9, 6, 2 ,0];

for n=2:5 %From order 2 to 5
    p=polyfit(x,y,n)

    ymodel=polyval(p,x);

    subplot(2,2,n-1)
    plot(x,y,'o',x,ymodel)
    title(sprintf('Model of order %d', n));
end
```

The polyfit gives the following polynomials:

```
p =
    0.0536   -3.1821   14.4643
p =
   -0.0648    0.5397   -4.0701   14.6587
p =
    0.1875   -1.9398    6.2986   -9.4272   14.9802
p =
   -0.0417    0.7083   -4.2083   10.2917  -11.7500   15.0000
```

Using the values, we get the following models:

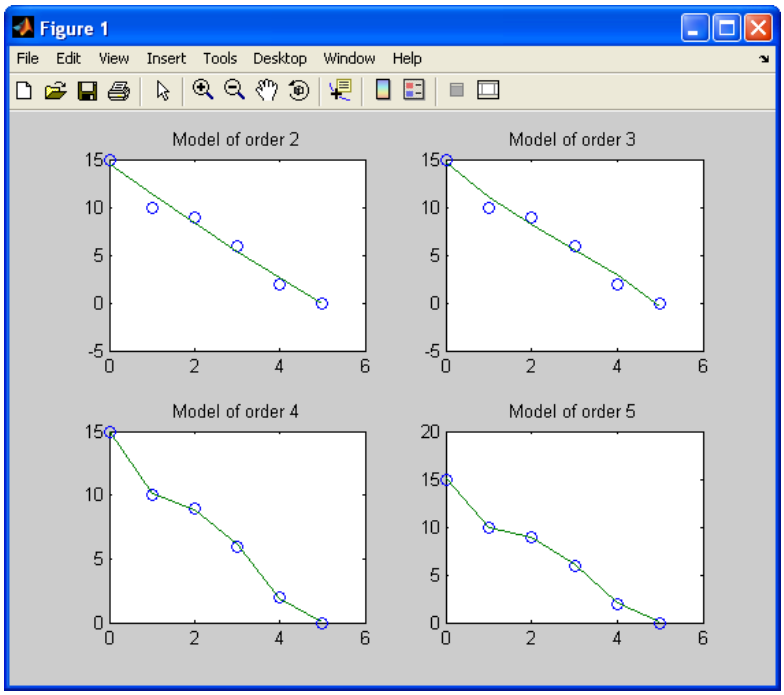
$$y_2(x) \approx 0.05x^2 - 3.2x + 14.5$$

$$y_3(x) \approx -0.065x^3 + 0.5x^2 - 4x + 14.7$$

$$y_4(x) \approx 0.2x^4 - 1.9x^3 + 6.3x^2 - 9.4x + 15$$

$$y_5(x) \approx -0.04x^5 + 0.7x^4 - 4.2x^3 + 10.3x^2 - 11.8x + 15$$

This gives the following results:



As expected, the higher order models match the data better and better.

Note! The fifth order model matches exactly because there were only six data points available.

[End of Example]

Task 10: Polynomial Regression

Given the following data:

x	y
10	23
20	45
30	60
40	82
50	111
60	140
70	167
80	198
90	200
100	220

→ Use the **polyfit** and **polyval** functions in MATLAB and compare the models using different orders of the polynomial.

Use subplots and make sure to add titles, etc.

[End of Task]

Task 11: Model fitting

Given the following data:

Height, h[ft]	Flow, f[ft^3/s]
0	0
1.7	2.6
1.95	3.6
2.60	4.03
2.92	6.45
4.04	11.22
5.24	30.61

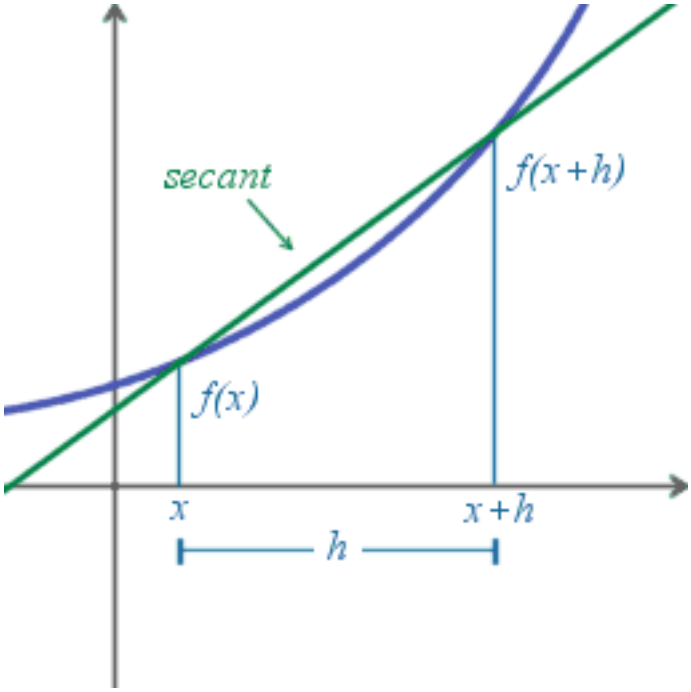
→ Create a 1. (linear), 2. (quadratic) and 3.order (cubic) model. Which gives the best model? Plot the result in the same plot and compare them. Add xlabel, ylabel, title and a legend to the plot and use different line styles so the user can easily see the difference.

[End of Task]

4.3 Numerical Differentiation

The derivative of a function $y = f(x)$ is a measure of how y changes with x.

Assume the following:



Then we have the following definition:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

MATLAB is a numerical language and do not perform symbolic mathematics (... well, that is not entirely true because there is Symbolic Toolbox available for MATLAB, but this Toolkit will not be used in this course).

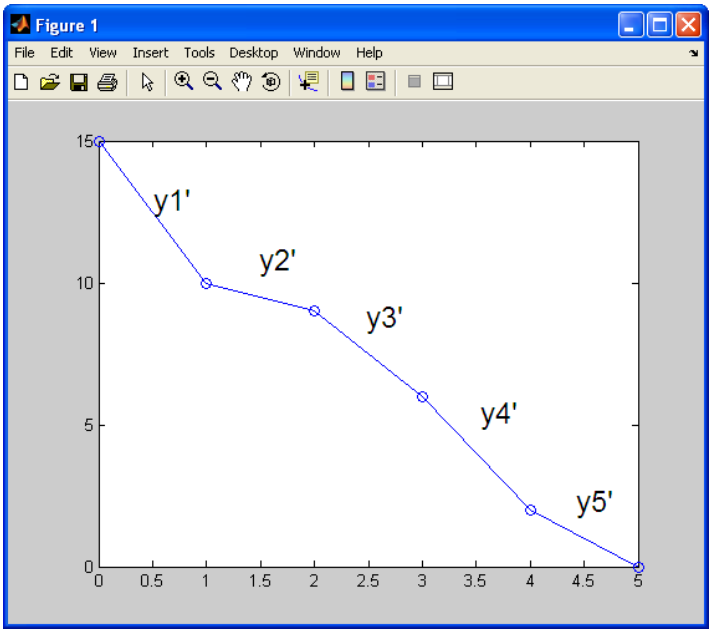
MATLAB offers functions for numerical differentiation, e.g.:

Function	Description	Example
diff	Difference and approximate derivative. DIFF(X), for a vector X, is [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)].	`>> dydx_num=diff(y)./diff(x);`
polyder	Differentiate polynomial. POLYDER(P) returns the derivative of the polynomial whose coefficients are the elements of vector P. POLYDER(A,B) returns the derivative of polynomial A*B.	`>>p=[1,2,3];` `>>polyder(p)`

A numerical approach to the derivative of a function $y = f(x)$ is:

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

This approximation of the derivative corresponds to the slope of each line segment used to connect each data point that exists. An example is shown below:



Example:

We will use Numerical Differentiation to find $\frac{dy}{dx}$ on the following function:

$$y = x^2$$

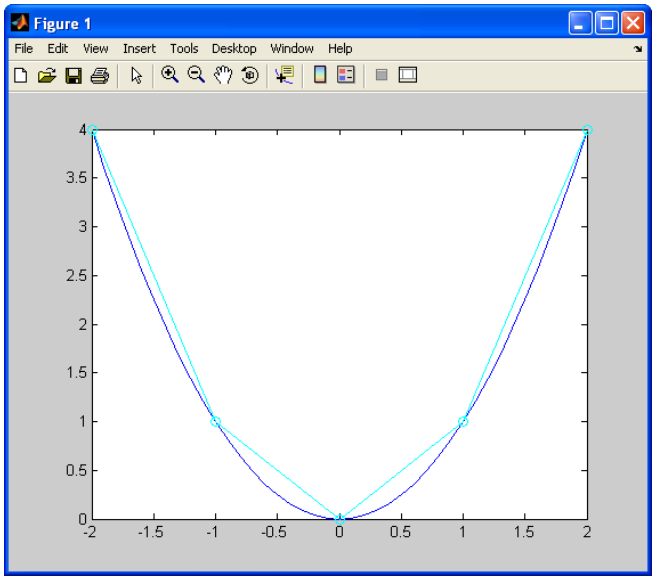
based on the data points (`x=-2:2`):

x	y
-2	4
-1	1
0	0
1	1
2	4

First, we will plot the data points together with the real function $y = x^2$ using the following code:

```
x=-2:0.1:2;
y=x.^2;
plot(x,y)

hold on

x=-2:2;
y=x.^2;
plot(x,y, '-oc')
```

This gives the following plot:



Then we want to find the derivative $\frac{dy}{dx}$

We know that the exact solution is:

$$\frac{dy}{dx} = 2x$$

For the values given in the table we have:

$$\frac{dy}{dx}(x = -2) = -4$$

$$\frac{dy}{dx}(x = -1) = -2$$

$$\frac{dy}{dx}(x = 0) = 0$$

$$\frac{dy}{dx}(x = 1) = 2$$

$$\frac{dy}{dx}(x = 2) = 4$$

We will use this to compare the results from the numerical differentiation with the exact solution (see above).

The code is as follows:

