Introduction to Matlab

- What is Matlab?

  Matlab is software written by a company called The Mathworks (`mathworks.com`), and was first created in 1984 to be a nice "front end" to the numerical routines created for matrices- Matlab is short for "Matrix Laboratory". Matlab now has software routines for just about any numerical problem you can think of, and has become an industry standard in engineering.

- Why not Maple?

  While Maple does great things **symbolically** (like differentiate, integrate, etc.), it was not built to handle numerical problems. Matlab was built to solve numerical problems.

  We might mention here that it is possible to call Maple from Matlab and vice versa, but that will take us too far into computer programming (the interface between the two is still quite raw).

- Where is Matlab?

  Matlab has been installed on all machines in the Mathematics Computer lab. To run it, you might set up the icon on your toolbar first. Go to `Applications`, then `Mathlab`, find the Matlab icon (a surface in 3-d), and drag it to the toolbar.

  Now **to run Matlab**, click the icon. You will see a "flashscreen", then a window will come up. The biggest window is the **command window**, where we type our Matlab commands.

- General help and getting started:

  When you first open the command window, you can watch a video, look at demos, or read the Getting Started guide. You might play around with these later to get to know Matlab a little better.

Using Matlab's Command Window

Basic Arithmetic and Assignment

We can use Matlab "live" by typing in commands. Here are some to get you started- Think about what you're doing as you type these in, and see if the Matlab output is what you expect.

```
x=pi
y=3*sin(x/2)
w=2^3
z=y+exp(-y)
```

```
log(z)
z=(1+3*i)*(5-2*i)
help log
whos
```

Things to notice: How is π defined? How was e^{-y} defined? The last command is one way to view all the variables that have been assigned so far.

Re-do the line `z=y+exp(-y)`, but add a semi-colon at the end (*Hint: Use the up-arrow key*). What happens?

Programming Note: Matlab looks at the equal sign like it is an assignment, not a mathematical statement of equality. That is, the following command will give an error:

```
y+exp(-y)=z
```

Therefore, the thing on the left side of the equation is a variable name you will use to assign whatever is on the right.

Creating Arrays and Manipulating Arrays

Before proceeding, clear the memory and the screen:

```
clear
clc
```

Arrays are created by assigning them to a variable. For example,

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

is entered as:

```
A=[1 2 3 4; 5 6 7 8];
```

or as:

```
A=[1 2 3 4
5 6 7 8];
```

Note the use of the semicolon: Inside a matrix, the semicolon indicates the end of a row. Outside the matrix, the semicolon suppresses Matlab output. You can also separate numbers using a comma if you'd prefer that. Rows and columns are entered in a corresponding way, as either a $1 \times n$ matrix or as a $n \times 1$ matrix.

We access elements of the matrix in a natural way. For example, the $(2, 3)$ element of A is written as `A(2,3)` in Matlab (in this case, $A(2, 3)$ is 7). You can change the elements using the assignment operator `=`. For example, if we want to change the $(1, 3)$ element of A from 3 to 6, type:

```
A(1,3)=6;
```

Special Commands: The colon operator

- `a:b`

  Produces a listing from a to b in a row:

  $$\texttt{a:b} \text{ gives } [a, a+1, a+2, \ldots, a+n]$$

  where n is the largest integer so that $a + n \leq b$. For example, $x = 2 : 9$ puts x as a row vector whose elements are the integers from 2 to 9.

- `a:b:c`

  Produces the numbers from a to c by adding b each time:

  $$\texttt{a:b:c} \quad \text{gives} \qquad [a, a+b, a+2b, \ldots, a+nb]$$

  where n is the largest integer so that $a + nb \leq c$. For example, $1 : 2 : 7$ returns the numbers $1, 3, 5, 7$. Type the following into Matlab to see what you get: `1:2:8` and `1:0.5:6`

Matlab commands associated with Arrays

- `linspace(a,b,c)`

  Produces c numbers evenly spaced from the number a to the number b (inclusive). For example, `x=linspace(2,3.5,40)` produces 40 numbers evenly spaced beginning with 2 and ending with 3.5.

  SHORTCUT: Leaving off the third number c will give you 100 numbers between a and b (That is, $c = 100$ is the default value.)

  Compare this with the colon operator. We would use the colon operator if we want to define the length between numbers, and use `linspace` if we want to define the endpoints.

- Random arrays (handy if you just need some quick data!)

  `A=rand(m,n)` Produces an $m \times n$ array of random numbers (uniformly distributed) between 0 and 1. If you just want a single random number between 0 and 1, just type `rand`

  `A=randn(m,n)` produces an $m \times n$ array of random numbers (with a normal distribution) with zero mean and unit variance. If you want a single random number (with a normal distribution), just type `randn`

- `A=zeros(m,n)` Produces an $m \times n$ array of zeros.

- `A=ones(m,n)` Produces an $m \times n$ array of ones.

- `A=eye(n)` Produces an $n \times n$ identity matrix.
- `A=repmat(B,m,n)` Matrix A is constructed from matrix (or vector) B by replicating B m times down and n times across.

