070905/ThomasMunther
Halmstad University
School of Information Science, Computer and Electrical Engineering

Simple exercises in Matlab I

Start matlab by clicking the matlab icon!
Enter the window named command window.
Here you can write your command lines and execute them by pressing the return button. Everything following >> is written in the command window and is executed straight away. But of course one can also open the editor window and write a group of command lines and save it as an m-file. Then you have formed a programme. This programme can be executed by writing the name of the m-file in the command window.
You can open the editor window by File->New->M-file or by pressing the white sheet in the left corner.

First we are going to plot some wellknown functions as sin(t), sin(2*t) and 0.1*t for the time interval $0 \leq t \leq 10$. Write help to find out how a specific command works.
Let's see how the sine function operates !
>> help sin

A rather small help help occurs, but you can click on the links and go further.
By doing so you will eventually come to the Help Browser. Here you will find syntax, description, examples, definition and related commands.

Exercise 1:
Write the following in the command window:

```
t=linspace(0,10,600);          % makes 600 points between time 0 and 10
y=sin(t);
plot(t,y)                      % timeplot, y as function of t
```

If you forget the semicolon all the computations will be visible in the command window. Notice that everything that follows % is considered as an comment.

Hopefully you can see a sinecurve with amplitude 1 and time duration 10 seconds. But we are not fully satisfied yet. Add a gridpattern to the plot by
writing **grid** in the command window .
Would it not be nice with some labels and a title. This can easily be arranged by entering the figure window. Look for **insert** and then write ”time(sec)” as **xlabel** and ”Y” as **ylabel**. Now conclude all this with a figure **title**: ”Exercise 1”.

Finally we would like to change the curve line style of the curve. Press the arrow in the menu, then double click on the curve and you will find a new window called **Property Editor**. Change the **Line Style from solid to dotted line.**

What happened to the other functions. Return to the command window and plot the other two functions in the same figure window. This can be done by writing **hold** and then by repeating the sequence above.

```
y2=0.1*t;
y3=sin(2*t);
plot(t,y2,t,y3);                    % plots both y2 and y3 as functions of t
```

Notice that y, y2,y3 and t are all vectors and to do a plot we must always keep in mind that these vectors must have the same length (size). This can be checked by the command
size(y2)
size(t)

