Laboratory 2: Discrete-Time Signals in the Time Domain

1. Objectives

- Learn how to generate and display discrete signals
  - using "stem" plots
  - using signals that do not begin at n=1 by using an index vector
- Learn how to construct common signals in Matlab, including
  - impulse
  - step
  - complex exponential
- Increase proficiency in Matlab programming, in preparation for later labs

2. Introduction

Digital Signal Processing (DSP) is a misnomer; it usually means *Discrete* Signal Processing in which an analog signal is sampled at some sampling frequency F_s. The signal can have any value (as opposed to a true digital signal which is quantized to typically a byte sized value ranging from 0 to 255), and since it is sampled it may be thought of as either a sequence of numbers or as a vector. In general, DSP is concerned with the processing of this type of sampled input signal x[n] to create an output signal y[n] with certain desirable properties. It is thus important to learn first how to generate some basic discrete-time signals in Matlab and perform elementary operations on them.

Many applications of DSP deal with "pure" numbers not sampled from an analog process. An example of such an application is the prediction of tomorrow's stock market prices given the sequence x[n] of the previous year's daily stock prices. Far more common, however, is the use of DSP to control or analyze a real-world analog process that is sampled, such as the use of a digital comb filter to sharpen the analog-transmitted signal displayed on a television set. This lab will introduce another such application: the construction of a digital echo filter to make a recording of your voice sound fuller.

3. Laboratory

1. Creating stem plots of discrete signals with an index vector

A discrete signal x[n] is not at all the same as a continuous signal x(t).
x(t) is defined for all values of t; it is a continuous graph. For instance, x(t) is defined at t=0.1, t=0.2, and t=0.2001. To graph x(t) then we must construct a continuous plot like the one in Figure 1. Try to exactly represent this figure as a sequence of numbers rather than a smooth curve; you can't! The best you can do is approximate it using a lot of closely-spaced samples.

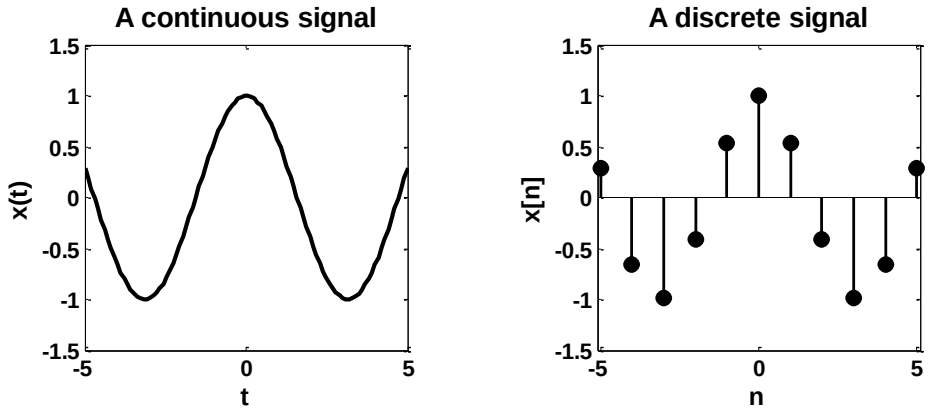

Figure 1: A continuous signal x(t) on the left and a discrete signal x[n] on the right. The continuous signal is defined for all values of t; the discrete signal is defined only for integer values of n, which is graphically represented by the *stem plot* on the right. Both are graphs of the cos function; x(t) = cos (t) and x[n] = cos [n].

x[n] is a mapping of a sample number n to a value x[n]. For example, when n = -2, -1, 0, 1, and 2, x[n] may be respectively 1, 3, 5, 3, 1. The index n may only be an integer; the value x[1.5] is NOT zero, but instead is undefined. The fact that the signal is defined only at integer values of n is graphically shown by using a *stem plot*, as shown to the right in Figure 1. Don't get the stem plot confused with a collection of time-shifted impulses. Although impulse functions look similar (except they have an arrow rather than a stem), impulse functions are continuous in nature and infinite in height. A continuous-time signal x(t) made of time-shifted impulses (Figure 2) looks similar to the right panel of Figure 1 except with arrows, but for this signal x(0.5) is defined and is exactly 0, rather than x[0.5] which is undefined, not zero. Further, in Figure 2, x(0) is infinite and the integral of x(t) for t ranging from 0- to 0+ is 1; in Figure 1, x[0] = 1, and integration is not possible on discrete signals.

