AVN Training 2019
Digital Signal Processing Exercises
Lecturer: Mr. Heystek Grobler

(This document is from Prof TG Swart from the University of Johannesburg, Department of Electrical and Electronic Engineering Science, and acknowledges that it from the course Signals and Systems 3A)

The purpose of this exercise is to use familiarize yourself with Matlab

---

1. **Finding Help**

   To find help or a list of available functions, type:

   ```
   >> help
   ```

   If you know what a specific function's name is, and want help on that function, then type `help <function name>`, e.g.:

   ```
   >> help plot
   ```

   If you are not sure what a function's name is, but have an idea of what it does, you can try and look for it using a keyword search using `help <keyword>`, e.g.:

   ```
   >> lookfor logarithm
   ```

2. **Command Window**

   You can use the command window as a calculator, or you can use it to call other Matlab programs (M-files).

   Say you want to evaluate the expression $a^3 + \sqrt{bd} - 4c$, where $a = 1.2$, $b = 2.3$, $c = 4.5$ and $d = 4$, then type in the command window:

   ```
   >> a=1.2;
   >> b=2.3;
   >> c=4.5;
   >> d=4;
   >> a^3+sqrt(b*d)-4*c
   ```

   It will then display the answer of the calculation as:

   ```
   ans =
       -13.2388
   ```

   Note the use of the semicolon! If the semicolon is omitted, then the variable or answer will be printed to the screen. If the semicolon is used then nothing will be printed to the screen, but in the background the value will be assigned to the variable or the answer will be calculated.

   When running a Matlab program, or just using the command window, you can get a list of all variables created thus far, as well as the size and class of each, by typing in:

   ```
   >> whos
   ```

3. **Arithmetic Operators and Matrix Algebra**

The four usual arithmetic operators are:

+ addition,

− subtraction,

* multiplication,

/ division (for matrices it also means inversion).

There are also three other operators that operate on an element by element basis:

.* multiplication of two vectors, element by element,

./ division of two vectors, element-wise,

.^ raise all the elements of a vector to a power.

The arithmetic operators + and − can be used to add or subtract matrices, scalars or vectors. As an example, enter the vectors X and Y and then add them together:

```
>> X=[1,3,4];
>> Y=[4,5,6];
>> X+Y
ans=
    5 8 10
```

For the vectors X and Y the operator + adds the elements of the vectors, one by one, assuming that the two vectors have the same dimension. In the above example, both vectors had the dimension 1×3, one row with three columns. An error will occur if you try to add a 1×3 vector to a 3×1 vector. The same applies for matrices.

To enter a row vector, we seperate the entries by spaces or commas. As before:

```
>> X=[1,3,4]
X =
    1 3 4
>> X=[1 3 4]
X =
    1 3 4
```

To enter a column vector, we seperate the entries by a semicolon:

```
>> Z=[9;8;7]
Z =
    9
    8
    7
```

Using the same approach, you can enter matrices into Matlab:

```
>> W=[1 2 3; 4 5 6; 7 8 9]
W =
    1 2 3
    4 5 6
    7 8 9
```

Experiment with the following commands and see if you can figure out which each one does. Change the values and see what effect it has on the answer:

```
>> A = [1:6]
>> A = [1:20]

>> A = [1:2:10]
>> A = [2:2:10]
>> A = [1:3:10]
>> A = [6:-1:1 6:10]
```

To use a specific entry from a matrix, you use the variable name with the indices of the entry you want:

```
>> W(1,1)
ans =
    1
>> W(1,3)
ans =
    3
>> W(3,2)
ans =
    8
```

The first index specifies the row number and the second index specifies the column number. Note that Matlab only makes use of positive index numbers, 0 can not be used as an index.