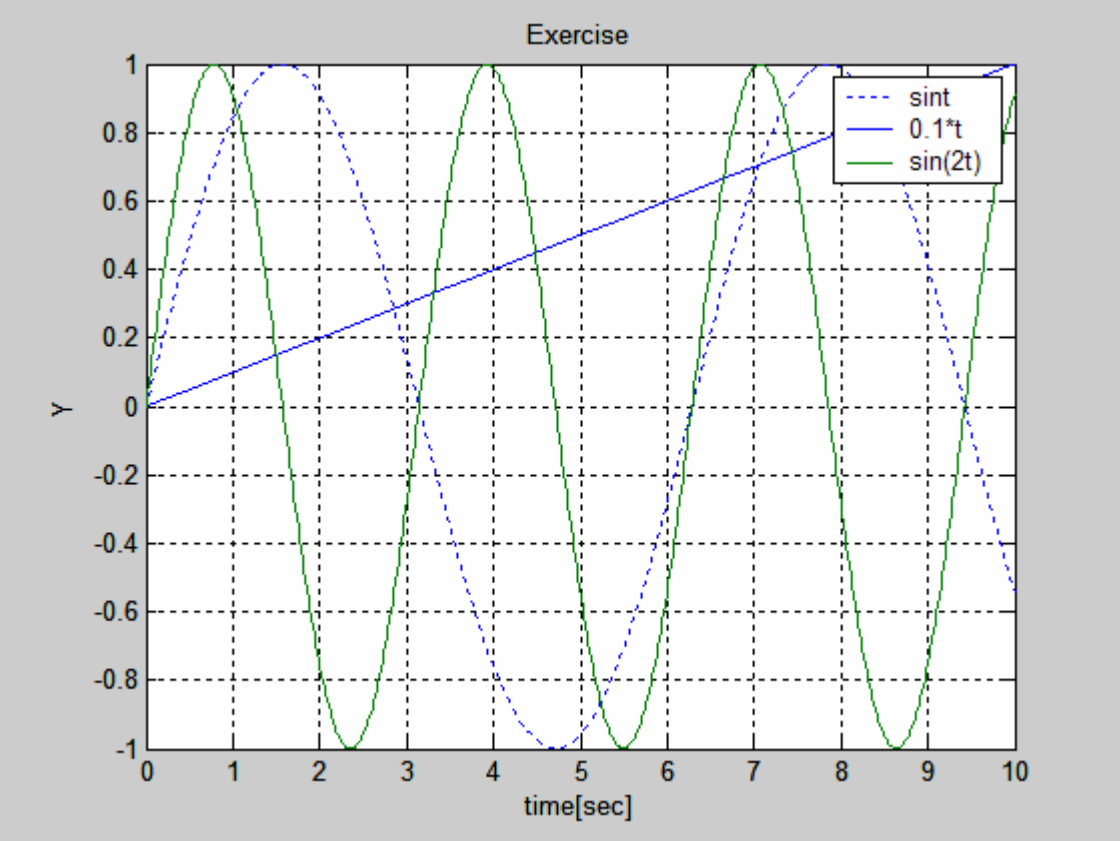


Figure 1

You will probably have a similar figure as above. In order to achieve the box.
Enter **insert->legend** in the figure menu. Edit the box.
Aside from command and figure window we also have two other windows. Workspace and command history. If these windows do not appear on the screen open them by **view-> command history, workspace** by entering the menu in matlab window. They have both changed since we started. In the workspace we can see all our variables created from the beginning. How many elements each vector variable contains and by double-clicking each and every element in the variable vectors can be seen.

In the command history we are able to see all our command lines stated from the beginning. This can also be used as a shortcut in case of needing these commands in the future.

Exercise 2:
Instead of writing all our command lines in the command window we can create an M-file (Macro-file) and write them there. These commands will be executed if you write the name of the M-file in the command window.
Enter **File->New->M-file** or **edit** filename will open up a texteditor. Let the filename be equation.

We are now going to plot this equation : e^(x)-e^(-x)=6

```
-------------------------------------------------------
% This m-file is created 2004 and gives a graphical solution to the equation
%  e^(x)-e^(-x)=6

x=linspace(-10,10)              % creates an x-vector from -10 to 10 in steps of 0.01
y=exp(x)-exp(-x)-6
plot(x,y),grid, zoom
-------------------------------------------------------
```

Save our file as equation.m in the catalogue work. Run the m-file by writing equation in the command window.
What is the solution to the equation ?
You are now able to get an graphical solution by zooming in the command window, when the function crosses the zero level.

Periodic Signals
We will see how signals like square, triangular, sawtooth, exponential vawes are created in matlab, but we will also look into valuable test signals like impulse, step and ramp signals can be done.

Square vawe: with a amplitude A , angular frequency w_0 and duty cycle rho.
The duty cycle, is the percent of the period in which the signal is positive.
Write the folllowing in the command window.

```
>> A=1;                         % by using semi-colon no result will be shown in
>> wo=10*pi;                    % the command window.
>> rho=50;
>> t=0: 0.001:1;                % startvalue 0sec, final value 1sec and time step 0.001 sec.
>> sq=A*square(wo*t,rho);       % calculates a square vawe based on the t vector.
>> plot(t,sq)                   % tries to make a continuos plot of data.
>>axis([ 0 1 -1.1 1.1])         % Rescales x- and y axis.
```

It should now show a fairly nice figure of a square vawe with amplitude 1.

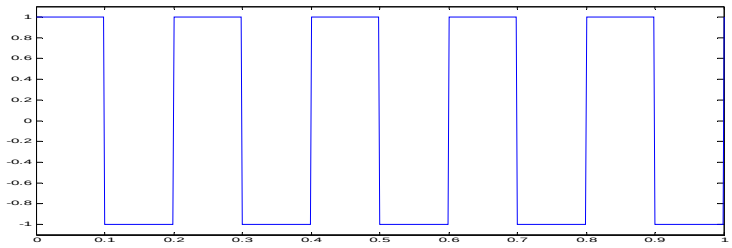


Figure 2

Sawtooth vawe:

```
>> A=1;
>> wo=10*pi;
>> t=0:0.001:1;                % a time vector with thousand elements. Start value
                               % 0 and stop value 1. Step time 0.001 sec.
>> y=A*sawtooth(wo*t,0.5);     % second parameter is the width, must be between 0
                               % and 1.
>> plot(t,y)
```

The plot can be seen in the figure below !