```
x=-2:2;
y=x.^2;
```

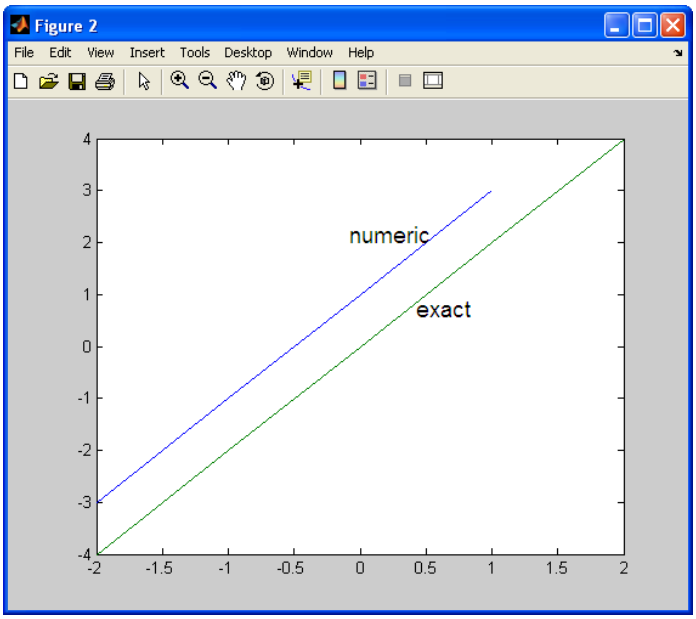
```
dydx_num=diff(y)./diff(x);
dydx_exact=2*x;
dydx=[[dydx_num, NaN]', dydx_exact']
```

This gives the following results (left column is from the numerical derivation, while the right column is from the exact derivation):

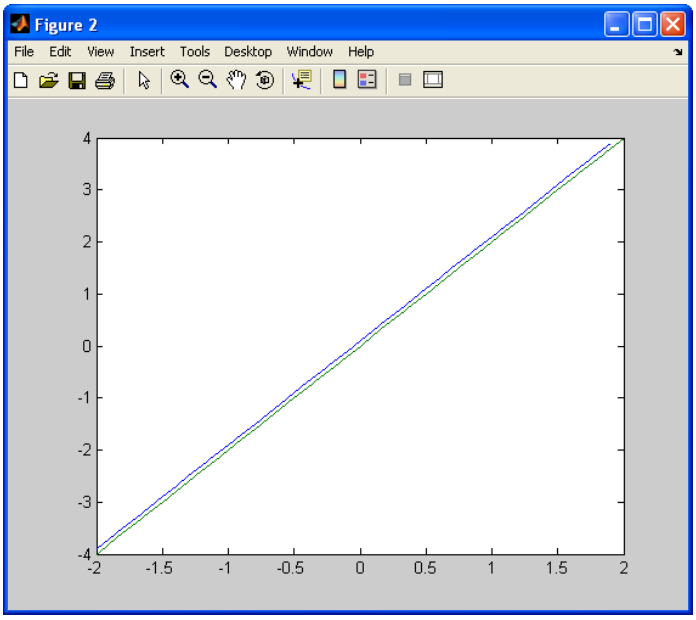
```
dydx =
    -3    -4
    -1    -2
     1     0
     3     2
   NaN     4
```

Note! NaN is added to the vector with numerical differentiation in order to get the same length of the vectors.

If we plot the derivatives (numerical and exact), we get:



If we increase the number of data points (`x=-2:0.1:2`) we get a better result:



[End of Example]

Task 12: Numerical Differentiation

Given the following equation:

$$y = x^3 + 2x^2 - x + 3$$

Find $\frac{dy}{dx}$ analytically (use "pen and paper").

Define a vector x from -5 to +5 and use the **diff** function to approximate the derivative y with respect to x ($\frac{\Delta y}{\Delta x}$).

Compare the data in a 2D array and/or plot both the exact value of $\frac{dy}{dx}$ and the approximation in the same plot.

Increase number of data point to see if there are any difference.

Do the same for the following functions:

$$y = \sin(x)$$

$$y = x^5 - 1$$

[End of Task]

4.3.1 Differentiation on Polynomials

A polynomial is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

where $p_1, p_2, p_3, \ldots$ are the coefficients of the polynomial.

The differentiation of the Polynomial will be:

$$p(x)' = p_1 n x^{n-1} + p_2 (n-1) x^{n-2} + \cdots + p_n$$

Example

Given the polynomial

$$p(x) = 2 + x^3$$

We can rewrite the polynomial like this:

$$p(x) = 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x + 2$$

The polynomial is defined in MATLAB as:

```
>> p=[1, 0, 0, 2]
```

We know that: $p' = 3x^2$

The code is as follows

```
>> p=[1, 0, 0, 2]
p =
    1    0    0    2
>> polyder(p)
ans =
    3    0    0
```

Which is correct, because

$$p(x)' = 3 \cdot x^2 + 0 \cdot x^2 + 0$$

with the coefficients:

$$p_1 = 3, p_2 = 0, p_3 = 0$$

And this is written as a vector $[3 \quad 0 \quad 0]$ in MATLAB.

[End of Example]

Task 13: Differentiation on Polynomials

Consider the following equation:

$$y = x^3 + 2x^2 - x + 3$$

Use Differentiation on the Polynomial to find $\frac{dy}{dx}$

[End of Task]

Task 14: Differentiation on Polynomials

Find the derivative for the product:

$$(3x^2 + 6x + 9)(x^2 + 2x)$$

Use the **polyder(a,b)** function.

Another approach is to use define is to first use the **conv(a,b)** function to find the total polynomial, and then **use polyder(p)** function.

Try both methods, to see if you get the same answer.

[End of Task]

4.4 Numerical Integration

The integral of a function $f(x)$ is denoted as:

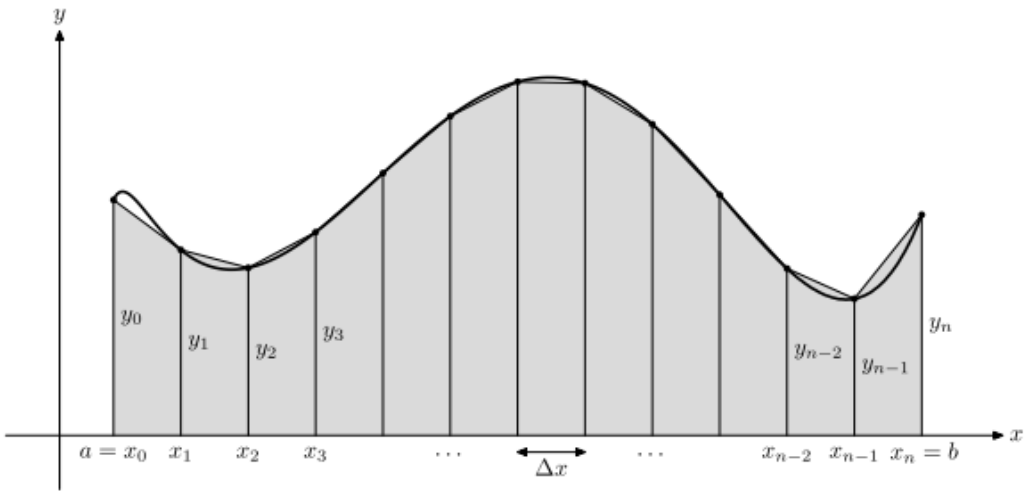
$$\int_a^b f(x)dx$$

An integral can be seen as the area under a curve. Given $y = f(x)$ the approximation of the Area (A) under the curve can be found dividing the area up into rectangles and then summing the contribution from all the rectangles:

$$A = \sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot (y_{i+1} + y_i)/2$$

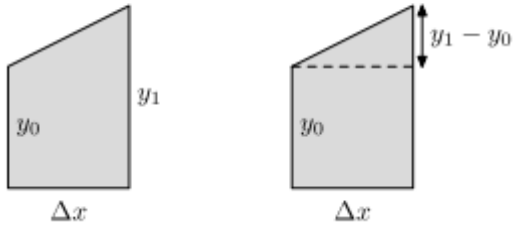
This is known as the trapezoid rule.

We approximate the integral by using n trapezoids formed by using straight line segments between the points (x_{i-1}, y_{i-1}) and (x_i, y_i) for $1 \leq i \leq n$ as shown in the figure below:



The area of a trapezoid is obtained by adding the area of a rectangle and a triangle:

$$A = y_0\,\Delta x + \frac{1}{2}(y_1 - y_0)\Delta x = \frac{(y_0 + y_1)\Delta x}{2}.$$



MATLAB offers functions for numerical integration, such as:

Function	Description	Example
diff	Difference and approximate derivative. DIFF(X), for a vector X, is [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)].	`>> dydx_num=diff(y)./diff(x);`
quad	Numerically evaluate integral, adaptive Simpson quadrature. Q = QUAD(FUN,A,B) tries to approximate the integral of calar-valued function FUN from A to B. FUN is a function handle. The function Y=FUN(X) should accept a vector argument X and return a vector result Y, the integrand evaluated at each element of X. Uses adaptive Simpson quadrature method	>>
quadl	Same as quad, but uses adaptive Lobatto quadrature method	>>
polyint	Integrate polynomial analytically. POLYINT(P,K) returns a polynomial representing the integral of polynomial P, using a scalar constant of integration K.	

Example:

Given the function:

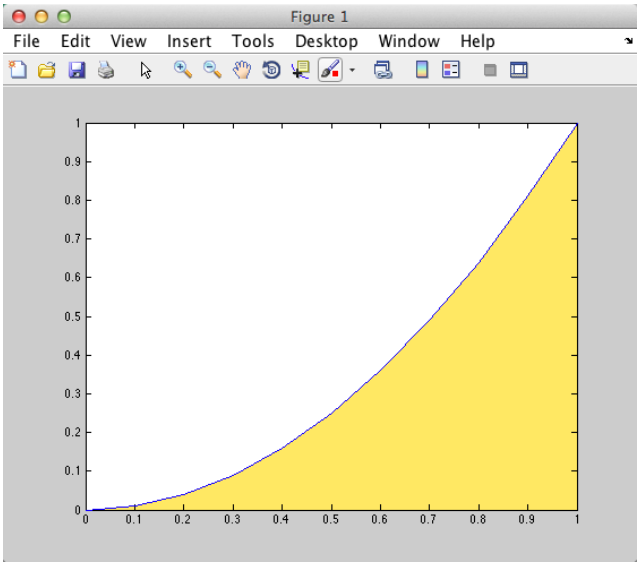
$$y = x^2$$

We know that the exact solution is:

$$\int_0^a x^2 dx = \frac{a^3}{3}$$

The integral from 0 to 1 is:

$$\int_0^1 x^2 dx = \frac{1}{3} \approx 0.3333$$



We will use the trapezoid rule and the **diff** function in MATLAB to solve the numerical integral of x^2 from 0 to 1.

The MATLAB code for this is:

```
x=0:0.1:1;
y=x.^2;

avg_y = y(1:length(x)-1) + diff(y)/2;
A = sum(diff(x).*avg_y)
```

Note!

The following two lines of code

```
avg_y = y(1:length(x)-1) + diff(y)/2;

A = sum(diff(x).*avg_y)
```

Implements this formula, known as the trapezoid rule:

$$A = \sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot (y_{i+1} + y_i)/2$$

The result from the approximation is:

```
A =
    0.3350
```

If we use the functions **quad** we get:

```
quad('x.^2', 0,1)
ans =
    0.3333
```

If we use the functions **quadl** we get:

```
quadl('x.^2', 0,1)
ans =
    0.3333
```

[End of Example]

Task 15: Numerical Integration

Use **diff**, **quad** and **quadl** on the following equation:

$$y = x^3 + 2x^2 - x + 3$$

Find the integral of y with respect to x, evaluated from -1 to 1

Compare the different methods.

The exact solution is:

$$\int_a^b (x^3 + 2x^2 - x + 3)dx = \left(\frac{x^4}{4} + \frac{2x^3}{3} - \frac{x^2}{2} + 3x\right)\Big|_a^b$$
$$= \frac{1}{4}(b^4 - a^4) + \frac{2}{3}(b^3 - a^3) - \frac{1}{2}(b^2 - a^2) + 3(b - a)$$

Compare the result with the exact solution.

Repeat the task for the following functions:

$$y = \sin(x)$$

$$y = x^5 - 1$$

[End of Task]

4.4.1 Integration on Polynomials

A polynomial is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

where $p_1, p_2, p_3, \ldots$ are the coefficients of the polynomial.

In MATLAB we can use the **polyint** function to perform integration on polynomials. This function works the same way as the **polyder** function which performs differentiation on polynomials.

Task 16: Integration on Polynomials

Consider the following equation:

$$y = x^3 + 2x^2 - x + 3$$

Find the integral of y with respect to x ($\int y dx$) using MATLAB.

[End of Task]

5 Optimization

Optimization is important in control and simulation applications. Optimization is based on finding the minimum of a given criteria function.

In MATLAB we can use the **fminbnd** and **fminsearch** functions. We will take a closer look of how to use these functions.

Function	Description	Example

fminbnd	X = FMINBND(FUN,x1,x2) attempts to find a local minimizer X of the function FUN in the interval x1 < X < x2. FUN is a function handle. FUN accepts scalar input X and returns a scalar function value F evaluated at X. FUN can be specified using @.
	FMINBND is a single-variable bounded nonlinear function minimization.
		`>> x = fminbnd(@cos,3,4)`
		`x =`
		`3.1416`

fminsearch	X = FMINSEARCH(FUN,X0) starts at X0 and attempts to find a local minimizer X of the function FUN. FUN is a function handle. FUN accepts input X and returns a scalar function value F evaluated at X. X0 can be a scalar, vector or matrix. FUN can be specified using @.
	FMINSEARCH is a multidimensional unconstrained nonlinear function minimization.
		`>> x = fminsearch(@sin,3)`
		`x =`
		`4.7124`

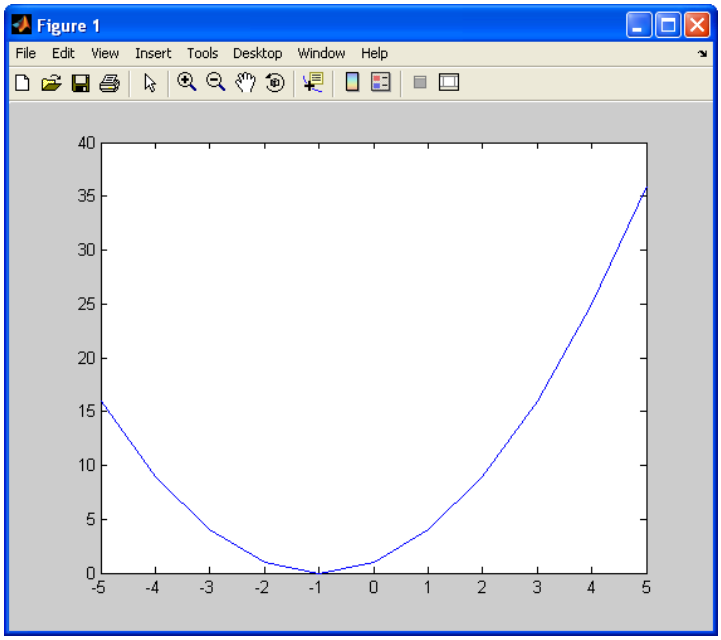
Example:

Given the following function:

$$f(x) = x^2 + 2x + 1$$

We will use **fminbnd** to find the minimum of the function.

We plot the function:



We write the following MATLAB Script:

```
x = -5:1:5;
f = mysimplefunc(x);
plot(x, f)

x_min = fminbnd(@mysimplefunc, -5, 5)
```

where the function (mysimplefunc.m) is defined like this:

```
function f = mysimplefunc(x)

f = x.^2 + 2.*x + 1;
```

This gives:

```
x_min =

    -1
```

→ The minimum of the function is -1. This can also be shown from the plot.

[End of Example]

Note! If a function has more than one variable, we need to use the **fminsearch** function.

Example:

Given the following function:

$$f(x,y) = 2(x-1)^2 + x - 2 + (y-2)^2 + y$$

We will use **fminsearch** to find the minimum of the function.

The MATLAB Code can be written like this:

```
[x,fval] = fminsearch(@myfunc, [1;1])
```

where the function is defined like this:

```
function f = myfunc(x)

f = 2*(x(1)-1).^2 + x(1) - 2 + (x(2)-2).^2 + x(2);
```

Note! The unknowns x and y is defined as a vector, i.e., $x_1 = x(1) = x$, $x_2 = x(2) = y$.

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

If there is more than one variable, you have to do it this way.

This gives:

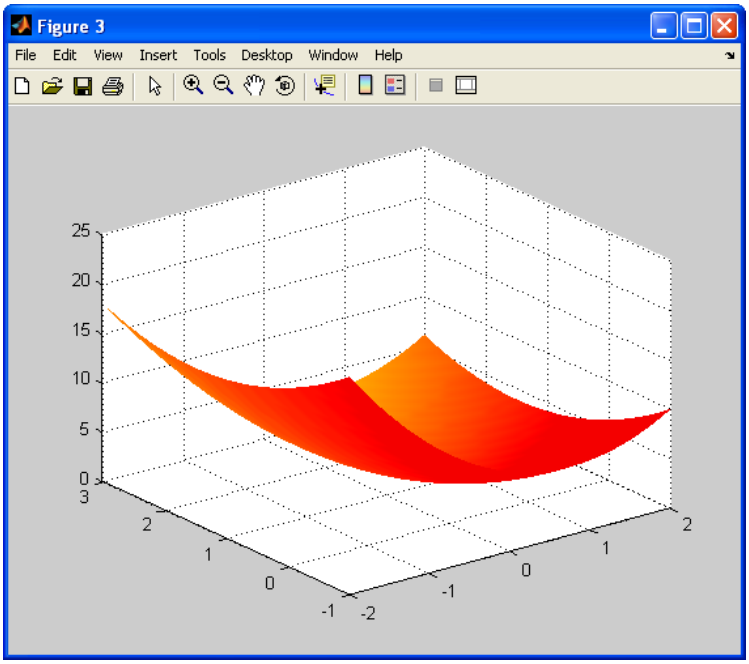
```
x =
    0.7500
    1.5000
fval =
    0.6250
```

→ The minimum is of the function is given by $x = 0.75$ and $y = 1.5$.

We can also plot the function:

```
clear,clc

[x,y] = meshgrid(-2:0.1:2, -1:0.1:3);

f = 2.*(x-1).^2 + x - 2 + (y-2).^2 + y;

figure(1)
surf(x,y,f)

figure(2)
mesh(x,y,f)

figure(3)
surfl(x,y,f)
shading interp;
colormap(hot);
```

For figure 3 we get:



[End of Example]

Task 17: Optimization

Given the following function:

$$f(x) = x^3 - 4x$$

→ Plot the function

→ Find the minimum for this function

[End of Task]

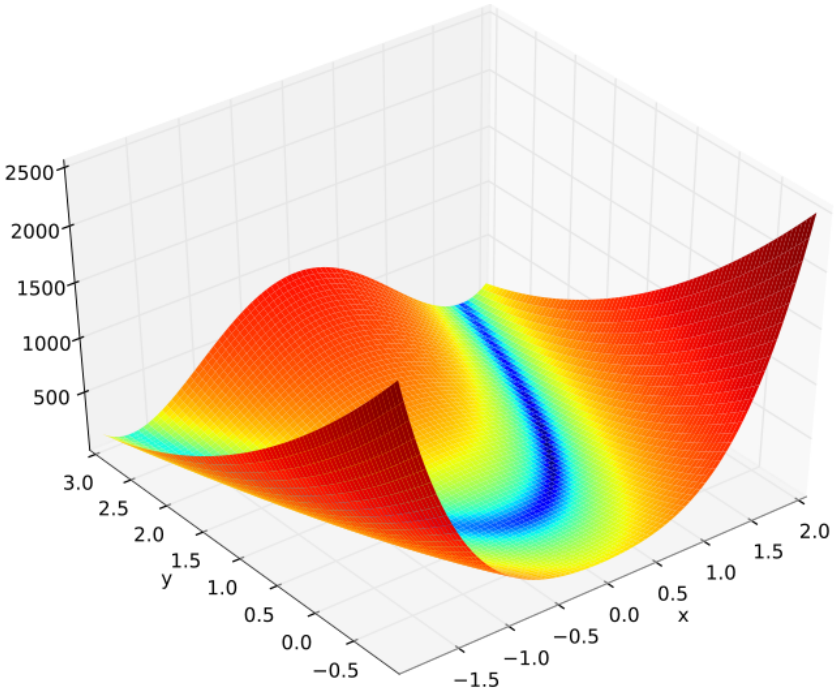
Task 18: Optimization - Rosenbrock's Banana Function

Given the following function:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

This function is known as Rosenbrock's banana function.

The function looks like this:



The global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. To converge to the global minimum, however, is difficult. But MATLAB will hopefully do the job for us.

Let's see if MATLAB can do the job for us.

→ Plot the function

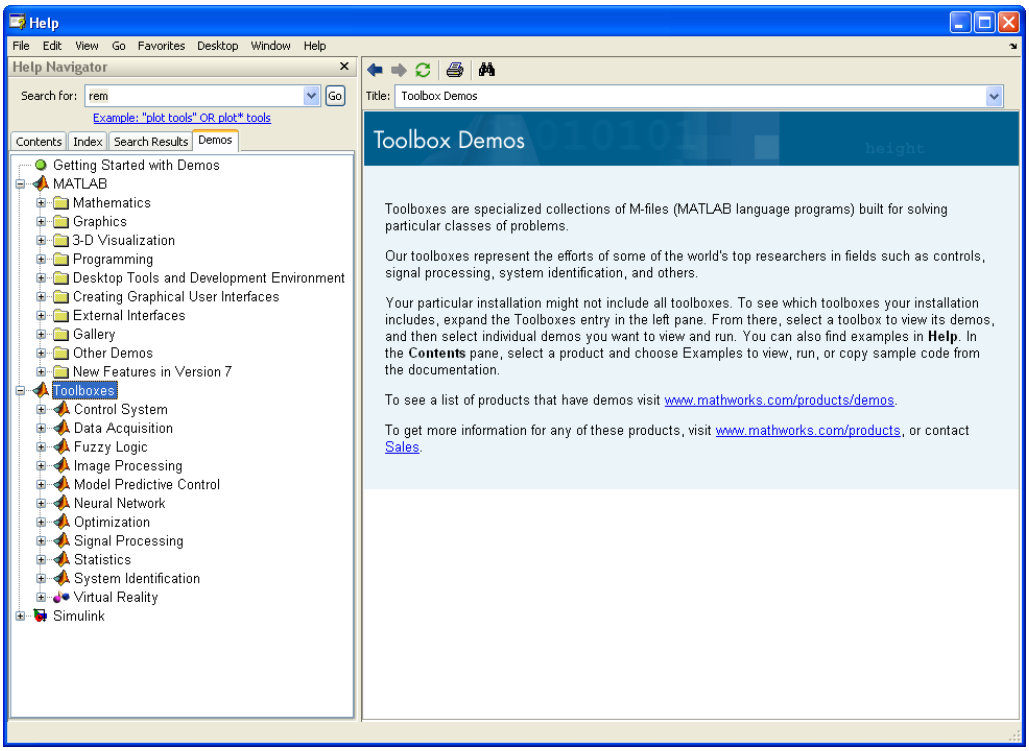
→ Find the minimum for this function

[End of Task]

6 Control System Toolbox

There are available lots of additional toolboxes for MATLAB. Toolboxes are specialized collections of M-files built for solving particular classes of problems, e.g.,

- Control System Toolbox
- Signal Processing Toolbox
- Statistics Toolbox
- System identification Toolbox
- etc.



Here we will take a closer look at the "**Control System Toolbox**".

Control System Toolbox builds on the foundations of MATLAB to provide functions designed for control engineering. Control System Toolbox is a collection of algorithms, written mostly as M-files, that implements common control system design, analysis, and modeling techniques. Convenient graphical user interfaces (GUIs) simplify typical control engineering tasks. Control systems can be modeled as transfer functions, in zero-pole-gain or state-space form, allowing you to use both classical and modern control techniques. You can manipulate both continuous-time and discrete-time systems. Conversions between various model

representations are provided. Time responses, frequency responses can be computed and graphed. Other functions allow pole placement, optimal control, and estimation. Finally, Control System Toolbox is open and extensible. You can create custom M-files to suit your particular application.

7 Transfer Functions

It is assumed you are familiar with basic control theory and transfer functions, if not you may skip this chapter.

7.1 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.

First order Transfer Function:

A first order transfer function is given on the form:

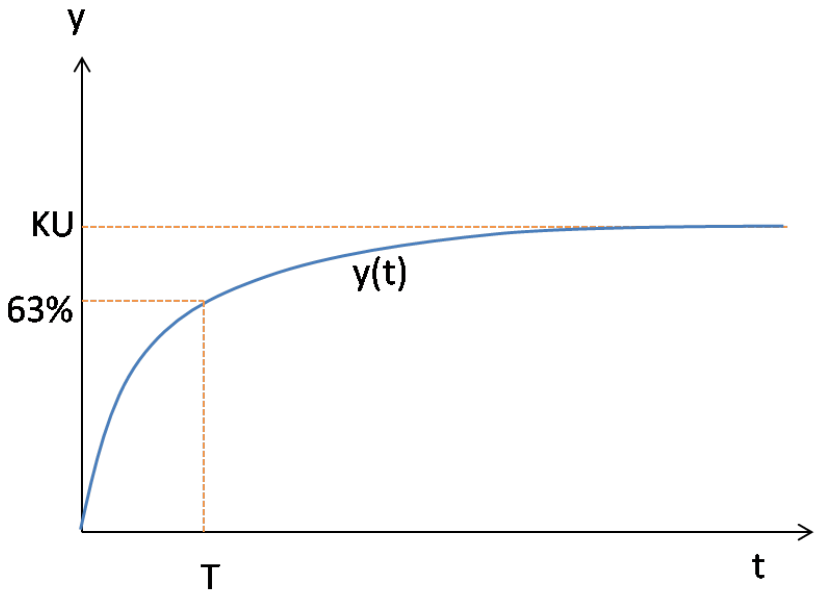
$$\boxed{H(s) = \frac{y(s)}{u(s)} = \frac{K}{Ts + 1}}$$

there

K is the Gain

T is the Time constant

A 1.order transfer function with time-delay has the following characteristic step response:

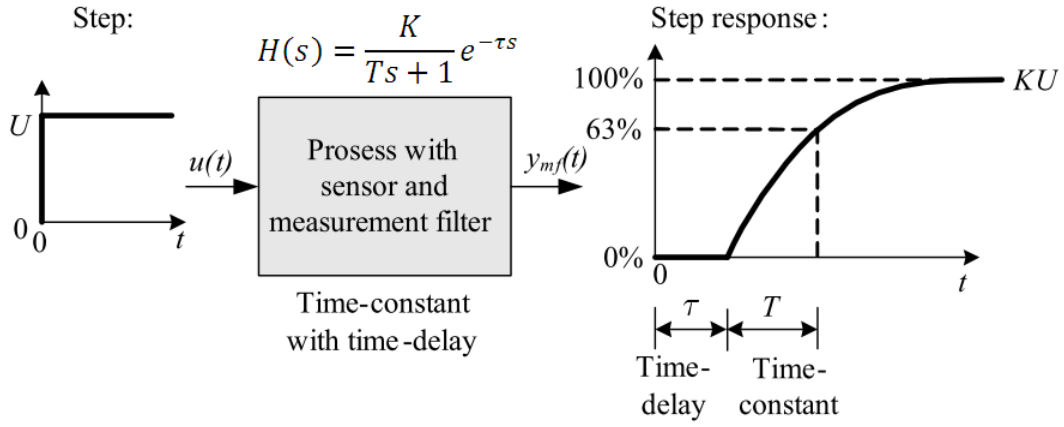


A first order transfer function with time-delay has the following transfer function:

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

Where τ is the time-delay.

A 1.order transfer function with time-delay has the following characteristic step response:



MATLAB have several functions for creating and manipulation of transfer functions:

Function	Description	Example

tf	Creates system model in transfer function form. You also can use this function to state-space models to transfer function form.	`>num=[1];`
		`>den=[1, 1, 1];`
		`>H = tf(num, den)`

pole	Returns the locations of the closed-loop poles of a system model.	`>num=[1]`
		`>den=[1,1]`
		`>H=tf(num,den)`
		`>pole(H)`

zero	Find the zeros	

step	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.	>num=[1,1]; >den=[1,-1,3]; >H=tf(num,den); >t=[0:0.01:10]; >step(H,t);
lsim	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.	>t = [0:0.1:10] >u = sin(0.1*pi*t)' >lsim(SysIn, u, t)
conv	Computes the convolution of two vectors or matrices.	>C1 = [1, 2, 3]; >C2 = [3, 4]; >C = conv(C1, C2)
series	Connects two system models in series to produce a model SysSer with input and output connections you specify	>Hseries = series(H1,H2)
feedback	Connects two system models together to produce a closed-loop model using negative or positive feedback connections	>SysClosed = feedback(SysIn_1, SysIn_2)
c2d	Convert from continuous- to discrete-time models	
d2c	Convert from discrete- to continuous-time models	

Before you start, you should use the Help system in MATLAB to read more about these functions. Type "help <functionname>" in the Command window.

Task 19: Transfer function

Use the **tf** function in MATLAB to define the transfer function above. Set $K = 2$ and $T = 3$.

Type "`help tf`" in the Command window to see how you use this function.

Example:

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

[End of Task]

7.2 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

ζ zeta is the relative damping factor

ω_0[rad/s] is the undamped resonance frequency.

Task 20: 2.order Transfer function

Define the transfer function using the **tf** function.

Set $K = 1, \omega_0 = 1$

→ Plot the step response (use the **step** function in MATLAB) for different values of ζ. Select ζ as follows:

$$\zeta > 1$$

$$\zeta = 1$$

$$0 < \zeta < 1$$

$$\zeta = 0$$

$$\zeta < 0$$

Tip! From control theory we have the following:

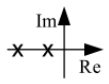
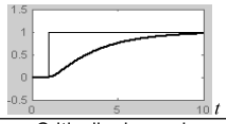
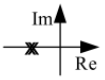
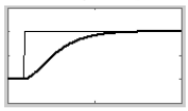
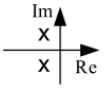
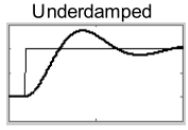
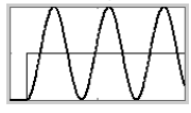
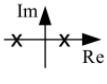
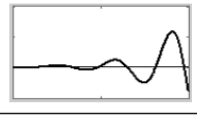
Value of ζ	Poles p_1 and p_2	Type of step response $y(t)$
$\zeta > 1$	Real and distinct 	Overdamped 
$\zeta = 1$	Real and multiple 	Critically damped 
$0 < \zeta < 1$	Complex conj. 	Underdamped 
$\zeta = 0$	Imaginary 	Undamped 
$\zeta < 0$	Pos. real part 	Unstable 

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

So you should get similar step responses as shown above.

[End of Task]

Task 21: Time Response

Given the following system:

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

Plot the time response for the transfer function using the **step** function. Let the time-interval be from 0 to 10 seconds, e.g., define the time vector like this:

```
t=[0:0.01:10]
```

and then use the function step(H,t).

[End of Task]

7.3 Analysis of Standard Functions

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

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

Task 22: Integrator

The transfer function for an Integrator is as follows:

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

→Find the pole(s)

→ Plot the Step response: Use different values for K, e.g., $K = 0.2, 1, 5$. Use the **step** function in MATLAB.

[End of Task]

Task 23: 1. order system

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

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

→ Find the pole(s)

→ Plot the Step response. Use the **step** function in MATLAB.

- Step response 1: Use different values for K, e.g., $K = 0.5, 1, 2$. Set $T = 1$
- Step response 2: Use different values for T, e.g., $T = 0.2, 0.5, 1, 2, 4$. Set $K = 1$

[End of Task]

Task 24: 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
- ζ zeta is the relative damping factor
- ω_0[rad/s] is the undamped resonance frequency.

→ Find the pole(s)

→ Plot the Step response: Use different values for ζ, e.g., $\zeta = 0.2, 1, 2$. Set $\omega_0 = 1$ and K=1. Use the **step** function in MATLAB.

Tip! From control theory we have the following:

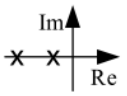
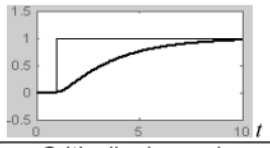
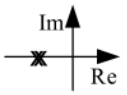
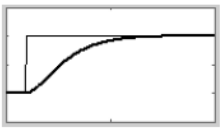
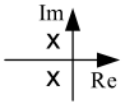
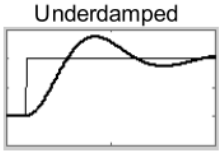
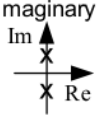
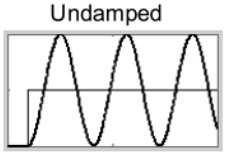
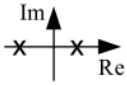
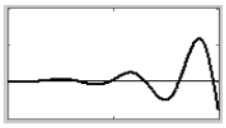
Value of ζ	Poles p_1 and p_2	Type of step response $y(t)$
$\zeta > 1$	Real and distinct 	Overdamped 
$\zeta = 1$	Real and multiple 	Critically damped 
$0 < \zeta < 1$	Complex conj. 	Underdamped 
$\zeta = 0$	Imaginary 	Undamped 
$\zeta < 0$	Pos. real part 	Unstable 

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

So you should get similar step responses as shown above.

[End of Task]

Task 25: 2. order system – Special Case

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

$$H(s) = \frac{K}{(T_1 s + 1)(T_2 s + 1)}$$

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

$$H(s) = H_1(s)H_1(s) = \frac{K}{(T_1 s + 1)} \cdot \frac{1}{(T_2 s + 1)} = \frac{K}{(T_1 s + 1)(T_2 s + 1)}$$

Set $T_1 = 2$ and $T_2 = 5$

→ Find the pole(s)

→ Plot the Step response. Set K=1. Set $T_1 = 1\ and\ T_2 = 0$, $T_1 = 1\ and\ T_2 = 0.05$, $T_1 = 1\ and\ T_2 = 0.1$, $T_1 = 1\ and\ T_2 = 0.25$, $T_1 = 1\ and\ T_2 = 0.5$, $T_1 = 1\ and\ T_2 = 1$. Use the **step** function in MATLAB.

[End of Task]

8 State-space Models

It is assumed you are familiar with basic control theory and state-space models, if not you may skip this chapter.

8.1 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 MATLAB.

We have the following equations:

$$\dot{x}_1 = a_{11}x_1 + a_{21}x_2 + \cdots + a_{n1}x_n + \ b_{11}u_1 + b_{21}u_2 + \cdots + b_{n1}u_n$$

$$\vdots$$

$$\dot{x}_n = a_{1m}x_1 + a_{2m}x_2 + \cdots + a_{nm}x_n + \ b_{1m}u_1 + b_{2m}u_2 + \cdots + b_{n1}u_n$$

$$\vdots$$

This gives on vector form:

$$\underbrace{\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \vdots\\ \dot{x}_n\end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix}a_{11} & \cdots & a_{n1}\\ \vdots & \ddots & \vdots\\ a_{1m} & \cdots & a_{nm}\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}x_1\\ x_2\\ \vdots\\ x_n\end{bmatrix}}_{x} + \underbrace{\begin{bmatrix}b_{11} & \cdots & b_{n1}\\ \vdots & \ddots & \vdots\\ b_{1m} & \cdots & b_{nm}\end{bmatrix}}_{B}\underbrace{\begin{bmatrix}u_1\\ u_2\\ \vdots\\ u_n\end{bmatrix}}_{u}$$

$$\underbrace{\begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_n\end{bmatrix}}_{y} = \underbrace{\begin{bmatrix}c_{11} & \cdots & c_{n1}\\ \vdots & \ddots & \vdots\\ c_{1m} & \cdots & c_{nm}\end{bmatrix}}_{C}\underbrace{\begin{bmatrix}x_1\\ x_2\\ \vdots\\ x_n\end{bmatrix}}_{x} + \underbrace{\begin{bmatrix}d_{11} & \cdots & d_{n1}\\ \vdots & \ddots & \vdots\\ d_{1m} & \cdots & d_{nm}\end{bmatrix}}_{D}\underbrace{\begin{bmatrix}u_1\\ u_2\\ \vdots\\ u_n\end{bmatrix}}_{u}$$

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

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

$$y = Cx + Du$$

Example:

Given the following equations:

$$\dot{x}_1 = -\frac{1}{A_t}x_2 + \frac{1}{A_t}K_p u$$

$$\dot{x}_2 = 0$$

These equations can be written on the compact state-space form:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \underbrace{\begin{bmatrix}0 & -\frac{1}{A_t}\\ 0 & 0\end{bmatrix}}_{A}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \underbrace{\begin{bmatrix}\frac{K_p}{A_t}\\ 0\end{bmatrix}}_{B}u$$

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

[End of Example]

MATLAB have several functions for creating and manipulation of State-space models:

Function	Description	Example

ss	Constructs a model in state-space form. You also can use this function to convert transfer function models to state-space form.	`>A = [1 3; 4 6];`
		`>B = [0; 1];`
		`>C = [1, 0];`
		`>D = 0;`
		`>sysOutSS = ss(A, B, C, D)`

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

lsim	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.	`>t = [0:0.1:10]`
		`>u = sin(0.1*pi*t)'`
		`>lsim(SysIn, u, t)`

c2d	Convert from continuous- to discrete-time models	
d2c	Convert from discrete- to continuous-time models	

Example:

```
% Creates a state-space model
A = [1 3; 4 6];
B = [0; 1];
C = [1, 0];
D = 0;
SysOutSS = ss(A, B, C, D)
```

[End of Example]

Before you start, you should use the Help system in MATLAB to read more about these functions. Type "help <functionname>" in the Command window.

8.2 Tasks

Task 26: State-space model

Implement the following equations as a state-space model in MATLAB:

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

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

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

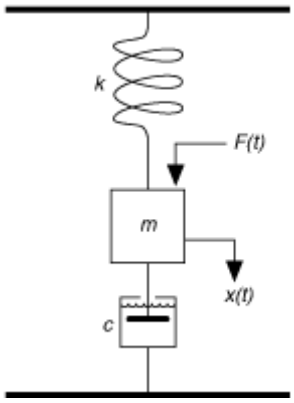
→ Find the Step Response

→ Find the transfer function from the state-space model using MATLAB code.

[End of Task]

Task 27: Mass-spring-damper system

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 MATLAB.

Set $c = 1,\ m = 1,\ k = 50$ (try also with other values to see what happens).

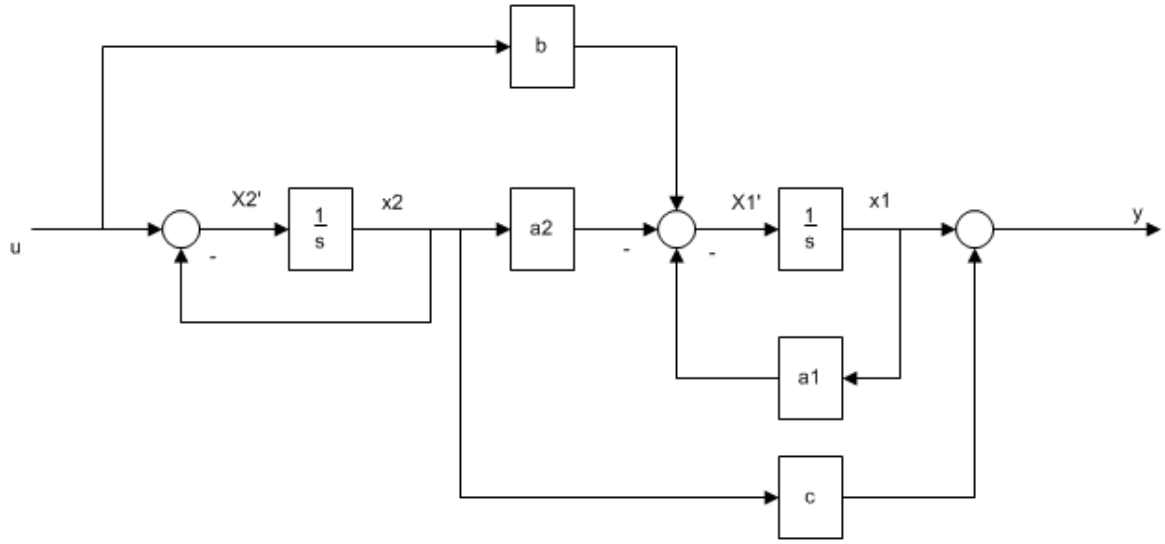
→Apply a step in F (u) and use the **step** function in MATLAB to simulate the result.

→ Find the transfer function from the state-space model

[End of Task]

Task 28: Block Diagram

Find the state-space model from the block diagram below and implement it in MATLAB.



Set

$$a_1 = 5$$

$$a_2 = 2$$

And b=1, c=1

→ Simulate the system using the **step** function in MATLAB

[End of Task]

8.3 Discrete State-space Models

For Simulation and Control in computers discrete systems are very important.

Given the continuous linear state space-model:

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

$$y = Cx + Du$$

Or given the discrete linear state space-model

$$x_{k+1} = \Phi x_k + \Gamma u_k$$

$$y_k = Cx_k + Du_k$$

Or it is also normal to use the same notation for discrete systems:

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

But the matrices A, B, C, D is of course not the same as in the continuous system.

MATLAB has several functions for dealing with discrete systems:

Function	Description	Example
c2d	Convert from continuous- to discrete-time models. You may specify which Discretization method to use	`>>c2d(sys,Ts)` `>>c2d(sys,Ts,'tustin')`
d2c	Convert from discrete- to continuous- time models	`>>`

Before you start, you should use the Help system in MATLAB to read more about these functions. Type "help <functionname>" in the Command window.

Task 29: Discretization

The state-space model for the system is:

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

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Set some arbitrary values for k, c and m.

Find the discrete State-space model using MATLAB.

[End of Task]

9 Frequency Response

In this chapter we assume that you are familiar with basic control theory and frequency response from previous courses in control theory/process control/cybernetics. If not, you may skip this chapter.

9.1 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 ω [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 (ϕ).

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 ϕ is a function of the frequency ω 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:

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

The **Phase** function:

$$\boxed{\phi(\omega) = \angle H(j\omega)}$$

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

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

MATLAB have several functions for frequency response:

Function	Description	Example
bode	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.	>num=[4]; >den=[2, 1]; >H = **tf**(num, den) >**bode**(H)
bodemag	Creates the Bode magnitude plot of a system model. If you do not specify an output, this function creates a plot.	>[mag, wout] = **bodemag**(SysIn) >[mag, wout] = **bodemag**(SysIn, [wmin wmax]) >[mag, wout] = **bodemag**(SysIn, wlist)
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.	>num = [1] >den = [1, 5, 6] >H = **tf**(num, den) **margin**(H)

Example:

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

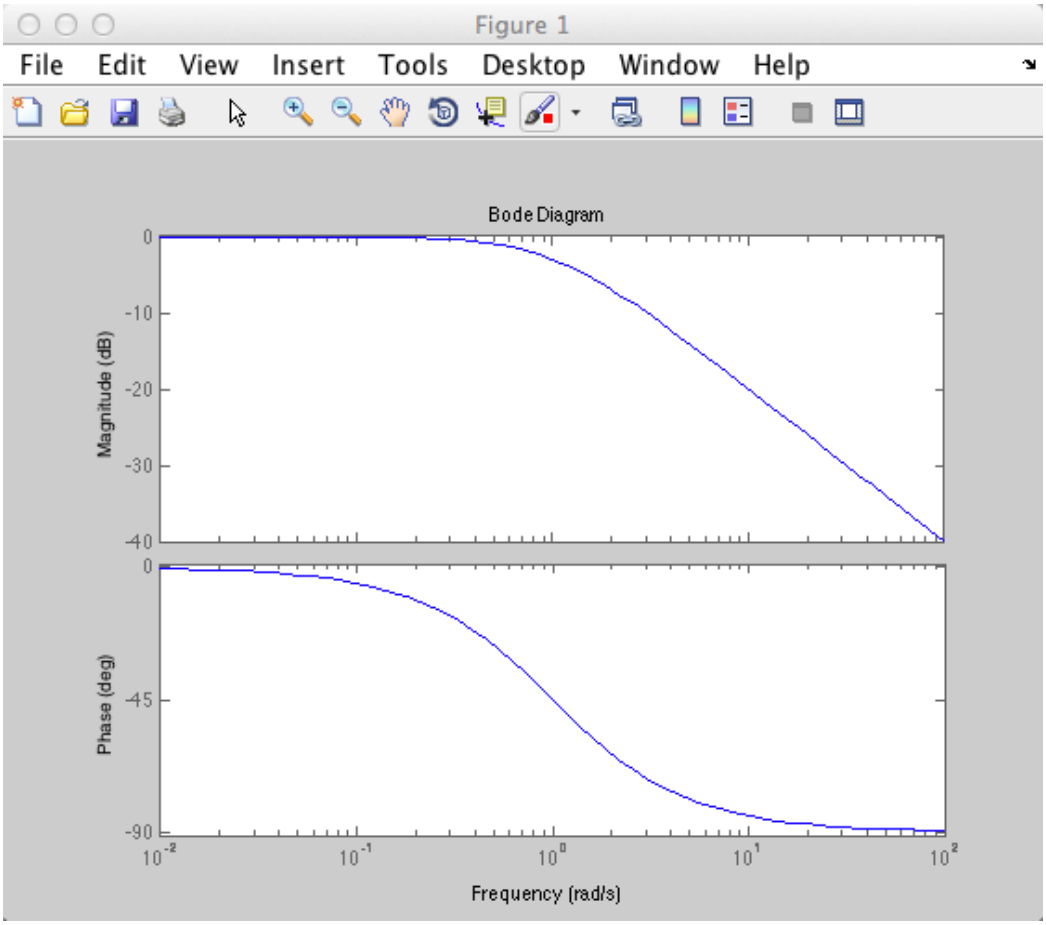
We have the following transfer function

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

Below we see the script for creating the **frequency response** of the system in a bode plot using the **bode** function in MATLAB. 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);
```

The Bode plot:



[End of Example]

Before you start, you should use the Help system in MATLAB to read more about these functions. Type "help <functionname>" in the Command window.

9.2 Tasks

Task 30: 1. order system

We have the following transfer function:

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

→ What is the **break frequency**?

→ Set up the mathematical expressions for $A(\omega)$ and $\phi(\omega)$. Use "Pen & Paper" for this Assignment.

→ Plot the **frequency response** of the system in a bode plot using the **bode** function in MATLAB. Discuss the results.

→ Find $A(\omega)$ and $\phi(\omega)$ for the following frequencies using MATLAB code (use the **bode** function):

$\boldsymbol{\omega}$	$\boldsymbol{A(\omega)[dB]}$	$\boldsymbol{\phi(\omega)(degrees)}$
0.1		
0.16		
0.25		
0.4		
0.625		
2.5		

Make sure $A(\omega)$ is in dB.

→ Find $A(\omega)$ and $\phi(\omega)$ for the same frequencies above using the mathematical expressions for $A(\omega)$ and $\phi(\omega)$. **Tip:** Use a For Loop or define a vector w=[0.1, 0.16, 0.25, 0.4, 0.625, 2.5].

