SIGNALS AND SYSTEMS LABORATORY 3:
Construction of Signals in MATLAB

INTRODUCTION

Signals are functions of time, denoted $x(t)$. For simulation, with computers and digital signal processing hardware, one must represent a signal as a vector. We limit the representation in two different ways: x is sampled and then only a finite number of samples is kept. The representation is the array of samples of the signal, spaced t_s seconds apart (this is the *sampling period*):

$$[x(t_0), x(t_0 + t_s), ..., x(t_0 + (L-1)t_s)].$$

Thus signals are represented in MATLAB by row-vectors. There are L samples in this row-vector, and the vector has *dimension L*. The purpose of this laboratory exercise is to gain experience in using MATLAB commands to construct row vectors that represent interesting signals. As you read this, reproduce all MATLAB interactions by typing the commands following the prompt [»]. To help you identify these, the commands and responses are indented.

MATLAB provides a 'help' tool that serves as an abbreviated manual. Typing 'help' followed by the command or function name usually provides enough information to use the function. In the case of functions, 'help' will list the comments immediately following the function declaration. Thus when you write your own functions, you should include enough information in the form of comments to be able to use the function. See what happens when you type the following.

```
»help who
»help plot
```

SIGNAL CONSTRUCTION

The primitive way to construct a row-vector is simply to type it all in:

```
»x=[1 3 5 4 6 2]
x =  1    3    5    4    6    2
```

You can separate the values with either spaces or commas. For long signals, this is not feasible. Fortunately, the MATLAB commands and functions allow for a rich variety of constructions that can be typed quickly.

CONSTANT (DC) SIGNALS

For *constant* signals $x(t) = c$, one uses the MATLAB function 'ones', which returns an array of ones. (Type »help ones while in the MATLAB command mode.) To get a row vector, the row dimension must be $M = 1$, and the column dimension must be $N = L$, the number of samples. Try this:

```
»x=ones(1,3)
x =  1    1    1
```

If you get the dimensions in the wrong order, you will get a column-vector:

```
»x=ones(3,1)
x =
     1
     1
     1
```

Without a trailing semicolon, MATLAB will print the results. However, interesting signals are usually very long, and a printout can be frustrating. To suppress the printout, append a semicolon to the command line as follows:

```
»x=ones(1,2000);
```

Of course, you will forget to do this, and then wish you hadn't. Try typing a control-C to stop the flood of output.

To construct a constant signal with value other than one, use the command x=c*ones(1,L). To construct a zero signal $x(t) = 0$, use the command x=zeros(1,L), which is equivalent to x=0*ones(1,L).

RAMP SIGNALS

The signal $x(t) = t$ has the representation [t0, t0+ts, ...,t0+(L-1)*ts] on the interval from $t = t0$ to $t = t1$. The integer L is the length of the row vector and is chosen so that

$$t_1 \le t_0 + (L-1)\cdot t_s < t_1 + t_s \, .$$

The MATLAB expression for such arrays is simply [t0:ts:t1]. You may think of this as a *sampling vector*. This reads "start at t1 and generate numbers spaced by ts until t1 is reached but not exceeded". The arguments are separated by colons. Try

```
»ts=.3;t0=1;t1=2;t=[t0:ts:t1]
t = 1.0000    1.3000    1.6000    1.9000
```

Note that the last value here is not t1 = 2 because 2 is not 1 plus an integer number of sampling periods. This construct is most often used for index arrays, which are integer valued. In this case you can suppress the middle parameter ts and simply write [t0:t1]:

```
»k=[2:6]

k = 2     3     4     5     6
```

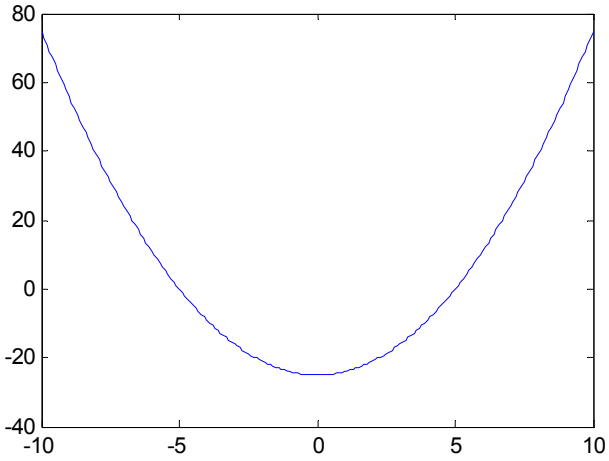
The parameters need not be integers, however. See what happens if you type [1.1:pi].

