EE 387 – Signal Processing

AUGUST 2015

Lab 1: Basic Signal Representation and Convolution in MATLAB

1. Introduction

Some of the very useful functions for signal processing include unit step, unit impulse and ramp signals. The first step of the lab would be to simulate signals in Matlab.

The second segment of the lab emphasizes how MATLAB can be used to represent continuous-time signals and to approximate continuous-time convolution.

2. Objectives

1. Learn Matlab basics to use basic signals.
2. Use MATLAB to perform convolution in time domain.

3. Procedure

PART 1: Basic Signal Representation in MATLAB

1. Write a Matlab program and necessary functions to generate the following signal:

$$y(t) = r(t+3) - 2r(t+1) + 3r(t) - u(t-3)$$

Then plot it and verify analytically that the obtained figure is correct.

```
clear all;

Ts=0.01; t= -5:Ts:5;
y1 = ramp(t,3,3);
y2 = ramp(t,-6,1);
y3 = ramp(t,3,0);
y4 = ustep(t,-3);
y = y1-2*y2+3*y3-y4;
plot(t,y,'k'); axis([-5 5 -1 7]); grid


function y = ramp(t,m,ad)

% t: length of time
% m: slope of the ramp function
% ad: advance (positive), delay (negative) factor
```

```
% Write your code

function y = ustep(t,ad)

% Write your code
```

2. For the damped sinusoidal signal $x(t) = 3e^{-t}\cos(4\pi t)$ write a MATLAB program to generate x(t) and its envelope, then plot.

PART 2: Time-Domain Convolution

Creating a rectangular pulse in MATLAB

Consider the rectangular pulse defined as:

```
x(t)  = 1   for -0.5 <=  t < 0.5 sec
      = 0    elsewhere
```

We would like to represent this continuous-time signal in MATLAB. However, digital computers can only operate on discrete samples of the signals. Therefore, we must sample the signal to make it discrete-time.

As a first step we must decide what sample frequency f_s we want to use. For our purpose, a sample frequency of f_s = 100 Hz is sufficient. (Are there any disadvantages if a high sampling frequency is used?) If a shaper edges are preferred you could set f_s = 1000 Hz.

Another important parameter is the sample period, which is just the inverse of the sampling frequency. Hence, sample period can be entered into MATLAB using:

```
T_s = 1/f_s;
```

Since MATLAB can only handle a finite number of samples at a time, it is suggested to assume a time period in the range -5<= t <= 5 seconds. Therefore, enter the following command to create the time axis in MATLAB as follows:

```
t = [-5:T_s:5];
```

Now we must generate our rectangular pulse. Although MATLAB has built in functions to generate a wide range of signals (such as cos, sin, and exp) it does not have a function to create the rectangular pulse. Thus, build your own rect.m function into your MATLAB program.

```
function x = rect(t)

%
% RECT rectangular pulse
%
```

```
% Usage: x = rect(t)
%
% This function takes in a vector t of sample instants and outputs the
% corresponding rectangular pulse contained in the function x
```

Now let's generate a rectangular pulse and plot it:

```
x1 = rect(t);
plot(t,x1);
```

You will observe that y-axis only shows values from 0 to 1, hence the pulse cannot be seen clearly. One solution would be to change the field of view of the plot by using the axis command:

```
axis( [-2 2 -1 2]);
```

Now label to axis and add a title to your plot. You can do that as follows:

```
xlabel( 'time (sec)' ) ;
ylabel( 'x_1(t)' ) ;
title ('Plot 1: A rectangular pulse');
```

Elementary signal operations

Now let's perform some elementary signal operations, such as time-delay, time-scaling, and time-reversal.

First let's create and plot the time-delayed signal, $x_2(t) = rect(t-1)$

```
x2 = rect(t-1);
plot(t,x2);
axis( [-2 2 -1 2]);
```

Now let's try to make the time-scaled signal $x_3(t) = rect(t/2)$

```
x3 = rect(t/2);
plot(t,x3);
axis( [-2 2 -1 2]);
```

Also create the following signal $x_4(t) = rect(t)+(1/2)rect(t-1)$ and perform a time reversal as: $x_5(t) = x_4(-t) = rect(-t)+(1/2)rect(-t-1)$.

Finally you can also create the signal: $x_6(t) = x_4(1-t) = rect(1-t)+(1/2)rect(-t)$.

Now use the MATLAB subplot command to plot all these signals in a single plot. (you might want to write something like below)

```
subplot(3,2,1)
plot(t,x1)
```

```
axis( [-2 2 -1 2]);
xlabel( 'time (sec)' )
ylabel('x_1(t) = rect(t)')

subplot(3,2,3)
plot(t,x2)
axis( [-2 2 -1 2]);
xlabel( 'time (sec)' )
ylabel('x_2(t) = x_1(t-1)')
```

Convolution

We can use MATLAB to approximate continuous-time convolution by using the conv function. As an example, the convolution of two rectangular pulses (using the same time axis t and pulse x1 from part 1) would be:

```
y = conv(x1,x1);
```

Now try plotting y by typing the following code:

```
close all;
plot ( t, y);
```

Notice the MATLAB generated error message because the size of y is now different than the size of t. To check the size of y and t type:

```
length(y)
length(t)
```

Notice that length(y) = 2*length(t)-1. Therefore, we need to create a separate time axis for the signal y as:

```
t_y = -10:T_s:10;
```

Now we can plot:

```
plot( t_y, y)
```

Do you notice any error with the plot? As expected, we got a triangular pulse. However, the peak of this triangle is f_s = 1000, but it should have been at one. This is because the way the conv function is performed in MATLAB. To correct type:

```
y1 = T_s*conv(x1,x1);
```

and plot

```
plot(t_y, y1);
axis( [-2 2 -1 2] ) ;
xlabel( 'time (sec)');
ylabel('y_1(t)');
```

```
title('Figure : y_1(t) = x_1(t)*x_1(t)');
```

Exercise

1. Perform convolution on discrete time signals x(n) and h(n), i.e., y(n) = x(n)*h(n) using MATLAB. For each set of signals, plot x(n), h(n) and y(n) as subplots in the same figure.

   - x(n) = { 1,2,4 }, h(n) = {1,1,1,1,1}
   - x(n) = { 1,2,3,4,5 }, h(n) = {1}
   - x(n) = h(n) ={ 1,2,0,2,1}

2. Assume a system with the following impulse response:

```
h(n)    = (0.5)^n       for 0<= n < 4
        = 0             elsewhere
```

Determine the input x (n) that will generate the output sequence y(n) = {1, 2, 2.5, 3, 3, 3, 2, 1,0...}.
Plot h(n), y (n) and x (n) in one figure.