Matlab primer for GEOL334/335/384/480/483

You will need Matlab to process and plot your results in some of the labs. Here are a few basic Matlab operations that may be helpful in these tasks.

Remember that Matlab means MATrix LABoratory. It is an *interactive* programming environment designed to work with matrices and vectors (arrays). Thus, matrix is the primary data element in Matlab, and Matlab allow you to add, subtract, multiply, compute functions of, and plot matrices and vectors easily.

Matlab statements are evaluated (executed) immediately as the interpreter reads a line of your script (*M-file*, see below). Note that if an operator ends with a semicolon ';', its operation is performed *silently*, otherwise Matlab will print the resulting value out immediately as it is evaluated.

To open Matlab:

Clicking on the appropriate icon on the Desktop. This will bring the main Matlab window up. It is usually better to keep your work in a Matlab program file, called M-file. To create a new Mfile, use File>New->M-file. To open an existing M-file, use File->Open.

As usual, don't forget to save your M-file before exiting Matlab.

To create an array of values:

To put a list of values in an array (vector), use an assignment operator, such as:

```
x = [0,1.2,2.3,5.0,12];   # x is an array of 5 real values
```

To create a vector of values increasing (decreasing) at regular increments, use the following operator:

```
x = min_value : increment : max_value;
```

for example:

```
x = 1.0 : 1.0 : 200;          # x contains 200 values from 1 through 200
```

The following function will return (assign to `x`) an array of 100 zeros:

```
x = zeros(100);
```

To create an array (row) of `N` values equally spaced from `a` to `b`, use:

```
x = linspace(a,b,N);
```

There is a similar function `logspace()` to create an array with logarithmic spacings (densely sampling the smaller values).

To compute a function of a vector (matrix):

Most functions can be taken of an vector as an argument. For example, let *x* be: an array of 100 values regularly distributed between 0 and 2π:

```
x = 0.0 : pi/50 : 2*pi;
```

Then, to compute 100 values of sin() at each of the points *x*(*i*), we write simply:

```
y = sin(x);
```

Note that Matlab knows how to add constants to arrays, e.g., the following is a valid call to compute sin(x)+1 for each element of an array:

```
y = sin(x)+1;
```

To plot:

To plot a functional dependence Y(X) we must first have the values of argument sampled in an array x, and the values of function filled in another array, y, of the same length (e.g., as done in the example above). Once we have these two arrays, plotting is performed by the following command

```
plot(x,y);
```

For example, a graph of sinusoid can be obtained simply by:

```
x = 0.0 : pi/50 : 2*pi;
plot(x,sin(x));
```

You can put several curves on the same plot, e.g.:

```
plot(x, y, x1,y1, x2,y2);          # three vectors plotted in one graph
```

Thus, the following commands

```
x = 0.0 : pi/50 : 2*pi;
plot(x,sin(x),x,cos(i));
```

will plot both sinusoid and cosinusoid in different colours on the same plot.

Note that `plot()` generates a pop-up window with interactive controls for changing line styles and colours, and for printing the resulting image.

You can add lines and points to an existing plot, so that your graphic will be built in a series of `plot()` calls. To do this, use command `hold` (try this!):

```
x = 0.0 : pi/50 : 2*pi;
plot(x,sin(x));
hold;
for i=1:9
     plot(x,sin(x)+i);
end
```

The above plots 10 sinusoids in the same plot, each offset by 1.0 in the vertical direction. This example should come useful in Lab 4.

Functions:

Matlab allows writing functions that can be used multiple times in different parts of the code. For example, if we want to write a function, named `sum`, that returns a sum of two of its arguments, `arg1` and `arg2`, we create an M-file `sum.m` (the name is essential!), and put the following code into it:

```
function [ result ] = sum ( arg1, arg2 )
result = arg1 + arg2;
end
```

Note that `arg1` and `arg2` may be arrays, in which case `result` will also be an array.

This function can be called from the main program or other functions as:

```
x = sum(1.0,2.0);
y = sum(x,x1);
z = sum(sum(x,x1),x2);
```

and so on.

Most functions allow accept variable numbers and types of arguments, which often creates a very broad functionality. However, be sure to use these multiple interpretations of parameters correctly and always verify the results.

Loops:

To arrange a loop, use for operator:

```
for i=1:n

     % any operations with i …

end
```

This loop repeats the sequence of operators bounded by `for … end` for each value of *i*, from 1 through n.

For example, to reverse the order of values in array x, write:

```
for i=1:size(x)/2

     x(i) = x(size(x)+1-I);

end
```

Useful functions:

If you remember a name of a `command` but not sure how to use it, type:

```
help command
```

for example: `help plot`

Matlab will display usage notes. At the end of the notes, note the list of related functions that could be useful when combine with the one you are checking about.

To determine the length of the vector (array) *x* (to assign the length to a variable `len`), write:

```
len=size(x);
```