[End of Task]

Task 31: Bode Diagram

We have the following transfer function:

$$H(S) = \frac{(5s + 1)}{(2s + 1)(10s + 1)}$$

→ What is the **break frequencies**?

→ Set up the mathematical expressions for $A(\omega)$ and $\phi(\omega)$. Use "Pen & Paper" for this Assignment.

→ Plot the **frequency response** of the system in a bode plot using the **bode** function in MATLAB. Discuss the results.

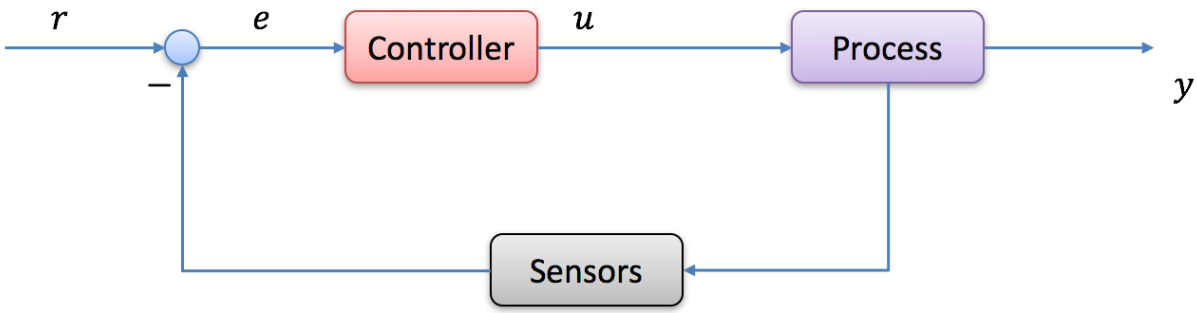
→ Find $A(\omega)$ and $\phi(\omega)$ for some given frequencies using MATLAB code (use the **bode** function).

→ Find $A(\omega)$ and $\phi(\omega)$ for the same frequencies above using the mathematical expressions for $A(\omega)$ and $\phi(\omega)$. Tip: use a For Loop or define a vector w=[0.01, 0.1, …].

[End of Task]

9.3 Frequency response Analysis

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



9.3.1 Loop Transfer Function

The **Loop transfer function** $L(s)$ 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

9.3.2 Tracking Transfer Function

The **Tracking transfer function** $T(s)$ 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** is good if the tracking function T has value equal to or close to 1:

$$|T| \approx 1$$

9.3.3 Sensitivity Transfer Function

The **Sensitivity transfer function** $S(s)$ 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:

$$|S| \approx 0 \; or \; |S| \ll 1$$

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:

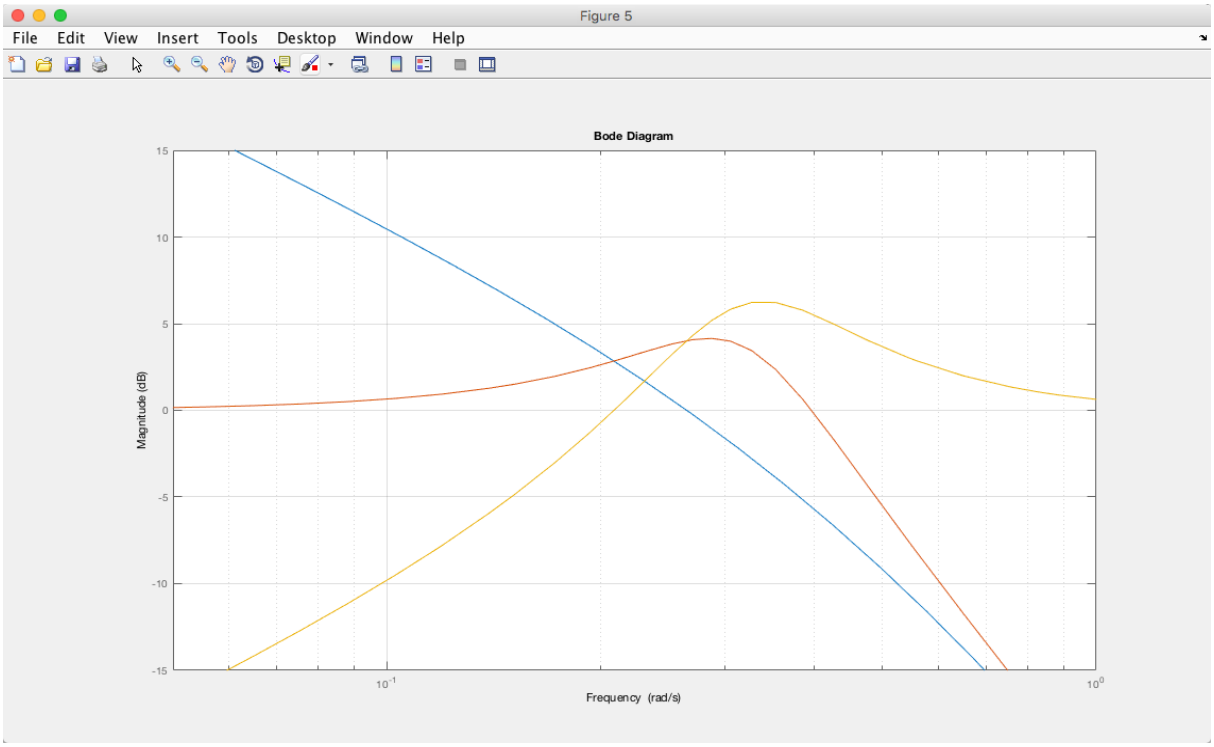
The **Tracking Property** is **good** if:

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

The **Tracking Property** is **poor** if:

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

If we plot L, T and S in a Bode plot we get a plot like this:



Where the following Bandwidths $\omega_t, \omega_c, \omega_s$ are defined:

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

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

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

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

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

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

Task 32: Frequency Response Analysis

Given the following system:

Process transfer function:

$$\boxed{H_p = \frac{K}{s}}$$

Where $K = \frac{K_S}{\varrho A}$, where $K_S = 0{,}556,\ A = 13{,}4,\ \varrho = 145$

Measurement (sensor) transfer function:

$$\boxed{H_m = K_m}$$

Where $K_m = 1$.

Controller transfer function (PI Controller):

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

Set Kp = 1,5 og Ti = 1000 sec.

→ Define the **Loop transfer function** $\boldsymbol{L(s)}$, **Sensitivity transfer function** $\boldsymbol{S(s)}$ and **Tracking transfer function** $\boldsymbol{T(s)}$ and in MATLAB.

→ Plot the Loop transfer function $L(s)$, the Tracking transfer function $T(s)$ and the Sensitivity transfer function $S(s)$ in the same Bode diagram. Use, e.g., the **bodemag** function in MATLAB.

→ Find the bandwidths $\omega_t, \omega_c, \omega_s$ from the plot above.

→ Plot the step response for the Tracking transfer function $T(s)$

[End of Task]

9.4 Stability Analysis of Feedback Systems

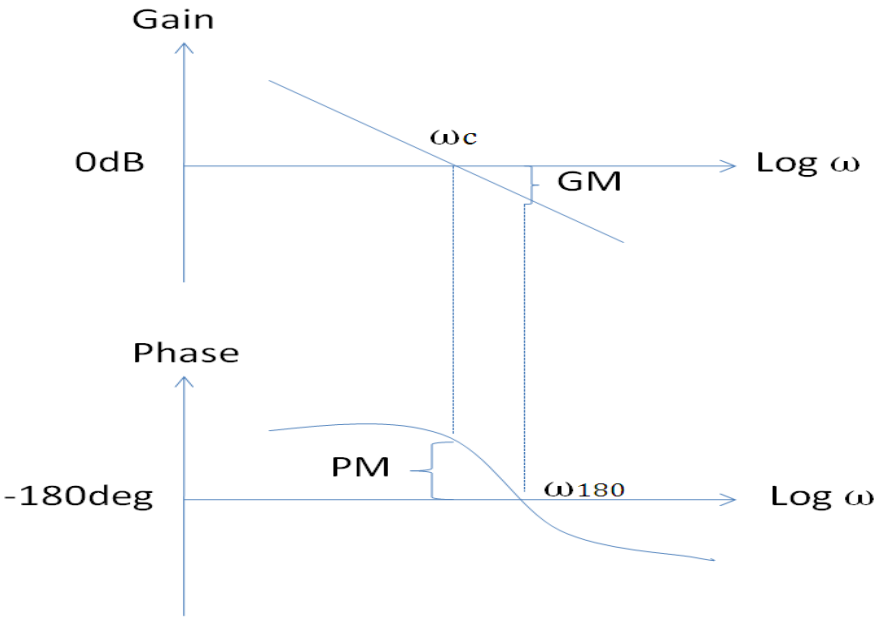
Gain Margin (GM) and Phase Margin (PM) are important design criteria for analysis of feedback control systems.

A dynamic system has one of the following stability properties:

- Asymptotically stable system
- Marginally stable system
- Unstable system

The Gain Margin – GM (ΔK) is how much the loop gain can increase before the system become unstable.

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



Where:

- ω_{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** (ΔK) is the gain margin(s) of the system.
- ω_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** (φ) is the phase margin(s) of the system.

Note! ω_{180} and ω_c are called the crossover-frequencies

The definitions are as follows:

Gain Crossover-frequency - ω_c :

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

Phase Crossover-frequency - ω_{180} :

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

Gain Margin - GM (ΔK):

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

or:

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

Phase margin PM (φ):

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

We have that:

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

We use the following functions in MATLAB: **tf**, **bode**, **margins** and **margin**.

