

EEL 4750 / EEE 5502 Foundations of Digital Signal Processing

Reorientation to MATLAB ¹

Files necessary to complete this assignment: `chirp.wav`, `check_myConcat.p`, `check_myEcho.p`

Introduction

The purpose of this tutorial is to reorient you to MATLAB (Matrix Laboratory), a powerful (yet relatively simple) programming language. It is commonly used to quickly prototype and test new algorithms for controls, signal processing, communications, linear algebra, and variety of other mathematical and engineering fields.

Note that MATLAB is generally not used in commercially sold software. Many programming languages can be implemented much faster (C++, Java, etc.), albeit sometimes at the cost of programming simplicity. We use MATLAB because (1) it is specifically designed for direct applications of signals and systems, signal processing, controls, communications, etc. and (2) it is relatively easy to learn. Nevertheless, the algorithms, concepts, and ideas we explore with MATLAB in this course can be applied to any programming language.

By the end of this tutorial, you should be able to:

1. Understand the basics of MATLAB: vectors and matrices, commands, whos, help, commenting, functions, and plotting
2. Program effectively using the special properties of MATLAB
3. Write MATLAB functions

¹Parts of this tutorial are based on a Signals and Systems course written by faculty members at Carnegie Mellon University.

MATLAB Basics

Vectors and matrices: Before we jump into MATLAB, let's briefly look at matrices and vectors. In MATLAB, almost everything is considered a matrix (hence its name: Matrix Laboratory). For those of you familiar with C, C++, or Java, a vector is (essentially) a 1-D array and a matrix is (essentially) a 2-D array. I say "essentially" because vectors and matrices can be used in many operations that are not inherent to all 1-D and 2-D arrays (e.g., matrix multiplication). The size of an array or matrix is written as (# of rows $\times$ # of columns). Consider the following examples:

A 1-D Array of size 1x3. This is called a row vector since it has only 1 row:

$$\begin{bmatrix} a & b & c \end{bmatrix} \quad (1)$$

A 1-D Array of size 3x1. This is called a column vector since it has only 1 column:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad (2)$$

Matrix/2-D Array of size 3x3:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad (3)$$

Typing commands: When you open MATLAB and you will see a prompt like this: `>>`. Since MATLAB is an interactive computing environment, you can start typing at the prompt and you see things happening right away.

Let's start by typing `a=2` and press the Enter key:

```
>> a=2
a =
    2
```

This statement assigns the variable `a` the value 2. You do not have to declare variables in MATLAB. The basic data type in MATLAB is a matrix of double precision numbers. Think of `a` as a 1x1 matrix.

We can also define vectors. Below are example row (1 x 3) and column (3 x 1) vectors.

```
>> b=[1 2 3]
b =
    1    2    3

>> c=[2;3;5]
c =
    2
    3
    5
```

MATLAB Help: One of the greatest benefits of MATLAB is its well-documented help system. I suggest you make use of the `help` command. For example, consider

```
>> help zeros
zeros  Zeros array.
      zeros(N) is an N-by-N matrix of zeros.

      zeros(M,N) or zeros([M,N]) is an M-by-N matrix of zeros.

      zeros(M,N,P,...) or zeros([M N P ...]) is an M-by-N-by-P-by-... array of
      zeros.

      zeros(SIZE(A)) is the same size as A and all zeros.

      zeros with no arguments is the scalar 0.

      zeros(..., CLASSNAME) is an array of zeros of class specified by the
      string CLASSNAME.

      zeros(..., 'like', Y) is an array of zeros with the same data type, sparsity,
      and complexity (real or complex) as the numeric variable Y.

      Note: The size inputs M, N, and P... should be nonnegative integers.
      Negative integers are treated as 0.

      Example:
          x = zeros(2,3,'int8');

      See also eye, ones.

      Reference page for zeros
      Other functions named zeros
```

Whenever you see a MATLAB function you do not know, I encourage you to use `help`. If you click on the link to the full documentation, you can usually learn even more about the function.

MATLAB Whos: Type `whos` to see defined variables and memory usage:

```
>> whos
Name Size Bytes Class
a 1x1 8 double array
b 1x3 24 double array
c 3x1 24 double array
d 3x1 24 double array
Grand total is 10 elements using 80 bytes
```

MATLAB Functions: In addition to working interactively with MATLAB by typing commands after the prompt, you can write functions or scripts in the MATLAB editor. Here is a very simple function:

```
function [y,Fs] = timeScale(inputFile,outputFile,scaleFactor)
[y, Fs] = audioread(inputFile);
Fs = Fs*scaleFactor;
audiowrite(outputFile,y,Fs);
end
```

This function takes as input the filename of an audio file with a '.wav' extension, the filename to save the output, and the scale factor to scale the audio signal. The function outputs the audio signal along with its properties. Type `help audioread` to know more about `Fs`. Time-scaling an audio signal makes the signal play faster or slower depending on the scale factor. Write this code in MATLAB's M-editor (or any other text editor), and save it as 'timeScale.m'. Use the 'chirp.wav' file distributed with this tutorial. You play the saved .wav file by clicking on it or by using `sound` in MATLAB.

To call this function, type:

```
>> [y,Fs] = timeScale('chirp.wav','chirpFaster.wav',2);
>> sound(y,Fs);
>> [y,Fs] = timeScale('chirp.wav','chirpSlower.wav',0.5);
>> sound(y,Fs);
```

If MATLAB complains it cannot find this function, you have to make sure the function is under the MATLAB path. To edit the path, go to File→Set Path and specify the folder you want to include in the path. The order of the paths is relevant, that is, if two functions from two different folders are both under the path, then the function from the first path will be called.

