

Wavelets MATLAB Assignment 1

1.1: Access to MATLAB

To do this assignment, you will need to access a computer on which MATLAB is installed.

1.2: Getting Into MATLAB In the Windows environment, you run MATLAB like any other program. Left click on the start bar and hold the mouse down. This should cause a menu to pop up. Scroll up the menu to the bar marked “Programs” (still holding the mouse down). You need to find the Matlab program, which may be on the main menu, or inside another menu such as “applications” or “math applications.” Slide over and double click on the MATLAB program, and the MATLAB program should start, eventually giving you a MATLAB announcement window and a prompt of the form “>>”.

In the linux/unix environment, open a terminal/console/X-term window and type “matlab” in the command prompt.

1.3: Getting Out Of MATLAB Exit the MATLAB program by typing “quit” at the >> prompt. MATLAB might tell you how many “flops” you have run (whatever this means). If you’re in a computer lab, logoff the machine you are on.

1.4: Entering vectors

The simplest way to enter a vector is by listing its components as in the following example:

```
>> x=[3 16 -2.2 45 0]
```

(Here >> is the matlab prompt on the screen, you just type “x=[3 16 -2.2 45 0]”.) This gives a vector called “x” with these components. Note that a space is needed between components. Type this and see how matlab prints out your vector.

Now type >> x again and see that the vector x reappears. It is stored for future use until you quit MATLAB or redefine it. The name given to a vector can contain up to 19 characters, but must begin with an alphabetical character.

You can get a vector with equally spaced increments via the format

```
>> b=2:3:98
```

This gives a vector starting with 2, with increments of 3 going up to a final value of 96. The format is always initial value:increment:final value. Note that since the components of b are all integers, the display only gave the integer value with no decimal places. Also note that the entire vector scrolls by on the screen. If you want to see some part of this vector, the command >>x(k:l) results in a display of the k^{th} through l^{th} components of x. Try

```
>> b(3:7)
```

and see what you get. If you just type >>x(k) the k^{th} component of x will be displayed.

You can concatenate vectors. For example, type

```
>> c=[x b x 5 2 -7]
```

and see the result. Note that decimals are displayed this time because the vector x has a non-integer component. Four decimal places are normally displayed, unless you change this option.

A useful vector for us is

```
>> n = 0:1:511
```

Type this and see the outcome. If you want to store a vector like n but you don’t really need to see it displayed on the screen, type a semicolon at the end. That will suppress the printout. (This saves a lot of time on long computations.) Try this:

```
>> m=0:1:155;
```

Even though it is not displayed, the vector m is stored. To check this, just type

```
>> m
```

and m will be displayed.

There are some convenient built-in functions that can be used to create vectors. For example,

```
>> w=ones(1,17)
```

will create a vector of length 17 whose entries are all 1. (The “1” is needed so that one gets a 1 by 17 matrix, that is, a vector. Most of the commands described here can be modified to create matrices, but we will stick with vectors for now because this is our main concern. The command `w=ones(17)` gives a 17 by 17 matrix whose entries are all 1.)

Similarly

```
>> w=zeros(1,17)
```

creates a vector of length 17 whose entries are all 0. Notice that we called this vector `w`, which we had used previously. This replaces the old definition of `w` by the new one.

The `linspace` command `linspace(x,y,N)` gives a vector of `N` components starting at `x` and changing linearly up to `y`. For example, try

```
>> w=linspace(0,20,41)
```

Notice that we needed a vector of length 41 to get increments of .5.

1.5: Operations on vectors

Mathematical Functions: Given a vector, you can apply various mathematical functions to each component of the vector as in the following example:

```
>> y=cos(n)
```

Since `n` was the vector obtained above by `n=0:1:511`, `y` is a vector of length 512 whose components are the cosines of the corresponding components of `n`. If you are interested in seeing the plot of `y`, you can read section 1.6 and then come back here. Actually the plot of `y` oscillates too fast for one to see very much, so you would get a more reasonable graph by letting `y=cos(n/64)` (see the text below for an explanation of the /64 part).

Any MATLAB manual should contain a list of MATLAB mathematical functions.

Adding, Subtracting, and Multiplying vectors componentwise: Given two vectors `x` and `y` of the same length, the command `>> z=x+y` produces the sum (componentwise) of `x` and `y`, while `>> z=x-y` produces the difference. These can be combined: try

```
>> p=n+n-n
```

Two vectors of the same length can be multiplied componentwise: `>> z=x.*y` forms the vector of the same length whose components are the product of the corresponding components of `x` and `y`. For example, with the vectors `n` as above, type

```
>> z= n.*n
```

You should get the vector whose components are the squares of the components of `n`. You can confirm this via `>>z(1:6)` for example. Similarly

```
>> t=n.*z
```

will give you the cubes of the components in `n`. (This can also be done directly as we will soon see.)