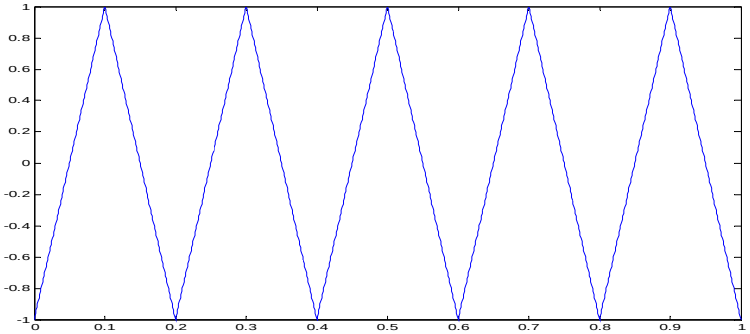


Figure 3

Exponential signals:

```
>> B=5;
>> a=3;
>> t= 0:0.001:1;
>> x= B*exp(-a*t)      % Calculates a decaying exponential.
>> plot(t,x)           % Makes a continuos plot.
```

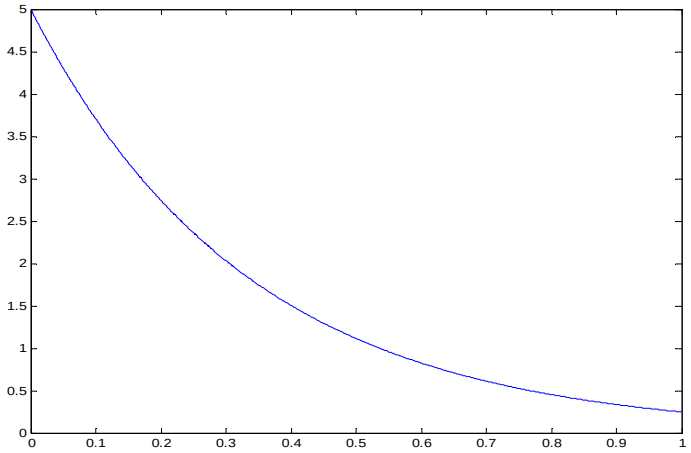


Figure 4

Notice that above we used a t vector with thousand elements. Based upon these t-values we have calculated the function x, but when using the plot commmand it connects these calculated points and the result looks very much like a continuos function. In fact this can never be the case in matlab. We would run out of memory very soon if we would try to implement such a function.

Sinusoidal Signals:
Matlab contains several trigonometric functions that can be used to generate several different signals, like tan, cos and sin . Suppose we would like to create a sinusoidal with amplitude 2 , angular frequency wo and phase angle phi.

A*sin(wo*t+phi)

Then we do like this:

```
>> A=2;
>> wo=20*pi;
>> phi=pi/6;
>> t=0:0.001:1;
>> sinus=A*sin(wo*t+phi);
>> plot(t,sinus)
```

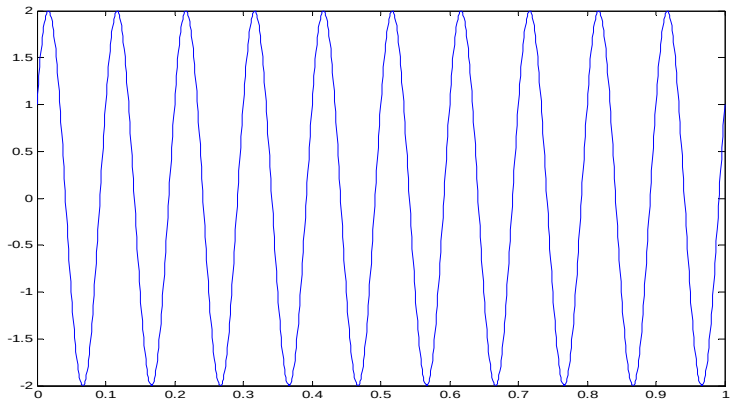


Figure 5

Or let it look more like discrete-time signal:

```
>> t=0:0.01:1;
>> sinus=A*sin(wo*t+phi);
>> stem(t,sinus)
```