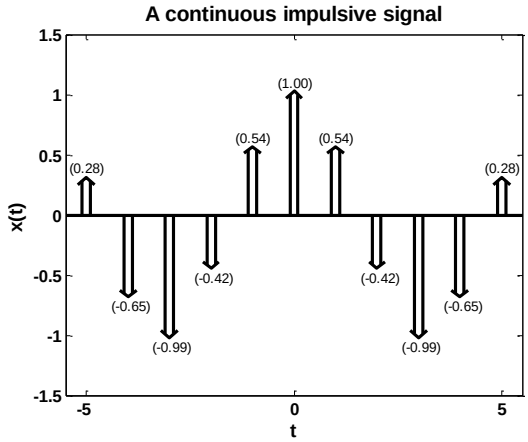

Figure 2: A continuous signal x(t) formed by a series of shifted and scaled impulse functions whose area maps the amplitude of the continuous signal in the left panel of Figure 1. Although this looks similar to the discrete signal in the right panel of Figure 1, it is still a continuous signal and defined for all values of t, not just integers.

Often a discrete time signal is represented as a sequence of numbers. This works well, except that the index information is lost. For instance, to represent the information in Figure 1 as x=[.28 -0.65 -0.99 -0.42 0.54 1.0 0.54 -0.42 -0.99 -0.65 0.28], while correct, does not show if these are the values as n ranges from 1 to 11, -5 to 5, or 10000 to 10010. To do this, a second vector, called an index vector and abbreviated n, can be associated with the signal vector. In this case n =

[-5 -4 -3 -2 -1 0 1 2 3 4 5]. The index vector will always need to be the same length as the signal vector. Random note: the signal vector x and the index vector n can be either column or row vectors; it makes no difference and you will encounter both. To generate the plot on the right panel of Figure 1, type

```
n = -5:5;
x = cos(n);
stem(n, x)
```

Just typing

```
stem(x)
```

creates a stem plot that assumes the first index value is 1; that is, it creates a default index vector as follows: n = 1:length(x).

Problem 1
You will frequently want to create a stem plot of a known signal x that begins at a known index. Unfortunately, the syntax of the stem command `stem(n,x)` requires you to create a vector n of the same size as your x vector. Write a m-file called stem1 that takes a signal vector x and a scalar nStart, and then creates the proper index vector n and then sends it to stem to plot. Test it by using it to plot the signal x=[1 3 5 3 1] using nStart = -2 and include the resulting plot.
Hint: the m-file should look like:

```
function stem1(x,nStart)
% stem1 creates a stem plot from a signal vector x and a scalar
% nStart indicating the index of the first value of x
% Usage: stem1(x, nStart)
% For instance, stem1([1 3 5 3 1], -2) plots a signal symmetric about 0

n = ** create the index vector here **
stem(n, x)
```

A unit "boxcar" signal is commonly-encountered in DSP. It is zero everywhere but from n=-1 to 1, where it is 1. To graph this signal over the range $-1 \le n \le 1$, we enter in Matlab the following commands. The result is shown in the left panel of Figure 3.

```
>> n = [-1 0 1];
>> x = [1 1 1];
>> stem(n,x)
```

This plot is confusing; for the small range of n plotted we can't tell if it is a boxcar or if the signal is simply always 1. A better plot would include a larger range of n, such as from -5 to 5, as shown with the commands below and graphed on the right panel of Figure 3.

```
>> n = [-5 -4 -3 -2 -1 0 1 2 3 4 5];
>> x = [0 0 0 0 1 1 1 0 0 0 0];
>> stem(n, x)
```