SIGNAL ALGEBRA

When they have the same length, signals can be added, element-by-element, using the usual symbol (+). Multiplication by scalars uses (*) and multiplication of two signals, element-by-element, uses (.*). This is read "dot times". The use of the dot, or period, distinguishes element-by-element multiplication from matrix multiplication, which is the default in MATLAB. Try the following;

```
»ts=.1;t=[-10:ts:10];
»x=t.*t-25;
»plot(t,x)
```

We have constructed samples of the function $x(t) = t^2 - 25$. This is the polynomial whose roots are at 5 and -5. This example shows that MATLAB treats a scalar 25 subtracted from a vector t.*t as a scalar subtracted from each element of the vector. The plot should look like this:



In summary, if x and y are row-vectors, and a and b are scalars, then

(1) The MATLAB expression a*x+b will be a row-vector having components $a \cdot x[k] + b$.

(2) The MATLAB expression a*x+b*y will not be computed unless x and y have the same size. If x and y have the same number of elements, the vector will be computed and have the values $a \cdot x[k] + b \cdot y[k]$.

(3) To construct a signal z having components $z[k] = x[k] \cdot y[k]$ (assuming that x and y have the same size) you must use the '.*' operator:

```
»z=x.*y % read "x dot times y"
```

The *dot* is crucial for component-by-component operations.

WHITE GAUSSIAN NOISE SIGNALS

There are many MATLAB functions for specialized signals. The one for constructing a vector of independent, mean zero, variance one, Gaussian arrays is the random number generator called 'randn'. Try

```
»x=randn(1,5)
x = 1.1650    0.6268    0.0751    0.3516   -0.6965
```

To produce a length L white Gaussian noise signal with variance σ^2, use the command x=sigma*randn(1,L). (There is another random number generator called 'rand' which produces uniformly distributed random variables. Be sure to use 'randn'.)

RANDOM BANDPASS SIGNALS

Random signals that are essentially band-limited in frequency may be obtained by passing white noise through a bandpass filter. These operations are easily achieved using common MATLAB functions. On the web page under 'Functions for Lab 3', you will find the m-file 'bandsig.m'. Type:

```
»help bandsig

 [y,fs]=bandsig(fc1,fc2,L)
 create a random bandpass signal with frequencies fc1 to fc2 .
 Sampling frequency is fs=6000 Hz, and signal duration is L/fs seconds.
```

We will experiment with this function in the formal assignment. You can list the function using

```
»type bandsig.
```

MATHEMATICAL FUNCTIONS (LIKE SINUSIODS)

Using the construction t=[t0:ts:t1] to compute a common time base, mathematical signals which are functions of time can be constructed by simply applying the MATLAB function. Try the following (you don't need to type the comments):

```
»fs=1000; % the sampling frequency is one kilohertz
»ts=1/fs; % establish the sampling period
»f0=5; % we want a frequency of 5 Hz
»t=[0:ts:1]; % construct the time base vector
»x=cos(2*pi*f0*t); % build a row vector having the same size as t
»y=x.*x-1/2; % y is a signal having the same time base as x
»plot(t,x,'r',t,y,'g') % plot x and y versus t
```

CONCATENATION

If x and y are two row vectors, then [x,y] is their concatenation. Try

```
»delta=[1,zeros(1,6)]
delta =  1     0     0     0     0     0     0
```

This is a short representation of the *unit-pulse* signal $\delta[n]$.

SUBARRAYS

In MATLAB, the index values of a vector are integers, beginning with one. One can recover elements of the vector by using scalar or vector indices:

```
»x=[-10 2 4 -8 -7 30]
x = -10     2     4    -8    -7    30
»x(1)
ans = -10
»x([3,5])
ans =  4    -7
```

The name 'ans' is used by MATLAB to identify objects that have been computed but not given a name.