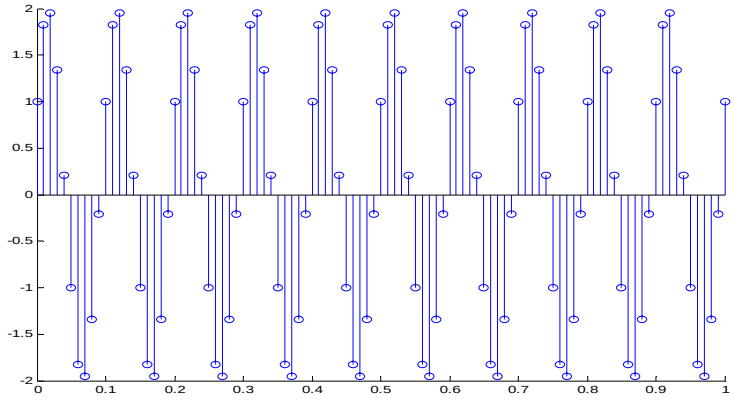


Figure 6

Since we dont use the same amount of calculation points and we use the stem command instead of the plot command means that no linear interpolation between the calculation points are carried out. This gives a plot that resembles a time-discrete sinusoidal signal very well. Every circle stands for the sample value at this time.

Non-periodic Signals

How can we create signals like step, impulse and ramp?

This is quite easy to achieve with matlab and its built-in commands like zeros and ones.

In matlab :

```
>> ones(1,12)   % means a matrix with one row and 12 columns containing only ones.
>>  zeros( 1,25) % means a matrix with one row and 25 columns containing only zeros.
```

With this knowledge we carry on to construct the
Step signal:

```
>> u= [ zeros(1,5) ones(1,25)];        % a vector containing 30 elements. The first 5 are
                                       % zeros and the next 25 are ones.
```

How does it look like ?
Try

```
>> plot(u)
```

or

```
>> stem(u)
```

Discrete-time impulse:

```
>> Delta=[ zeros(1,10), 1, zeros(1,10)]      % create a vector containing 21 elements.
                                             % Element number 11 is one the rest are
                                             %  zeros.
>> stem(Delta)                               % we plot the delta vector against its own indices.
```

The corresponding impulse plot is seen below:

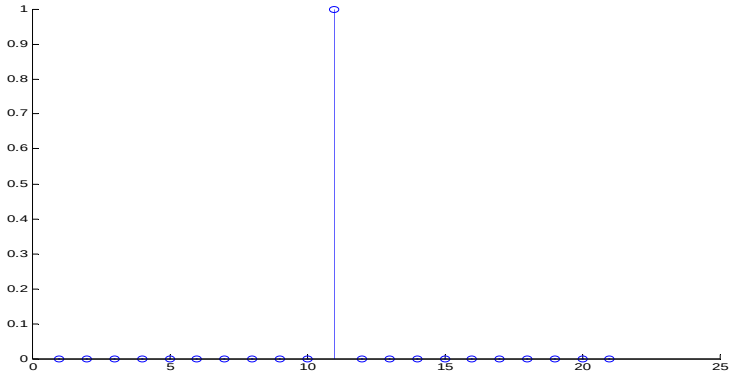


Figure 7

Ramp function:
This is a function that increases linearly. A unit ramp function increases the amplitude with in one time unit. We give an example:

```
>> ramp=0:1:10;
>> stem(ramp)
```

Convolution
The following are written in the matlab command window:

```
>> h=0.25*ones(1,4);          % impulse response
>> x=ones(1,10);              % input signal
>> n=0:12;                    % number of samples
>> y=conv(x,h);               % calculates the convolution
>> stem(n,y);                 % stem diagram of the covolution result.
>> xlabel('n'),ylabel('y[n]')
```

The result of the commands can be seen below in figure 8.