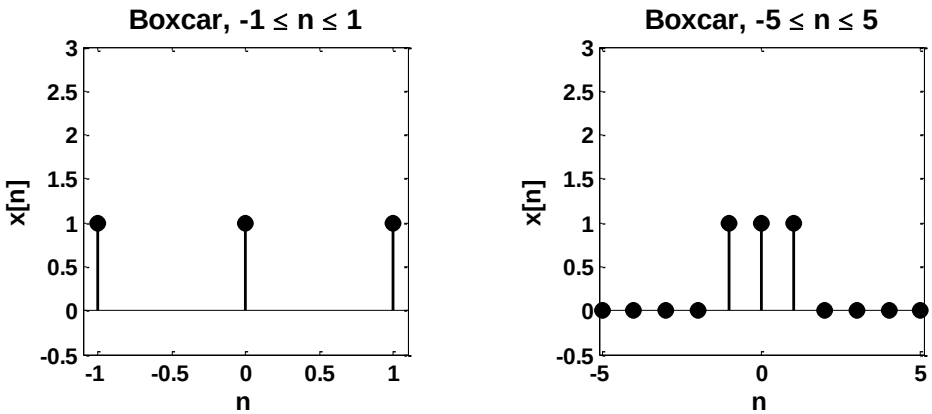

Figure 3: a unit boxcar signal plotted using two different ranges of n. On the left, -1≤n≤1, and on the right, -5≤n≤5. The range on the right is much clearer to interpret.

Advanced Matlab trick: For the curious, the special symbol in the title ≤ is created by inserting a TEX command in the title string, like this: title('Boxcar, -1 \leq n \leq 1'). Do a search under Matlab help for TEX for full instructions.

The plot in the right panel of Figure 3 is much clearer, but the Matlab syntax of manually typing in the zeros would become awkward if the range of n was much larger. A plot over the range -50 ≤ n ≤ 50, would require typing 49 zeros, three ones, and another 49 zeros just to create the x vector. A much more efficient way is to use Matlab's matrix colon operator and `zeros` and `ones` function as follows:

```
>> n = -50:50;
>> x = [zeros(1,49)  ones(1,3)   zeros(1,49)];
>> stem(n, x);
```

The result is shown in Figure 4.

Advanced Matlab trick: I modified the plots in Figure 3 slightly because Matlab's default axis scaling put the stem plot's vertical scale between 0 and 1 which made the plot much harder to read (try it and you'll see). I forced Matlab to graph using an upper vertical limit of 3 using the axis command; type `help axis` for details. For example, after the stem command to create Figure 3, I typed `axis([-1.1 1.1 -0.5 3])` to set the minimum and maximum limits of x and y. Figure 4 used `axis([-50 50 -0.5 10])`.

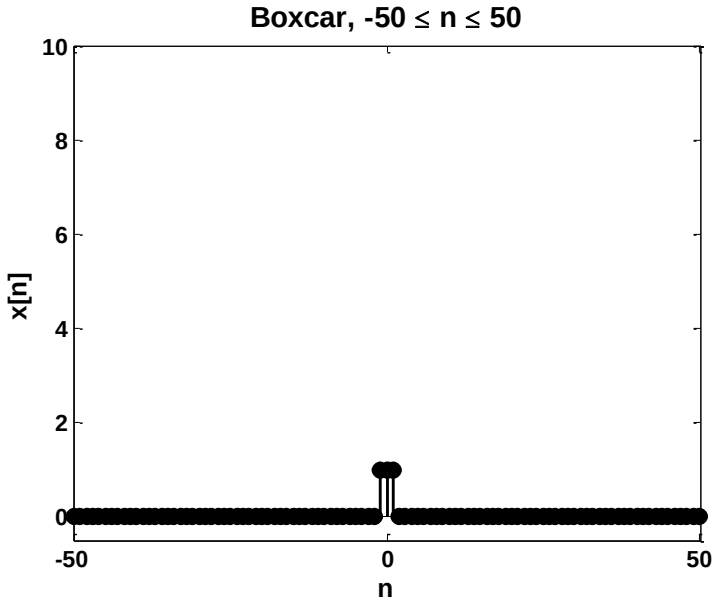

Figure 4: The same unit boxcar signal as shown in Figure 3 plotted from n=-50 to +50, using Matlab's colon operator and ones and zeros functions. With 101 data points, the stems overlap.

Problem 2
Plot the signal

$$x[n] = \begin{cases} 2n, & -3 \le n \le 3, \\ 0, & \text{otherwise} \end{cases}$$

over the range $-10 \le n \le 10$. Include your code to create the plot.