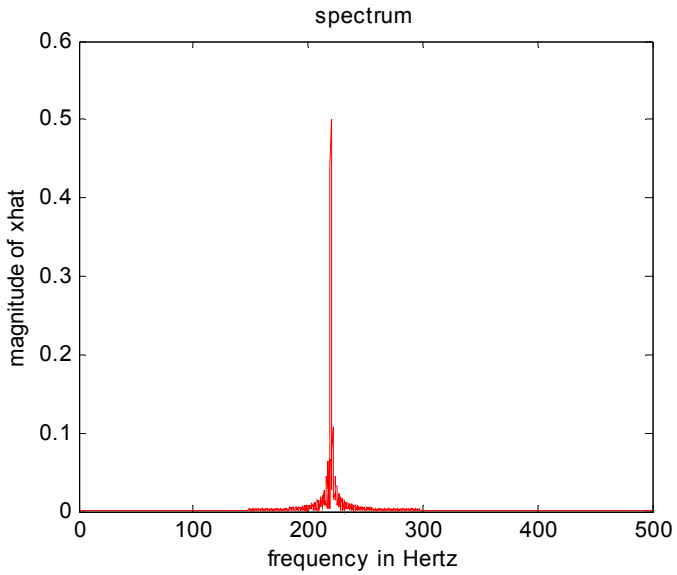
PLOTTING THE SPECTRUM OF A SIGNAL

The *spectrum* of a signal measures its frequency content. This is a good description of how we hear an audio signal, since our human sensors consist of tiny *tuning forks*, which resonate in different narrow bands like bandpass filters, and then communicate this information to the brain. In electronic hardware, frequency content is computed with a *Fourier Transform*, which produces a complex valued function. We shall study Fourier or frequency representations of various kinds of signals throughout this course. For row vectors, one can compute a row vector transform by using the *fast Fourier transform* or FFT, which approximates the Fourier Transform. We shall plot only the magnitude of the complex valued transform. (One ear cannot detect phase information. Two ears can detect only phase differences.) To look at the *magnitude spectrum,* it is helpful to create a function. For this lab, we have created the function 'spectrm.m', not to be confused with the built-in MATLAB function 'spectrum'. Type:

»type spectrm

to see the function.

This m-file computes the Fourier transform and plots the magnitude against a frequency scale from zero to $f_s/2$, which is one half the sampling frequency. The plot may appear or it may occupy a hidden window, in which case you will need to look for it. To test the 'spectrm.m' tool, run the m-file: 'spectrm_demo.m' from the 'Demos For Lab 3'. The result should look like this:



The commenting style used in this example of an m-file is recommended. If you use the 'help' command and type in the name of this function, you will get the comments following the function declaration. This information is the bare minimum to remember what the function does, and what its parameters are. If you build your own library of MATLAB tools, you should include such comments under the function statement. Type »help spectrm. The tools that come with MATLAB use the same device. Try »help fft.

SOUND

On computers which have hardware audio generation supported by MATLAB, one can play the signal as an audio source. Most of the computers in the Design Studio, Engr B203, have sound cards, but you may need to bring your own headphones to listen to the output. Find a computer that has a working sound card, and download and run the demo 'pure_tone_A.m' from the class web page under 'Demos for Lab 3'. This demo creates a pure tone of 440 Hz and plays it for one second. To hear this tone again, type

»sound(x,fs)

Here the scalar parameter fs is the assumed sampling rate. Thus the time interval between samples is ts=1/fs. The duration in seconds of the sound output will be T=L*ts=L/fs, where fs is the sampling frequency, and L is the length of the signal. In this demo, L=fs, so the duration is one second. The maximum usable frequency will be the *half sampling frequency* $f_s/2$. In commercial AM quality audio, the maximum frequency is 5 kHz, and thus f_s = 10kHz.

(NB: In music, a sinusoid with frequency 440 Hz is known as the tone A.)

Assignment:

You will need the following signals for this assignment. Construct them (you do not need to enter the comments).

```
»[ylow,fs]=bandsig(0,500,6000);
»yhigh=bandsig(2500,3000,6000);
»fc=261.2 % frequency of the note C
»fe=329.6 % frequency of the note E
»fg=392 % frequency of the note G
»yc=bandsig(.98*fc,1.02*fc,6000); %narrowband signal about fc
»ye=bandsig(.98*fe,1.02*fe,6000);
»yg=bandsig(.98*fg,1.02*fg,6000);
»phrase=[yc,yg,ye,yc]; % concatenation of C,E,G,C
»chord=yc+ye+yg;
```

Dowload the file 'plotsig.m' from the class web page, under 'Functions for Lab 3', and examine it via the command

```
»type plotsig
```

You should be able to explain what each line does. The purpose of this tool is to display the signal power spectrum, and a part of the time signal on two panels of a graph, and to generate sound. For example, enter the command

```
»plotsig(yg,fs)
```

This will display the signal in two panels, and create sound. If you want to hear the sound again, type

```
»sound(yg,fs)
```

as many times as you like (or use the up arrow to repeat). The tool 'plotsig.m' displays the entire phrase of all the notes. To see just one signal, use the command

```
»figure,plot(yg)
```

(The first part of this command creates a new figure window.)

1. Run through the whole set (ylow, yhigh, yc, ye, yg, phrase, chord) of signals that you have created. Sketch the time and frequency plots from the output of 'plotsig.m', and comment on the significance of each. You should be able to identify which signal you have from its spectrum except that you might not be able to distinguish between the musical phrase and the chord. Why not?

2. What happens if the sampling frequency parameter is changed? Try the following and explain the results.

   ```
   »plotsig(phrase,2*fs)
   ```

3. Download the function 'modulate.m' from the class web page, under 'Functions for Lab 3', and try this modification:

   ```
   »ygm=modulate(yg,fs,50);
   »sound([yg,ygm],fs)
   ```

   and explain what you hear.

4. Run the demo 'pure_tone_C.m' from the web page under 'Demos for Lab 3'. Then type

   ```
   »figure, plot(yc), hold on, plot(pc)
   ```

   Examine the graph. Use the zoom tool to zoom in on the signal. Explain what you see. Listen to each signal via the sound command. Which sounds like a violin string and which sounds like a Viking's horn?