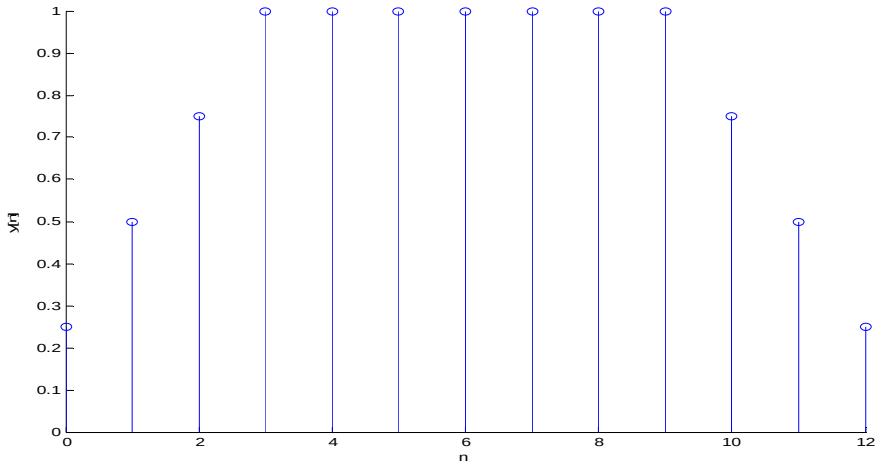


Figure 8

Step Response

The output from a system excited with a step input. The output step response lasts for infinite time. With the conv command we can evaluate the first p values of the step response. Determine the first 50 values of the step response of a system with a known impulse response $h[n]=0.5^n*u[n]$.

```
>> h=0.5.^[0:49];       % .^ performs element wise operation on the vector.
Other element wise operations are .* and ./  .
>> x=ones(1,50);
>> y=conv(x,h);
>> stem([0:49],y(1:50));
```

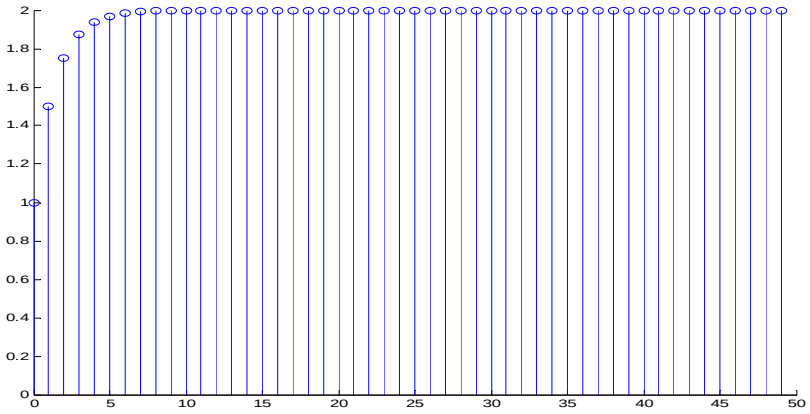


Figure 9

Simulating Difference Equations

We will investigate a system expressed as a difference equation:

$a_0*y[n] + a_1*y[n-1] + a_2*y[n-2] + ... a_N*y[n-N] = b_0*x[n] + b_1*x[n-1] + ...+ b_M*x[n-M]$

Notice that $N \geq M$.
This is a recursive equation. This means that for instance y[n] depends on the input signal x[n] and previous values of input and output signals.
Let us use matlab to display this system. We will use the filter command.
First we define the vectors a and b that represents the coefficients in the equation.
$a=[\ a_0\, a_1 \ldots a_N\ ]$ and $b=[\ b_0\, b_1 \ldots b_M]$
Then we apply the filter command. It has three arguments and can produce the output from the system with zero initial conditions: y=filter(b ,a ,x) or if we have initial conditions that are non-zero we use the alternative command with a fourth argument z_i: y=filter(b ,a ,x , z_i)

Example: plot the impulse response and step response from a system descibed by the following difference equation:
y[n] – 1.143*y[n-1] + 0.4128*y[n-2] = 0.0675*x[n] + 0.1349*x[n-1] + 0.675*x[n-2]

```
>> a=[1 -1.143 0.4128]
>> b=[0.0675 0.1349 0.675]

>> impz(b,a,30)          % Calculates and plots the impulse response, 30 samples.
```

See the plot in figure 10 !

