Last class you were introduced to the Matlab environment. Among the things you should have learned is the various ways to input data into Matlab as well as some simple array operations specific to Matlab. Today we will cover some basic Matlab functions which are broken down into four broad groups: general functions, output formatting functions, array and matrix information functions, and math functions. We will also cover the problem solving method that we will use throughout the remainder of this course.

General Functions:

`help function` : Displays help on a particular function. Type "help" without the `function` argument to see help topics.

`who` : Lists all variables that have been defined.

`whos` : Lists all variables that have been defined and gives their sizes and types.

`clear` : Clears variables and functions from memory.

`clc` : Clears command window. Does not clear any variables from memory.

`clf` : Clears the graphics window. Does not clear any variables from memory.

`what` : List Matlab specific files in the current directory.

`diary filename` : Writes Matlab work to a file. Type "diary off" to stop saving Matlab work. Any available drive, directory, and filename may be used as the argument `filename` as long as the drive or directory is not write protected. Example `filename`: `a:\class3.txt`

`cd path` : Changes the current working directory. The argument `path` may be any available drive or folder.

`dir` : Lists all directories and files in the current working directory.

Output Formatting Functions:

`format short` : Scaled fixed point format with 5 digits.

`format long` : Scaled fixed point format with 15 digits.

`format short e` : Floating point format with 5 digits.

`format long e` : Floating point format with 15 digits.

`format bank` : Fixed format for dollars and cents.

`format rat` : Approximation by ratio of small integers.
Array and Matrix Information Functions:

- **length(variable)**: Returns the number of elements in an array variable or the largest dimension of a matrix.

- **size(variable)**: Returns a vector containing two integers which represent the number of rows and columns in the variable. The number of rows is the first number and the number of columns is the second number.

Math Functions:

- **sin(argument)**: Calculates the sine of the argument. The argument may be a scalar, vector, or matrix.

- **cos(argument)**: Calculates the cosine of the argument. The argument may be a scalar, vector, or matrix.

- **tan(argument)**: Calculates the tangent of the argument. The argument may be a scalar, vector, or matrix.

- **exp(argument)**: Raises e to a power given by the argument's value. The argument may be a scalar, vector, or matrix.

- **log10(argument)**: Base 10 logarithm of the argument. The argument may be a scalar, vector, or matrix.

- **abs(argument)**: Absolute value of the argument. The argument may be a scalar, vector, or matrix.

- **floor(argument)**: Rounds the argument towards negative infinity. The argument may be a scalar, vector, or matrix.

- **ceil(argument)**: Rounds the argument towards positive infinity. The argument may be a scalar, vector, or matrix.

- **round(argument)**: Rounds the argument to the nearest integer. The argument may be a scalar, vector, or matrix.

- **rand(m,n)**: Creates an $m \times n$ matrix with random numbers between 0 and 1 as elements. Type rand without any arguments to get a single random number between 0 and 1.

It is important to note that this is merely a small fraction of the functions that Matlab has to offer. Please explore the help function in Matlab for a full list of all the functions Matlab has at its disposal. This is also by no means all of the functions that we will use in this class. Other functions will be covered as the course progresses.
Problem Solving Method

Problem solving is a large part of engineering. If you have not found this out for yourself yet, you are probably doing something wrong. Today we want to develop a general problem solving approach that we can apply to any engineering problem, particularly those that are solved with computers and/or programming.

The approach that we will use has six steps:
1. Familiarize yourself with the problem.
2. Describe the input required to solve the problem and the desired output.
3. Work the problem by hand for a simple set of representative data.
4. Develop a detailed algorithm for solving the problem.
5. Write the program to solve the problem.
6. Test the program's solution extensively and with a variety of data.

Familiarize yourself with the problem:

The first step is to familiarize yourself with the problem at hand. You should obviously read the problem in its entirety a few times. Once you feel that you fully understand the problem, read it again because you probably still missed something. You should be able to describe the problem in great detail to someone who is not familiar with it. Once you can do this, then you might understand the problem well enough to solve it. Keep in mind that understanding the problem is at least half of the battle.

After you have a satisfactory understanding of the problem you should formulate a problem statement. This problem statement helps direct your solution to the problem. It is important to develop a clear and concise problem statement to avoid any possible misunderstandings. If you can't state what the problem is, then you obviously don't understand it well enough to attempt a solution.

Input / Output Description:

The second step is to carefully describe the information that is given to solve the problem and then identify the values to be computed. These items represent the input and output for the problem. At this point, we are not defining the steps needed to determine the output. We are simply deciding what information should be used to compute the desired output.

By Hand Example:

The third step is to work the problem by hand using a calculator, if necessary, with a simple set of representative data. This step is very important, as it is here that you work out the details of the solution. The idea is that if you cannot take a small set of data and compute the output by hand, then you are not ready to move on to the next step. If you cannot solve the problem for the small data set, you should reread the problem and correct any mistakes or misunderstandings.
Develop an Algorithm:

The fourth step is to develop a detailed algorithm. Stated bluntly, an algorithm is a finite sequence of steps for solving a logical or mathematical problem. In more simple terms, an algorithm is a step-by-step process for doing some task, solving an engineering problem in our case. An algorithm typically takes the form of a list of steps outlining the solution process for a particular problem.

A tendency to use an underdeveloped algorithm is usually prevalent, especially with problems that are seemingly easy. I warn you that few problems are as easy to solve as they appear. It is very important to develop an algorithm extensively. You should be able to give your algorithm to someone unfamiliar with the problem at hand and he or she solve the problem without any further information or explanation. This is not easy to accomplish but it is crucial to solving the problem. Time spent up-front developing a good algorithm is easily offset by time saved down the road during the coding and debugging process.

Write the Program:

The fifth step is to write the program to solve the problem or, in other words, "code it up". If you spent the time needed to develop an extensively detailed algorithm, then this step should be fairly easy and painless. A well developed algorithm should translate almost line for line into computer code. If you did not spend the time needed to develop a good algorithm, then this step becomes much more difficult to perform. If you get to this point and realize that your algorithm is not as fully developed as it probably should be, I suggest leaving the computer and further developing your algorithm. You can always come back after you have revised your algorithm.

Test the Program:

This sixth step of the problem solving process is commonly known as debugging. You have now spent ample time making sure you understand the problem, developing a good algorithm, and coding the program carefully. Now we need to test the program to make sure that it runs properly and gives us the correct output. So you run your program and…

Guess what? It did not work!

…not the first time anyway. The process of running the program, identifying and correcting errors, and eventually emerging with a fully operational and correct piece of code is what is termed debugging. Debugging is a large part of programming. It is typical to spend 5 minutes coding the program and something along the lines of 5 days debugging the program.

After you have finished debugging your program, you should test the output extensively. You should definitely check your program output against your by hand calculation. If the two do not match, then obviously there is a problem somewhere. You should test your program exhaustively until you are satisfied that the output it gives is correct. A program that occasionally yields a correct answer is absolutely useless because it is unreliable. You need your program to be 100% correct 100% of the time, not 50% correct 50% of the time (or worse).