(The symbol `*` without the `.` is reserved for multiplication of matrices; if you type `>> z=n*y` you will get an error message telling you the matrices cannot be multiplied because the dimensions don't match correctly.)

You can add a scalar to every every component of a vector (or subtract a scalar from every component) as in this example

```
>>t=n-100
```

or

```
>> t=5+n
```

You can also multiply or divide a vector by a scalar componentwise, as follows. Try

```
>> r=3*n
```

and

```
>> s=r/3
```

Exponentiating vectors componentwise: You can raise each element of a vector `x` to a power `p` with the command `>> z=x.^p` For example

```
>> w=n.^2
```

gives the same result as `>> z=n.*n` above. The power `p` does not have to be an integer.

These operations can be combined. For example,

```
>> w=sin(pi*n.^2/10000)
```

gives a vector with values $w(n) = \sin \frac{\pi n^2}{10,000}$. Notice that π can be entered with the symbol pi.

Similarly, the imaginary number $\sqrt{-1}$ can be entered with the symbol i, unless you have defined i to be another variable (so it's better to not use i as a variable).

In general, MATLAB respects the usual order of operations. Parentheses can be added if needed.

For example,

```
>> w=sin((pi*n).^2)
```

gives the vector $w(n) = \sin(\pi n)^2$. If you enter the vector

```
>> w=sin(pi*n.^2)
```

you may be surprised that the output is not 0 (it should be 0, since $\sin \pi n^2 = 0$ for every n).

However, it is effectively 0. If you go back and look at the first few entries via the command `>>w(1:5)`, you will see a term `1.0e-14*` in front of the vector. This means that the listed values are all multiplied by 10^{-14} . The value is not exactly 0 due to round-off error in the MATLAB computations, which are done to 16 significant figures.

1.6: Plotting vectors Vectors are plotted with the plot command. For example, define a vector m by

```
>> m = [6 2 -1 5]
```

and then try

```
>>plot(m)
```

A window labelled Figure No. 1 should pop up on the screen with a plot of m. The window can be moved by putting the cursor on the top bar, holding the left side of the mouse down, and dragging the window to the location you want (this works with any window in the windows environment). The window can be resized by putting the mouse on the edge (whichever edge you are resizing) until you see the sign \leftrightarrow or \updownarrow , then holding the left side of the mouse down, and dragging the edge to the desired position.

Warning: If x is a vector with complex-valued components, the command “plot(m)” will plot the imaginary part of each component above the real part. This is something we never really want to do, so make sure your vectors are real-valued before you plot them. The command “real(z)” will give you the vector whose components are the real parts of the components of z, while “imag(z)” will do the same thing with the imaginary parts. You can also obtain the magnitude of each component with “abs(z)” and the phase angle with “angle(z)”.

Notice that in the plot of m, the value 6 is plotted at the point 1, then 2 at the point 2, -1 at 3 and 5 at 4. Without any further information, MATLAB assumes that a vector x of length N is thought of as $[x(1), x(2), \dots, x(N)]$. However, in this class we prefer to think of a vector of length N as $[x(0), x(1), \dots, x(N-1)]$. To get x plotted starting at 0, create the vector

```
>> r=[0 1 2 3]
```

and then use

```
>> plot(r,m)
```

In general the command “plot(x,y)”, for vectors x and y of the same length, plots the components of y above the corresponding components of x. This is why a vector like `n=0:1:511` is so useful for us; we will plot any vector z of length 512 by the command “plot(n,z)”. For example, with

```
>> z=n.^2
```

try

```
>> plot(n,z)
```

You should get a graph of the function $f(t) = t^2$. If you use “plot(z,n)” instead, you get a graph of the square root function.

Notice that this plotting property allows one to graph functions on whatever interval you like. For example, to graph $f(t) = e^t$ on for $0 \leq t \leq 1$, one gets a very continuous-looking graph by plotting 1,000 evenly spaced points in $[0,1]$. So define

```
>> t=linspace(0,1,1000)
```

```
>> y=exp(t);
```

(recall that the semi-colon prevents the vector from scrolling on the screen)

```
>> plot(t,y)
```

You can plot more than one vector at a time. If you use “plot(x1,y1,x2,y2)” you will get the plots of y1 versus x1 and y2 versus x2. (Similarly for 3 or more plots.)

