# Introduction to Scientific Computing with Matlab

MATLAB is an interactive system for numerical computations. It is widely used in universities and industry, and has many advantages over languages such as C, Fortran and Java, including:

- It is very easy to write MATLAB code to solve complex problems in mathematics, sciences and engineering.
- Data structures in MATLAB require minimal attention. Arrays, for example, do not need to be declared.
- MATLAB has high quality graphics and visualization tools that can be used to analyze computational results.
- MATLAB provides additional *toolboxes* that are designed to solve specific classes of problems. For example, there are toolboxes for statistics, image processing, signal processing, differential equations, splines and optimization.

Because MATLAB is an "interpretive" language codes written in C, C++ and Fortran can be more efficient for very large problems. However, MATLAB is an excellent for developing algorithms and problem solving environments.

**Starting and Exiting** MATLAB.

- Log into one of the computers in the Math/CS lab.
- In a terminal window, at the prompt enter the command: `matlab6`.
- When MATLAB starts up, a MATLAB gui interface should appear on the screen. The *command window* contains a prompt that looks like:

  ```
  >>
  ```

  We can enter data and execute commands at this prompt.
- To exit MATLAB, you can pull down the *File* menu, and let go on *Exit MATLAB*. Alternatively, in the command window, you can use the `exit` command:

  ```
  >> exit
  ```

MATLAB gets its name from MATrix LABratory. A matrix is a 2-dimensional array of numbers, with a certain number of rows and columns. For example:

$$A = \begin{bmatrix} 200 & 107 & -10 & 52 \\ 0 & 221 & 13 & 25 \\ -7 & 1 & 194 & 3 \end{bmatrix}$$

is a matrix with 3 rows and 4 columns. We often say $A$ is a $3 \times 4$ matrix.

A matrix with only one row is sometimes called a row vector. A matrix with only one column in sometimes called a column vector. For example, if

$$\mathbf{x} = \begin{bmatrix} -1 \\ 0 \\ 2 \\ 5 \\ 6 \end{bmatrix} \quad \text{or} \quad \mathbf{y} = \begin{bmatrix} 20 & -15 & 10 & 16 \end{bmatrix},$$

then we say $\mathbf{x}$ is a column vector of length 5, and $\mathbf{y}$ is a row vector of length 4. MATLAB is very useful for solving problems involving matrices and vectors. We explore some of these basics through a series of examples.

**Initializing Vectors**. We can create row and/or column vectors in MATLAB as follows:

```
>> x = [1 2 3 4]

>> x = [1
2
3
4]

>> x = [1, 2, 3, 4]

>> x = [1; 2; 3; 4]
```

Note that we can also refer to these as either $1 \times 4$ or $4 \times 1$ matrices.

**Initializing Matrices**. In general, we can create matrices with more than one row and column just as we did above. Here are some examples:

```
>> A = [1 2 3 4
5 6 7 8
9 10 11 12]

>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
```

**Array Operations**. MATLAB supports certain array operations that can be very useful in scientific computing applications. Some of these operations are:

```
.*    ./    .^
```

The *dot* indicates that the operation is to act on the matrices in an element by element way. That is,

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} .* \begin{bmatrix}5\\6\\7\\8\end{bmatrix} = \begin{bmatrix}5\\12\\21\\32\end{bmatrix}$$

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} .\hat{}3 = \begin{bmatrix}1\\8\\27\\64\end{bmatrix}$$

**Initializing Vectors with Many Entries**. Suppose we want to create a vector, `x`, containing the values $1, 2, \ldots, 100$. We could do this using a loop:

```
n = 100;
for i = 1:n
  x(i) = i;
end
```

In this example, when the code begins, how does MATLAB know `x` will be a vector of length 100? The answer to this question is: it does not know! But MATLAB has a very smart *memory manager* that creates space as needed. Forcing the memory manager to work a lot can make codes very inefficient.

Fortunately, if we know how many entries `x` will have, then we can help out the memory manager by first allocating space using the `zeros` function. Here is how we might do it:

```
n = 100;
x = zeros(1,n);
for i = 1:n
  x(i) = i;
end
```

In general, the function `zeros(m,n)` creates an $m \times n$ array containing all zeros. Thus, in our case, `zeros(1,n)` creates a $1 \times n$ array, which is just a row vector with $n$ entries.

Actually, there is a much easier, and better way, to initialize a simple vector like this using MATLAB's *vector operations.* This can be done as follows:

```
n = 100;
x = 1:n;
```

The colon operator is very useful! Let's see another example where we use it to create a vector. Suppose we want to create a vector, $x$, containing $n$ entries equally spaced between

$a = 0$ and $b = 1$. The distance between each of the equally spaced points is given by $h = \dfrac{b-a}{n-1} = \dfrac{1}{n-1}$, and the vector, $x$, should therefore contain the entries:

$$0, \quad 0+h, \quad 0+2*h, \quad \cdots, \quad (i-1)*h, \quad \cdots, \quad 1$$

We can create a vector with these entries, using the colon operator, as follows:

```
n = 100;
h = 1 / (n-1);
x = 0:h:1;
```

We often want to create vectors like this in mathematical computations. Therefore, Matlab provides a function for it, called `linspace`. In general, `linspace(a, b, n)` generates a vector of $n$ equally space points between $a$ and $b$. So, in our case with $a = 0$ and $b = 1$, we can use:

```
n = 100;
x = linspace(0, 1, n);
```

The **moral** is: If we want to do something fairly standard, then chances are Matlab provides an optimized function for it. To find out, we could use the `help` and/or `lookfor` commands.