To use a specific row from a matrix, you use the variable name with the index of the row you want with a colon for the column index:

```
>> W(2,:)
ans =
    4 5 6
```

To use a specific column from a matrix, you use the variable name with the index of the column you want with a colon for the row index:

To use a specific column from a matrix, you use the variable name with the index of the column you want with a colon for the row index:

```
>> W(:,3)
ans =
    3
    6
    9
```

To compute the dot product of two vectors ($\sum_i x_i y_i$), you can use the multiplication operator *. Using X and Y from the previous example, we use:

```
>> X*Y'
ans =
    43
```

Note the single quote after Y. The single quote denotes the *transpose* of a matrix or a vector. You can see the difference by typing in:

```
>> Y
Y =
    4 5 6
>> Y'
ans =
    4
    5
    6
```

To compute the element by element multiplication of two vectors (or two arrays), you can use the .* operator:

```
>> X .* Y
ans =
    4 15 24
```

Thus, X.*Y means $[1 \times 4,\ 3 \times 5,\ 4 \times 6] = [4\ 15\ 24]$. The '.*' operator is very often used because it is executed much faster compared to code that uses `for` loops.

Matlab (MATrix LABoratory) was specifically designed for doing matrix computations. In general, avoid using `for`'s and `if`'s in your program whenever you can. Matlab's strength lies in its ability to perform efficient matrix and vector computations. Always try to write your program in such a way that vectors or matrices are used.

Other useful functions are `sum`, `std`, `mean`, `min`, `max`, `disp`, `rand`, `clear`, `floor`, `ceil`, `zeros`, `ones`, `find`, `length` and `size`. For more information read the help for each function.

4. **Complex Numbers**

Matlab also supports complex numbers, which plays an important role in signals and systems. The imaginary number is denoted with the symbol i or j, assuming that you did not previously declare or use these as variables. Enter the following commands and see what happens (read the help for each if you are not sure):

```
>> z=3+4i;
>> conj(z)
>> angle(z)
>> real(z)
>> imag(z)
>> abs(z)
```

5. **Plotting**

When working with signals and systems, it is important to be able to visualise either signals or system responses. The plot function in Matlab can be used to create figures. Read the help for plot.

See what happens when you enter the following:

```
>> x=1:20;
>> plot(x)
>> stem(x)
```

To create a new figure, so you can have both figures open at the same time, type `figure` before entering `stem(x)` in the code above.

You can insert x-labels, y-labels and titles to the plots, using the functions `xlabel`, `ylabel` and `title` respectively. To plot two or more graphs on the same figure, use the command subplot. For instance, to show the above two plots in the same figure, type:

```
>> subplot(2,1,1), plot(x)
>> subplot(2,1,2), stem(x)
```

If you have a set of data that contains x and y values, then `plot(x,y)` can be used. Experiment with the following:

```
>> inputs = [0.2 0.3 0.4 0.5];
>> outputs = [300 500 100 900];
>> plot(inputs, outputs);
>> axis([0 1 0 1000]);
```

Matlab will automatically clip the axes to crop the plot data. The `axis` command gives you some control over the axes.

Multiple sets of data can be plotted together, and the marker and line properties for each set can be specified. See `help plot` for more details. Use `inputs` from before, with the following:

```
>> outputs1 = [300 500 100 900];
```

```
>> outputs2 = [200 600 700 800];
>> plot(inputs, outputs1, 'k+-', inputs, outputs2, 'r*:');
```

To create a histogram of randomly generated values, type in:

```
>> samples = rand(1, 1000);
>> hist(samples, 20);
>> axis([0 1 0 1000]);
```

6. **Basic Programming**

Instead of having to repeatedly type in commands, a list of commands can be saved to an M-file, and this file can then be executed from the command window. For instance, if your M-file is called `myfile.m`, then you can execute your file by typing in:

```
>> myfile
```

Make sure that Matlab's working directory is set to the directory where your file is saved.

You can also use M-files to create your own functions to use in Matlab. For instance, if you want to calculate the average for the values in a vector, the following function can be written in an M-file:

```
function y=average(x)
L=length(x);
sum=0;
for i=1:L
    sum=sum+x(i);
end
y=sum/L;
```

To test your function, type in the following:

```
>> x=1:100;
>> y=average(x)
ans =
    50.5000
```

The above function is a good example of where a Matlab function can be used instead of a `for` loop. In this case, the `for` loop can be replaced by:

```
sum = sum(x);
```