For example, let

```
>> l = 2*n;
```

```
>> k = 3*n;
```

and then try

```
>> plot(n,l,n,k)
```

For this x1 and x2 do not have to be the same size. Let

```
>> h=[n n+512];
```

```
>> p=[1 1];
```

and try

```
>> plot(n,k,h,p)
```

Changing the plot linestyle: Notice that all the plots so far have displayed a connect-the-dots graph, even for the vector $m=[6 \ 2 \ -1 \ 5]$. This is the default option, but there are others. They are obtained by adding an argument at the end of the plot command of the form ‘?’, where here you do type the single quotation marks, and ? is any of the following symbols associated with the following types:

<u>line-types</u>	<u>symbol</u>	
solid	-	
dashed	-	(typed as - - , it just looks like one keystroke)
dotted	:	
dashdot	-.	
<u>point-types</u>	<u>symbol</u>	
point	.	
plus	+	
star	*	
circle	o	
x-mark	x	

For example, try

```
>> plot(r,m,'x')
```

and

```
>> plot(r,m,'-.')
```

This is particularly useful when plotting more than one vector at a time. In this case, insert the ‘?’ after each pair of vectors: try

```
>> plot(n,l,'-',n,k,':')
```

For this example, since ‘-’ is the default option, you get the same result with

```
>> plot(n,l,n,k,':')
```

Setting your graph viewing window: Consider this example:

```
>> plot(n,n.^2)
```

(Before we did this by defining $m=n.^2$, but it can be done this way also, without explicitly defining m.) Notice that although n is a vector of length 512, the viewing window goes from 0 to 600 along the x-axis. For some reason MATLAB has a mind of its own when setting viewing windows, in both the x and y axes. However, you can set these windows to your own choice as follows. Suppose we want this plotted only for values along the x-axis going from 0 to 511, and along the y-axis we decide we want the limits to be -100,000 to 500,000. After obtaining the graph as we have already done, define a vector

```
>> v=[0 511 -100000 500000]
```

and then give the command

```
>> axis(v)
```

The result will be that the current graph is replotted in the window with bounds equal to the components of v (it doesn’t have to be called v).

Captions: You can add a caption on top of your graph window with the command `>> title('???')`, where `???` is your choice for the caption, and again you do type the single quotation marks. A caption along the x-axis can be added via `>>xlabel('???')` and one along the y-axis via `>>ylabel('???')`. These are added after one has obtained the graph. For example, with the graph already present, try

```
>>title('who cares')
then
>>xlabel('not me')
then
>>ylabel('me neither')
```

1.7: Printing MATLAB plots In the window in which your plot is displayed, go to the option “File” at the top left of the window. Click on it and use the print option that appears.

1.8: The Discrete Fourier Transform

Given a vector z , its Discrete Fourier Transform (or DFT) \hat{z} is computed in MATLAB via the command `>>w=fft(z)` (“fft” stands for “Fast Fourier Transform,” the algorithm for computing the DFT). Try

```
>>z=[1 3 -2 7 8]
>>w=fft(z)
```

Note that w has complex-valued components, even though z is a real-valued vector. Hence the warning above about plotting complex vectors applies. (To see what I mean, try `>> plot(w)`.) Also, we would like to think of z and w as defined on $0, 1, 2, 3, 4$, so let

```
>>m=[0 1 2 3 4]
```

Then you can plot as follows

```
>>r=real(w)
>>plot(m,r,'x')
```

We chose an x-plot because it seems natural for short vectors; for long vectors the default is fine. Similarly you can plot

```
>>s = imag(w)
>>t=abs(w)
```

The inverse DFT is obtained via the command `ifft`. Try

```
>>p=ifft(w)
```

This should agree with the original vector z . Notice that this gives apparently complex values, even though the imaginary parts of all components seem to be 0. However, try

```
>>imag(p)
```

You will see the factor $1.0e-14$ in front of the vector, meaning that its components are multiplied by 10^{-14} . This should be exactly 0, but it isn't because of round-off error in the computations. One gets a similar small error by looking at z minus `real(p)`:

```
>>z-real(p)
```

Assignment 1

In this assignment, you are asked to reproduce certain figures from the text. To get full credit, the type of plot (e.g., x-mark or solid, connect-the-dots) should match the one in the book, and the viewing window should match as well. You may want to add captions for your own identification purposes, but the captions will not be graded. With each figure, please print out the MatLab code you used in creating that figure.

For this assignment, reproduce:

- 1.) Figures 4a and 4b on p. 112
- 2.) Figure 6c on p. 116 (It is easiest to first do 6a and 6b, but only hand in 6c.)
- 3.) Figures 7a, 7d, and 7e on p. 118
- 4.) Figure 9b on p. 119

Remark: The bounds on the viewing window along the x-axis in Figures 4a, 4b, 7a, 7d, and 7e are 0 and 127. The bounds along the y-axis in Figures 6c and 7a are -1.2 and 1.2.