2. Common types of discrete signals

We already examined the boxcar signal. There are a handful of other common signals that you will encounter in DSP, including the unit sample, the step, complex exponentials, and periodic signals.

a) Unit sample

The discrete unit sample's notation is typeset as δ[n], which in Matlab we can write as d[n]. It is defined to be zero everywhere except at n=0, where it is 1. Notice how similar this is to the continuous-time concept of the unit impulse which is zero everywhere except at t=0 where it is infinite, with an area of 1. To make things confusing, a discrete sample function is also often called a discrete impulse function. A discrete unit sample of length N can be generated in Matlab using the following commands for 0≤n≤N-1:

```
>> d=[1 zeros(1,N-1)];
>> n = 0:N-1;
```

One could create a function that generates both the sample sequence and its index, given a starting index N_1 and final index N_2, as shown in the following m-file:

```
function [n, s] = MakeSample(N1,N2)
% MakeSample creates a discrete sample vector
%    for N1 <= n <= N2
% [n,s] = MakeSample(N1,N2)
% For instance try [n,x] = MakeSample(-5,5), and then
%    stem(n,x)

% error check the input arguments
if N2<=N1
    error('N2 must be larger than N1')
end

% create the index vector
n = N1:N2;

% create the sample vector
s = zeros(1,N2-N1+1);  % the output is mostly zeros
whereIsnZero = find(n==0); % at this location n is 0
s(whereIsnZero) = 1; % make vector s 1 where n is 0
```

To make the above code generate a unit sample vector over the range $-5 \le n \le 5$, do the following:

```
[n,x] = MakeSample(-5,5);
stem(n,x)
axis([-5 5 -0.5 2])   % this just scales the stem plot better
```

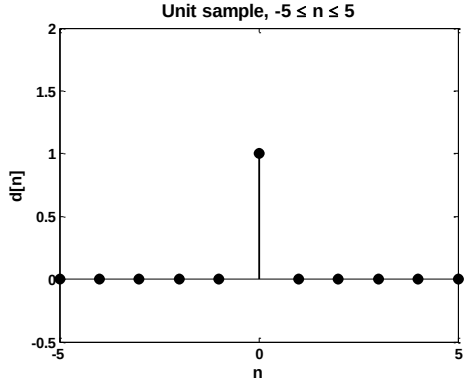

Figure 5: A unit sample $\delta[n]$ plotted over the range $-5 \le n \le 5$.

The `find(v==s)` command is new; it returns all locations in vector v that hold the scalar s. For instance, if v = [-3 -2 -1 0 1 1] and s = 0, `find(v==s)` returns 4 since v(4) is 0. If s = 1 it would return a vector [5 6], since v(5) and v(6) are both 1.

The time shifted unit sample $\delta[n-n_0]$ is another very common signal, defined as 0 everywhere except where $n = n_0$ where it is 1. Figure 6 shows $\delta[n-2]$ plotted over the range $-5 \le n \le 5$; it looks similar to Figure 5 shifted to the right by 2.

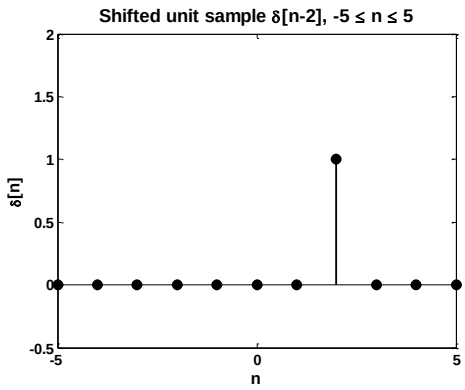

Figure 6: A time-shifted unit sample δ[n-2] plotted over the range $-5 \le n \le 5$.

Problem 3
Using MakeSample.m as a template, create an m-file that can produce the signal vector and index vector to create the plot shown in Figure 6, given N_1, N_2, and the timeshift n_0. Specifically, the first line of the m-file is given below:

```
function [n, s] = MakeShiftedSample(N1,N2,n0)
```

To test your function, execute the following code:

```
[n,x] = MakeShiftedSample(-5,5,2);
stem(n,x)
axis([-5 5 -0.5 2])
```

Include in your report both your code and the test stem plot.

b) Step Function u[n]

A discrete step signal u[n] is defined as

$$u[n] = \begin{cases} 0, & n < 0 \\ 1, & n \ge 0 \end{cases}$$

Use the Matlab function zeros and ones to graph this. As an example, u[n] can be graphed over $-20 \le n \le 20$ using the following commands and is shown in Figure 7.

```
u = [zeros(1,20) ones(1,21)];
n = [-20:20];
stem(n,u)
axis([-20 20 -0.5 2])
```

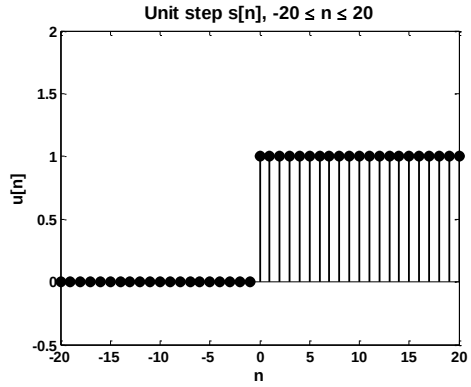

Figure 7: A unit step function.