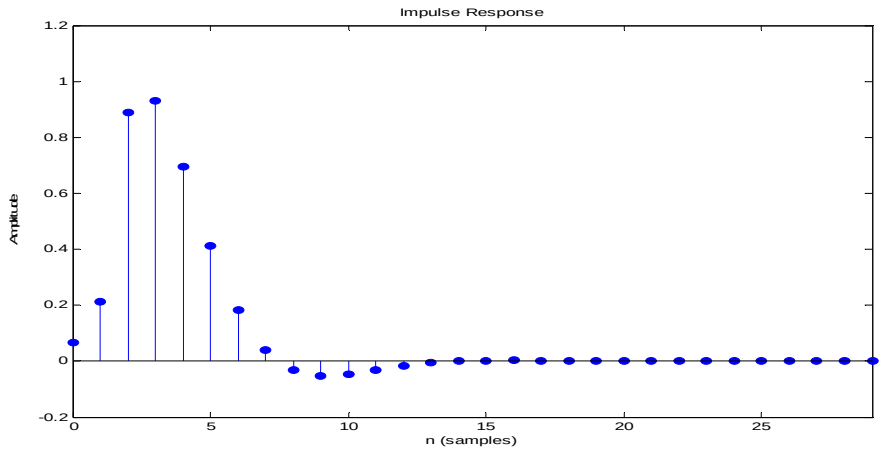


Figure 10

The step response can be executed with the command stepz.
>> stepz(b,a, 30) % calculates and plots the step response, 30 samples.

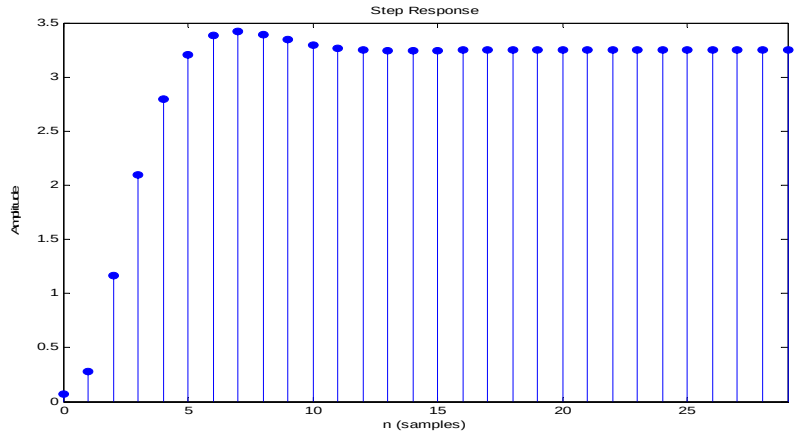


Figure 11

Suppose we don't have any input signal, only initial energy stored in the system.
We would like to see the natural response from the system. This can also be shown by the command filtic.
Assume the initial conditions to be: y[-1]=1 and y[-2]=2. The system is of degree 2.
Therefore we need two initial conditions in order to simulate the system.

```
>> a=[1 -1.143 0.4128]
>> b=[0.0675 0.1349 0.675]
>> x=zeros(1,50);       % creates a vector with 50 zero elements.
>> zi=filtic(b,a,[ 1, 2]);          % the third argument contains the initial conditions.
```

Notice that the initial conditions comes in the order [y[-1], y[-2], ...y[-N]].

```
>> y=filter(b,a,x,zi);              % calculates the natural response.
>> stem([0:49],y)                   % plots the first 50 samples in the natural response.
```

The plot can be seen in figure 12 !

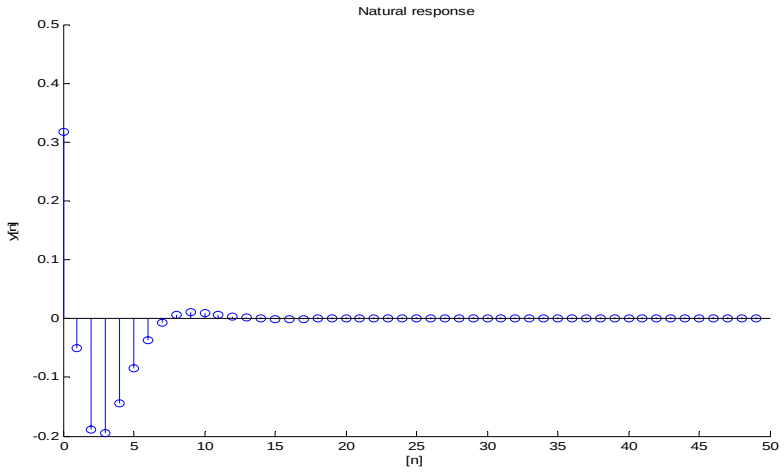


Figure 12

Example:

Find the output for the 50 first samples from the system in the previous example if the input signal is x[n]=sin(pi/10*n) ! Assume zero initial conditions !

```
>> x=sin(pi/10*[0:49]);      % creates the input vector x, contains 50 values.
>> y=filter(b,a,x);          % calculates 50 samples in the output vector.
>>  stem([0:49],y)           % gives the plot in figure 13.
```

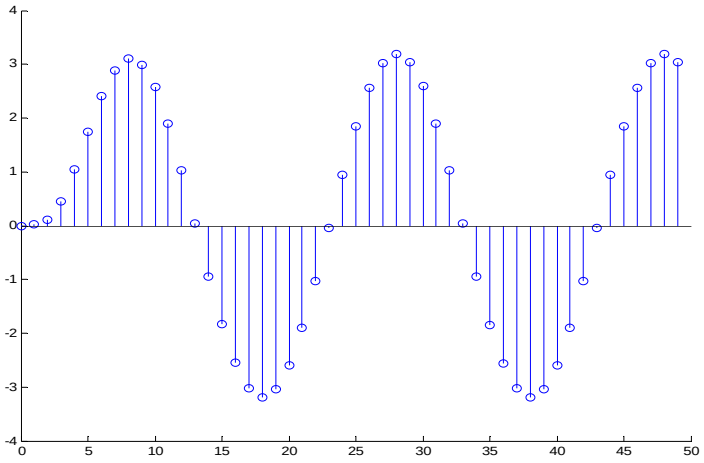


Figure 13

Exercises to be solved

The exercises should be handed in one m-file and sent to my emailadress.
Use the pause command in matlab to separate the assignments from each other.
These exercises can be handed in individually or as a group, but notice that the group must not be larger than 2. Can give bonus points on the written exam.
The m-files should be compressed and sent to me in zipped format.
Note that identical m-files are not allowed. It is considered as cheating.

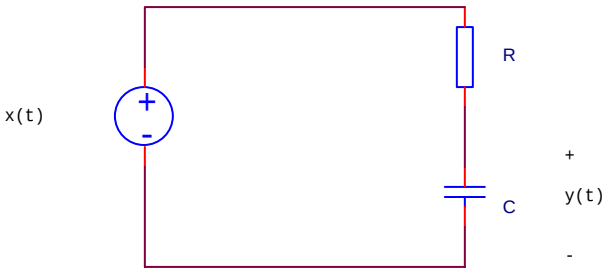
1) Create an m-file that produces the following signals. Use subplots or pause command to separate the signals from each other in time and plot window.
   a) Create a step function with a step amplitude 3 and with a step appearing after 10.5 seconds !
   b) Create a impulse function at time t=4.6 sec and a amplitude 3!
   c) Create a ramp function with a slope 3. The ramp starts after 2.5 seconds.

2) Decide the impulse- and step response from a system described by the following difference equation:
   y[n] – 0.85*y[n-1] = 0.25*x[n-1] + 0.675*x[n-2]
   Make a m-file that gives both the impulse- and step response for the first 25 samples. Are there any conclusions that can be drawn from the transient responses, regarding the stability and final values ?
   Assume zero initial conditions.

3) Use the same system as in the previous exercise., but assume we have another input signal: 3*exp(-2t)*sin(4*pi*t). Show the response from the system assuming zero initial conditions for the the first 5 seconds !

4) Use the same system as in exercise 2 but assume non-zero initial conditions, y[-1]=2 and y[-2]=0 . As input signal we use a step with amplitude 2. The step comes after 5 samples.
   The plot should at least show the first 25 samples !

5) Solve the following differential equation, with matlab. Use **dsolve** !
   R= 10kΩ, C= 100 µF and x(t)=1V is the input for t $\geq$ 0.



Make a plot of y versus t. Especially mark the time constant in the graph.

6) If we instead of considering the voltage over the capacitor to be the output we use the current in the circuit above . How will the solution look like in that case ? Use **dsolve** ! The input signal is still x(t).
Make a plot of y versus t, but now we let the current to be denoted y(t).
Especially mark the time constant in the graph.