**Evaluating Functions and Plotting**. Suppose we want to plot the function $\sin(3\pi x)$ on the interval $0 \le x \le 1$. We learned the basic idea of plotting when we were six years old: Plot a bunch of points $(x_i, y_i)$, and connect the dots. We can do this in Matlab by:

- create a vector of $x$-coordinates
- create a vector of $y$-coordinates
- use the Matlab command `plot(x,y)`

This can be done as follows:

```
n = 100;
x = linspace(0, 1, n);
y = zeros(1, n);
for i = 1:n
  y(i) = sin(3*pi*x(i));
end
plot(x, y)
```

Note that in this code we have

- used the `linspace` command to efficiently create the vector `x`,

- helped out the memory manager by using the `zeros` command to allocate space for the vector `y`,
- used a loop to generate the entries of `y` one at at time,
- and used the `plot` command to draw the graph.

We can actually shorten this code by replacing the loop with a single, *vector operation.* In general, functions like `sin` can be use on arrays of entries. That is, if `z` is an $m \times n$ array containing entries $z_{ij}$, then `sin(z)` is an $m \times n$ array containing the entries $\sin(z_{ij})$. Thus, we can do the above computations simply as:

```
n = 100;
x = linspace(0, 1, n);
y = sin(3*pi*x);
plot(x,y)
```

If you can use array operations instead of loops, then you should do it. In general, **array operations are more efficient than using loops**.

Finally, we mention that for *easy* functions, we can use MATLAB's `inline` and `ezplot` commands. For example, to plot $f(x) = \sin(3\pi x)$ on the interval $0 \le x \le 1$, use:

```
f = inline('sin(3*pi*x)');
ezplot(f, [0, 1])
```

**Function mfiles and Script mfiles.** So far we have only used MATLAB as a sophisticated calculator. It is much more powerful than that, and should be thought more as a computer language. Therefore, there should be some capability for writing sophisticated programs, which include functions, subroutines, structures, classes, objects, etc. MATLAB has all of these capabilities, but since this is not a MATLAB programming class, we cannot go into details about them.

However, we should be aware at least of two types of programs that we can write: *scripts* and *functions.* Each kind of program is written using your favorite editor, and should be saved in a file with the *.m* extension. The difference between a script and a function is:

- A *script* is a file containing a collection of MATLAB commands which are executed when the name of the file is entered at the MATLAB prompt. This is very convenient if you have to enter a lot of commands. But you must be careful: variables in a script are global to the MATLAB session, and it can be easy to unintentionally change values in certain variables.
- A *function* is a file containing a collection of MATLAB commands that are executed when the function is called. The first line of a function must have the form:

function [*out1, out2,...* ] = *FunctionName(input1, input2, ...)*

Any data or variables created within the function are private, and so there is less of a chance of accidentally changing a variable. You can have any number of inputs to the function. If you want to pass back results from a function, then you can specify any number of outputs. Functions can call other functions, and in this way you can write sophisticated programs as in any powerful language such as C, Fortran and Java. The beauty of MATLAB, though, is that you do NOT need to declare variables as `int, double, double[]`, etc.

For additional information on functions, please see other, more complete references.

**Naming Functions and Scripts**. MATLAB is case sensitive both in terms of variables, and with respect to the names of functions and script files. As mentioned above, these should all have names of the form:

*FunctionName*`.m` or *ScriptName*`.m`

MATLAB has **a lot** of built in functions available for your use. For example, if you want to find the roots of a polynomial, there is a function available for this purpose – you don't need to write your own. To find out more about this function, use the `help` command:

```
>> help roots
```

All MATLAB functions are named with lower case letters. If you write a function with the same name, MATLAB has a way of choosing which function to use. To be sure that you (or MATLAB) don't get confused over which one to use, you might include some upper case letters in your function name. For example, if you wanted to name a function roots, you might use `Roots.m` instead of `roots.m`.

**Getting Help**. MATLAB has two useful commands for getting help:

- The `help` command can be used to get some basic help on how to use a MATLAB function. For example, if we want to know how to use MATLAB's `plot` command, we can use:

  ```
  >> help plot
  ```

  The difficulty with this command is that we have to know that there is a function with the name plot. For example, suppose we try:

  ```
  >> help polynomial
  ```

  MATLAB will respond with a message saying there are no functions called `polynomial.m`. But MATLAB is a sophisticated package, so there must be some functions that can be used to manipulate polynomials. How do we find these functions?

- One way is to use the `lookfor` command, which searches for functions that reference key words. So we can try:

  ```
  >> lookfor polynomial
  ```

  and see what is found.