A step signal time delayed by n_0, can be calculated as

$$u[n-n_0]=\begin{cases}0, & n<n_0\\ 1, & n\geq n_0\end{cases}$$

You could use the `find` command as you did in Problem 3 to design a function that plotted a delayed unit step.

Problem 4
Plot the signal u[n-10] for -5 ≤ n ≤ 25 and display the code you used to plot it.

c) **Complex exponential**

Recall Euler's Identity from Signals and Systems that $e^{j\theta}=\cos(\theta)+j\sin(\theta)$

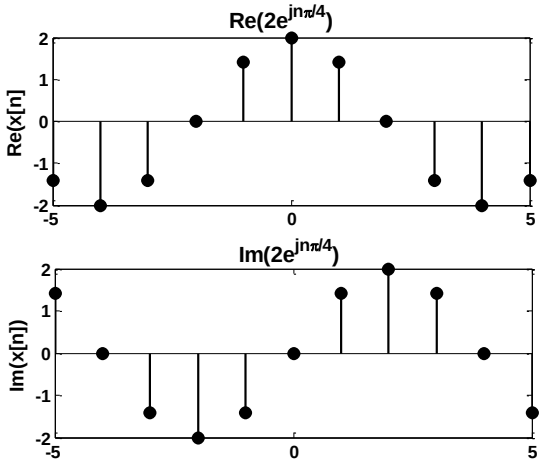

Figure 8: A complex exponential signal requires two separate plots, either displaying the real and imaginary components, or else the magnitude and angle.

A complex exponential signal such as $x[n] = 2e^{n\pi/4}$ will therefore be complex-valued. To display this signal we may either choose to plot the real part and the imaginary part as shown in Figure 8, or we may choose to plot the magnitude and the phase.

Matlab's subplot command was used to create Figure 8. subplot(rows, cols, active) takes three arguments: the number of rows of subplots, the number of columns of subplots, and the active subplot. Figure 8 used subplot(2,1,1), to create a single column of two rows of plots and to activate the top plot. Then the plotting commands for the top graph were given, and then subplot(2,1,2) to activate the lower plot region, followed by the lower plotting commands. In detail, the commands used are

```
n = -5:5;   % create the index vector
x = 2*exp(j*pi*n/4); % create the signal vector

% top plot
subplot(2,1,1)     % must come BEFORE the plotting commands
stem(n,real(x));
title('Re(2e^{jn\pi/4})')   % fancy TEX labels in the title
xlabel('n');  ylabel('Re(x[n])')

% bottom plot
subplot(2,1,2)
stem(n,imag(x));
title('Im(2e^{jn\pi/4})')
xlabel('n');  ylabel('Im(x[n])')
```

Problem 5

Figure 8 is the graph of $x[n] = 2e^{jn\pi/4}$ in rectangular notation, since it separates the real and imaginary components much like in a Cartesian coordinate system. Use subplot to create the graph in polar notation, that is the magnitude of x[n] in the top plot and the phase angle in degrees in the bottom, over the same region of n, from $-5 \le n \le 5$. Do not forget that angle(z) returns the phase angle of the complex variable z in radians. As usual, include your plotting commands (yet another reason to make each problem solution a script or m-file!) How would you change the mathematical definition of x[n] to decrease the magnitude? To increase the slope of the phase curve? Why does the phase seem to suddenly change at n = -5 and n=+5?