MATLAB Commenting: In MATLAB, commenting is done with the % symbol. Anything on a single line after a % symbol is considered a comment and is ignored. For example,

```
>> a=1 % assign the value 1 to a
```

ignores everything after the % symbol.

MATLAB Plotting: Plotting with MATLAB is easy. Execute the following code one by one.

```
>> stem([1:5],[3 6 2 4 1])
>> plot([1:5],[3 6 2 4 1])
>> plot([1:5],[3 6 2 4 1],'r+')
```

If you feel uncomfortable with the display, try the `axis` command as follows:

```
axis([-1 10 0 7])
```

What does this command do? Note that after you plot a figure (with `stem`, `plot`, etc.), you can export and save it in different formats. Check out the under File→Save As.

MATLAB Loops: MATLAB has both ‘for’ loops and ‘while’ loops. We will generally use ‘for’ loops in this course. To write a ‘for’ loop, execute the following code.

```
>> for n = 1:10
>> disp(['Number ' num2str(n)])
>> end
```

The `n = 1:10` indicates the range of numbers (from 1 to 10) that loop execute over. To write a ‘while’ loop, execute the following code.

```
>> n = 0;
>> while n <= 10
>> disp(['Number ' num2str(n)])
>> n = n + 1;
>> end
```

What do these commands do?

MATLAB If-Else statements: By using ‘if’ and ‘if-else’ statements, you can conditionally execute specific commands. Try the following code.

```
>> rnumber = rand;
>> if rnumber > 0.5
>> disp('Large Number')
>> else
>> disp('Small Number')
>> end
```

What do these commands do? The ‘if-else’ statement is useful when your current analysis depends on previous results.

Exercises (Submit the answers to these exercises)

Exercise 1 (Commenting and commands): Briefly comment every line of the following. Say what each line is doing and write down the outcome: [use no more than one line per MATLAB line]

```
a=zeros(1,5)
a=zeros(1,5);
b=ones(3,2)
c=size(a)
abs([-5.2 , 3])
floor(3.6)
d=[1:-3.5:-9]
e = d
f=d(2)
g=sin(pi/2)
h=exp(1.0)
K=[1.4, 2.3; 5.1, 7.8]
m=K(1,2)
n=K(:,2)
comp = 3+4i
real(comp)
imag(comp)
abs(comp)
angle(comp)
disp('haha, MATLAB is fun')
3^2
4==4
2==8
3~=5
x=[1:1:5]
y=[3 5 7 6 8]
figure
plot(x,y)
figure
stem(x,y)
figure
plot(x,y,'+r')
```

To Submit:

- **Submit:** All of the above commands with your comments.

Exercise 2 (Vector dimensions): Using MATLAB, generate the following discrete-time signals.

```
vect1 = [0 pi/4 2*pi/4 3*pi/4 4*pi/4 5*pi/4 6*pi/4 7*pi/4]
vect2 = cos(vect1)
```

This tells you that the input of many built-in functions can be vectors. Let `vect3` be the transpose of `vect1`. Generate `vect4 = cos(vect3)`.

To Submit:

- **Answer:** Is `vect4` the same as `vect2`?
- **Answer:** What happens when you compute `vect4 + vect2` using MATLAB? (Note the result has recently changed in recent versions of MATLAB)

Exercise 3 (Write a function #1): Here we will practice writing a simple function. Given a vector `input`, write a MATLAB function “`output = myConcat(input)`” that takes a **column vector** as the input and generates a **column vector** that is the concatenation of the input vector with itself. For example, if the input is `[5, 4, 3]'`, then output should be `[5, 4, 3, 5, 4, 3]'`. If the input is not of size $N \times 1$, then the function should display a message saying “This function works only for column vectors” and output `-1` instead of a vector.

Check your function by placing `check_myConcat.p` into the same folder as your `myConcat` function and typing `>> check_myConcat` into the command line. The obfuscated (P-code) script will run your function under many different test conditions and output a grade based on the results.

To Submit:

- **Submit:** The function code for `myConcat.m`.
- **Answer:** What is the output for input `[1:5]`?
- **Answer:** What is the output for input `[1:5]'`?
- **Answer:** What is the output for input `magic(5)`?

Exercise 4 (Write a function #2) : Given a **column vector** `input`, write a MATLAB function “`output = myEcho(input,d,a)`” that delays the signal by `d` samples (a sample is one value in your vector) and has an amplitude that is `a` times larger than the original signal `input`. The output vector `output` should also be a **column vector**. Design your function so that `length(input) == length(output)`. Do not use any for or while loops when you write the function. Name the function “`myEcho.m`,” plot the result (using the code below), and save the graph as “`myEcho.png`”

```
out=myEcho(mod([1:15000]',1000)-500, 2400, 0.7);
plot(1:length(out),out)
```

Check your function by placing `check_myEcho.p` into the same folder as your `myEcho` function and typing `>> check_myEcho` into the command line. The obfuscated (P-code) script will run your function under many different test conditions and output a grade based on the results.

To Submit:

- **Submit:** The function code for `myEcho.m`.
- **Submit:** Saved image `myEcho.png`.
- **Answer:** How does this function model an echo?
- **Answer:** What can be done to make it a better echo model, in other words, what might be lacking in this function?

Exercise 5 (Write a function #3) : Now we will use your two functions to create an echo of a bird chirp sound file. Specifically, we will use `myConcat` to extend the length of the signal and `myEcho` to create echoes. You can achieve this by running the code below:

```
[x, Fs] = audioread('chirp.wav');
y = myEcho(myConcat(x), 1e4, 1) + myEcho(myConcat(x), 3e4, 1) + ...
    myEcho(myConcat(x), 6e4, 1);
audiowrite('chirpecho.wav', y, Fs);
```

To Submit:

- **Submit:** The `chirpecho.wav`.
- **Answer:** What do you hear in the file? Explain why.