Task 33: Stability Analysis

Given the following system:

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

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

Plot a bode diagram where the crossover-frequencies, GM and PM are illustrated. **Tip!** Use the **margin** function in MATLAB.

[End of Task]

10 Additional Tasks

If you have time left or need more practice, solve the tasks below.

Task 34: ODE Solvers

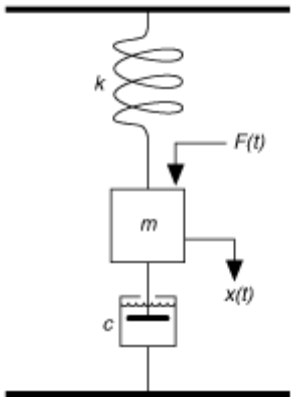
Use the ode45 function to solve and plot the results of the following differential equation in the interval $[t_0, t_f]$:

$$\mathbf{3}\boldsymbol{w}' + \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{t}^{\mathbf{2}}}\boldsymbol{w} = \boldsymbol{cost},\ t_0 = 0, t_f = 5, w(t_0) = 1$$

[End of Task]

Task 35: Mass-spring-damper system

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:

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

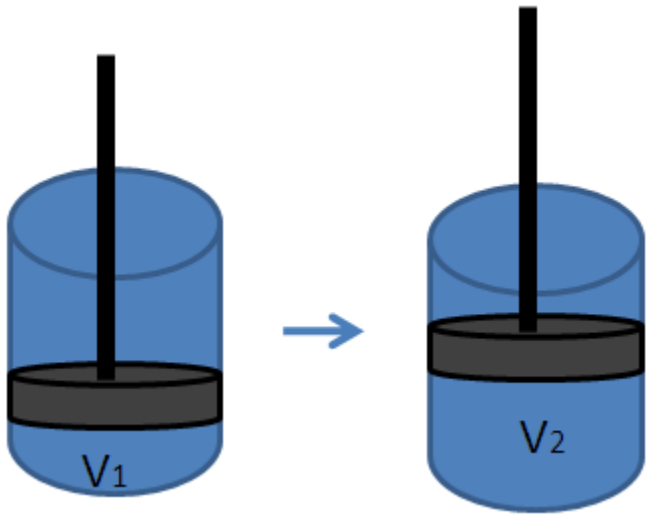
Set $c = 1, m = 1, k = 50$.

→ Solve and Plot the system using one or more of the built-in solvers (use, e.g., **ode32**) in MATLAB. Apply a step in F (which is the control signal u).

[End of Task]

Task 36: Numerical Integration

Given a piston cylinder device:



→ Find the work produced in a piston cylinder device by solving the equation:

$$W = \int_{V_1}^{V_2} PdV$$

Assume the ideal gas low applies:

$$PV = nRT$$

where

- P= pressure
- V=volume, m^3
- n=number of moles, kmol
- R=universal gas constant, 8.314 kJ/kmol K
- T=Temperature, K

We also assume that the piston contains 1 mol of gas at 300K and that the temperature is constant during the process. $V_1 = 1m^3, V_2 = 5m^3$

Use both the **quad** and **quadl** functions. Compare with the exact solution by solving the integral analytically.

[End of Task]

Task 37: State-space model

The following model of a pendulum is given:

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

$$\dot{x}_2 = -\frac{g}{r}x_1 - \frac{b}{mr^2}x_2$$

where m is the mass, r is the length of the arm of the pendulum, g is the gravity, b is a friction coefficient.

→ Define the state-space model in MATLAB

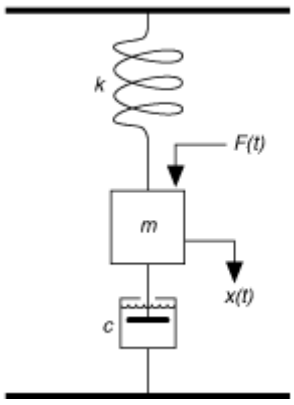
→ Solve the differential equations in MATLAB and plot the results.

Use the following values $g = 9.81, m = 8, r = 5, b = 10$

[End of Task]

Task 38: lsim

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:

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

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

→ Simulate the system using the **lsim** function in the Control System Toolbox.

Set c=1, m=1, k=50.

[End of Task]

Appendix A – MATLAB Functions

Numerical Techniques

Here are some descriptions for the most used MATLAB functions for Numerical Techniques.

Solving Ordinary Differential Equations

MATLAB offers lots of ode solvers, e.g.:

Function	Description	Example
ode23		
ode45		

Interpolation

MATLAB offers functions for interpolation, e.g.:

Function	Description	Example
interp1		

Curve Fitting

Here are some of the functions available in MATLAB used for curve fitting:

Function	Description	Example
polyfit	P = POLYFIT(X,Y,N) finds the coefficients of a polynomial P(X) of degree N that fits the data Y best in a least-squares sense. P is a row vector of length N+1 containing the polynomial coefficients in descending powers, P(1)*X^N + P(2)*X^(N-1) +...+ P(N)*X + P(N+1).	`>>polyfit(x,y,1)`
polyval	Evaluate polynomial. Y = POLYVAL(P,X) returns the value of a polynomial P evaluated at X. P is a vector of length N+1 whose elements are the coefficients of the polynomial in descending powers. Y = P(1)*X^N + P(2)*X^(N-1) + ... + P(N)*X + P(N+1)	

Numerical Differentiation

MATLAB offers functions for numerical differentiation, e.g.:

Function	Description	Example
diff	Difference and approximate derivative. DIFF(X), for a vector X, is [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)].	`>> dydx_num=diff(y)./diff(x);`
polyder	Differentiate polynomial. POLYDER(P) returns the derivative of the polynomial whose coefficients are the elements of vector P. POLYDER(A,B) returns the derivative of polynomial A*B.	`>>p=[1,2,3];` `>>polyder(p)`

Numerical Integration

MATLAB offers functions for numerical integration, such as:

Function	Description	Example
diff	Difference and approximate derivative. DIFF(X), for a vector X, is [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)].	`>> dydx_num=diff(y)./diff(x);`
quad	Numerically evaluate integral, adaptive Simpson quadrature. Q = QUAD(FUN,A,B) tries to approximate the integral of calar-valued function FUN from A to B. FUN is a function handle. The function Y=FUN(X) should accept a vector argument X and return a vector result Y, the integrand evaluated at each element of X. Uses adaptive Simpson quadrature method	`>>`
quadl	Same as quad, but uses adaptive Lobatto quadrature method	`>>`
polyint	Integrate polynomial analytically. POLYINT(P,K) returns a polynomial representing the integral of polynomial P, using a scalar constant of integration K.	

Optimization

MATLAB offers functions for local minimum, such as:

Function	Description	Example

fminbnd	X = FMINBND(FUN,x1,x2) attempts to find a local minimizer X of the function FUN in the interval x1 < X < x2. FUN is a function handle. FUN accepts scalar input X and returns a scalar function value F evaluated at X. FUN can be specified using @.
	FMINBND is a single-variable bounded nonlinear function minimization.
		`>> x = fminbnd(@cos,3,4)`
		`x =`
		`3.1416`

fminsearch	X = FMINSEARCH(FUN,X0) starts at X0 and attempts to find a local minimizer X of the function FUN. FUN is a function handle. FUN accepts input X and returns a scalar function value F evaluated at X. X0 can be a scalar, vector or matrix. FUN can be specified using @.
	FMINSEARCH is a multidimensional unconstrained nonlinear function minimization.
		`>> x = fminsearch(@sin,3)`
		`x =`
		`4.7124`

Control and Simulation

Here are some descriptions for the most used MATLAB functions for Control and Simulation.

Function	Description	Example

plot	Generates a plot. plot(y) plots the columns of y against the indexes of the columns.	`>X = [0:0.01:1];`
		`>Y = X.*X;`
		`>plot(X, Y)`

tf	Creates system model in transfer function form. You also can use this function to state-space models to transfer function form.	`>num=[1];`
		`>den=[1, 1, 1];`
		`>H = tf(num, den)`

pole	Returns the locations of the closed-loop poles of a system model.	`>num=[1]`
		`>den=[1,1]`
		`>H=tf(num,den)`
		`>poles(H)`

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

lsim	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.	`>t = [0:0.1:10]`
		`>u = sin(0.1*pi*t)'`
		`>lsim(SysIn, u, t)`

conv	Computes the convolution of two vectors or matrices.	`>C1 = [1, 2, 3];`
		`>C2 = [3, 4];`
		`>C = conv(C1, C2)`

series	Connects two system models in series to produce a model SysSer with input and output connections you specify	`>Hseries = series(H1,H2)`
feedback	Connects two system models together to produce a closed-loop model using negative or positive feedback connections	`>SysClosed = feedback(SysIn_1, SysIn_2)`

ss	Constructs a model in state-space form. You also can use this function to convert transfer function models to state-space form.	`>A = eye(2)`
		`>B = [0; 1]`
		`>C = B'`
		`>SysOutSS = ss(A, B, C)`

bode	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.	`>num=[4];`
		`>den=[2, 1];`
		`>H = tf(num, den)`
		`>bode(H)`

bodemag	Creates the Bode magnitude plot of a system model. If you do not specify an output, this function creates a plot.	`>[mag, wout] = bodemag(SysIn)`
		`>[mag, wout] = bodemag(SysIn, [wmin wmax])`
		`>[mag, wout] = bodemag(SysIn, wlist)`

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.	`>num = [1]`
		`>den = [1, 5, 6]`
		`>H = tf(num, den)`
		`margin(H)`

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)`
c2d	Convert from continuous- to discrete-time models	
d2c	Convert from discrete- to continuous-time models	

Hans-Petter Halvorsen, M.Sc.

E-mail: hans.p.halvorsen@hit.no

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

University College of Southeast Norway

www.usn.no