d) **Periodic signals**

Many signals x[n] are periodic in nature, such as shown in the top panel of Figure 9. You learned in EE330 that periodic signals can be constructed using an aperiodic generator function $x_g[n]$ (shown in the bottom panel of Figure 9) that is zero everywhere except from n=0 to N. An infinite number of appropriately time-shifted generator functions can then be summed to create the periodic signal. Specifically, a periodic signal x[n] with period T is constructed from a generator function $x_g[n]$ as follows:

$$x[n] = \sum_{k=-\infty}^{\infty} x_g[n - kT]$$

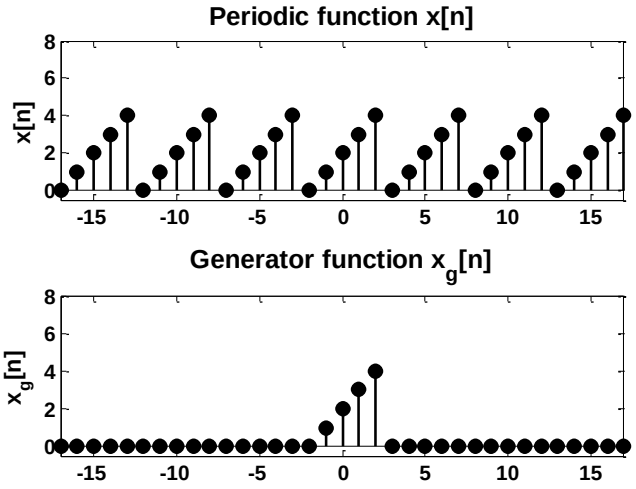

Figure 9: A periodic function x[n] can be created from summing an infinite number of time-shifted generator functions $x_g[n]$.

One way to do this in the Matlab environment is to create the generator function $x_g[n]$ first. For Figure 9, `xg = [0 1 2 3 4]`. Then duplicate it either manually, for instance

```
x = [xg xg xg xg xg xg xg]
```

or within a loop, for instance

```
x = [];   % begin with an empty vector x
for k=1:7
% each time it loops, concatenate xg in front of the old x
      x = [xg x];
end
```

Lastly, decide what the indexing vector should be. The code above produces odd length of x, so I chose to make the index vector symmetric around n=0 with the following commands:

```
N=length(x);
N1 = -(N-1)/2;  N2 = (N-1)/2;
n = N1:N2;
```

A different indexing time-shifts the periodic function. For instance, to generate Figure 10 use the indexing

```
n = 0:N-1;
```

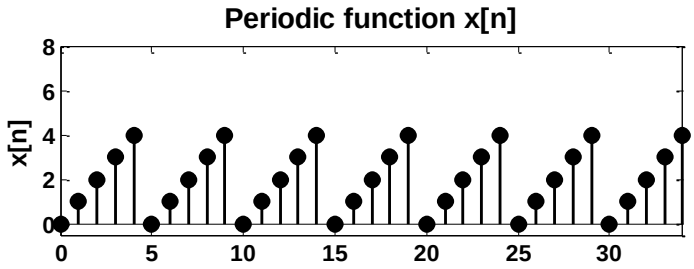

Figure 10: The same generator function $x_g[n]$ as in Figure 9, but with a different index vector produces a time-shifted version of the same periodic signal.

Problem 6

Figures 9 and 10 are called sawtooth waves. A triangle wave is related, but uses a symmetric generator function to create equilateral, rather than right, triangles. Create a plot of a triangle wave using 6 cycles of the generator function

```
xg=[ -1 -0.5 0 0.5 1 0.5 0 -0.5];
```

Have the index vector start at n=-5 (i.e. x[-5]=-1, x[-4]=-0.5, x[-3]=0, etc.) and increment as far as necessary to plot six full cycles of the generator function. Include both the code to make the plot (which may by an m-file or a script since it neither takes nor returns arguments, or code cut and pasted from the command line) and the plot itself.

Hint: Unless you are really proficient in Matlab, you'll find that you save time by writing a script or function whenever you need to do operations that take multiple commands. If you attempt to do it all through the command window, and then realize that you made a mistake on the 5th command, you'll have to re-issue the first 4 commands before you can fix the problem on the 5th. With a script or m-file, just make the one-line change and then re-run your program. As a bonus, you then have a neat record of your code in one file.

4. Matlab Commands Used in Laboratory 2

Plotting/display commands

stem(index, x)	creates a stem plot of x plotted against the index vector
axis([xmin,xmax,ymin,ymax])	Forces the axis of a plot to these values
subplot(N_1,N_2, n)	Creates a system of N_1 rows and N_2 columns of subplots, and sets the n^{th} one (counted in row-major ordering) to be active and receive subsequent plotting commands.

Matlab vector commands

find(v,2)	this returns all the indices where vector v equals 2. For instance, if v=[2 2 3 2 1], this will return [1 2 4].
zeros(n1,n2)	creates an n1 row by n2 column vector of zeros
ones(n1,n2)	creates an n1 row by n2 column vector of ones