  Example: Let $B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Then `A=repmat(B,2,3)` creates the array:

```
A =
        1        2        1        2        1        2
        3        4        3        4        3        4
        1        2        1        2        1        2
        3        4        3        4        3        4
```

Matrix Arithmetic

- Transposition is denoted by the single quote character `'`. That is, `A'` $= A^T$. (CAUTION: If A contains complex numbers, then `A'` is the *conjugate transpose* of A, sometimes denoted as $A^* = \bar{A}^T$)
- Matrix addition and subtraction is performed automatically and is only defined for matrices of the same size.
- Scalar addition. If we want to add a constant c to every item in an array A, type: `A+c`
- Scalar Multiplication: We can multiply every number in the array by a constant: If A is the array and c is the constant, we would write: `B=c*A`
- Matrix Multiplication: Use the regular multiplication sign for standard matrix multiplication. If A is $m \times n$ and B is $n \times p$, then `A*B` is an $m \times p$ matrix, as we did in linear algebra.
- Elementwise Multiplication. We can multiply and divide the elements of an array A and an array B *elementwise* by `A.*B` and `A./B`

  Exponentiation is done in a similar way. To square every element of an array A, we would write: `A.^2` This is the same as saying `A.*A`
- Functions applied to arrays: Matlab will automatically apply a given function to each element of the array. For example, `sin(A)` will apply the sine function to each element of the array A, and `exp(A)` will apply e^x to each element of the array. If you write your own functions, you should always decide ahead of time how you want the function to operate on a matrix.

Accessing Submatrices

Let A be an $m \times n$ array of numbers. Then:

The notation:	Yields:
`A(i,j)`	The (i,j)th element
`A(i,:)`	The entire ith row
`A(:,j)`	The entire jth column
`A(:,2:5)`	The 2d to fifth columns, all rows
`A(1:4,2:3)`	A 4×2 submatrix

Example: What kind of an array would the following command produce?

```
A([1,3,6],[2,5])
```

A 3×2 matrix consisting of the elements:

$$\begin{array}{cc} A(1,2) & A(1,5) \\ A(3,2) & A(3,5) \\ A(6,2) & A(6,5) \end{array}$$

Example: Create a 5×5 zero array, and change it to:

$$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

Answer:

```
A=zeros(5,5);  %Create the matrix of zeros
b=[1 2 3;4 5 6; 7 8 9];
A(2:4,2:4)=b;
```

Note also the use of the `%` sign. It is used to denote *comments*; that is, Matlab would ignore everything on the same line after the `%` sign.

Adding/Deleting Columns and Rows:

Its straightforward to insert and/or delete rows and columns into a matrix. Before doing it, we define `[ ]` as "the empty array": the array with nothing in it.

In the following, let A be a 4×5 array, let b be a 1×5 row, and c be a 4×1 column.

Examples of use (each of these are independent from the previous):

The command:	Produces:
`A(1,:)=[];`	Delete the first row.
`A([1,3],:)=[];`	Delete rows 1 and 3.
`A(:,3)=[];`	Delete column 3.
`A(:,1:2:5)=[];`	Delete the odd columns.
`A(1,:)=b;`	Put **b** as row 1.
`A(:,6)=c;`	Add c as the last column.
`d=[c , A(:,1:3)];`	d is c and columns $1-3$ of A.
`A=[A(:,1), c, A(:,2:5)];`	Insert c as column 2 of A, others shift 1 over.
`A=[A(1,:); b; A(2:4,:)];`	Insert b as row 2 of A, others shifted 1 down.

Example: Matlab comes with some built-in data sets. One such set is the image of a clown. For fun, we'll load the array in, display it, then we'll remove all of the even rows and columns, then re-display it:

```
load clown
whos
image(X);
colormap(map);
X(2:2:200,:)=[];
X(:,2:2:320)=[];
image(X);
```

Once you're done, you may want to clear the memory and the screen:

```
clear
clc
```

How do I get a Plot?

Here's a quick example to get us started:

```
x=linspace(-pi,3*pi,200);
y=sin(x);
plot(x,y);
```

You'll see that we had to create a domain array and a range array for the function. We then plot the arrays. For example,

```
plot([1,2],[3,4]);
```

will plot a line segment between the points $(1,3)$ and $(2,4)$. So, Matlab's plotting feature is drawing small line segments between data points in the plane.

Examples

1. Matlab can also plot multiple functions on one graph. For example:

   ```
   x1=linspace(-2,2);
   y1=sin(x1);
   y2=x1.^2;
   x2=linspace(-2,1);
   y3=exp(x2);
   plot(x1,y1,x1,y2,x2,y3);
   ```

   produces a single plot with all three functions.

2. `plot(x1,y1,'*-');`

   Plots the function `y1`, and also plots the symbol `*` where the data points are.

3. `plot(x1,y1,'k*-',x2,y3,'r^-');`

   Plots the function `y1` using a black (k) line with the asterisk at each data point, PLUS plots the function `y2` using a red line with red triangles at each data point.

   The following lists all of the built in colors and symbols that Matlab can use in plotting: (NOTE: You can see this list anytime in Matlab by typing: `help plot`)

Code	Color	Symbol	
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	−	solid
b	blue	∗	star
w	white	:	dotted
k	black	−.	dashdot
		−−	dashed

4. The following sequence of commands also puts on a legend, a title, and relabels the $x-$ and $y-$axes: Try it!

```
x=linspace(-2,2);
y1=sin(x);
y2=x.^2;
plot(x,y1,'g*-',x,y2,'k-.');
title('Example One');
legend('The Sine Function','A Quadratic');
xlabel('Dollars');
ylabel('Sense');
```

5. Other Things: If you look at the plotting window from the last example, you'll see lots of things that you can do. For example, there's a zoom in and a zoom out feature. You can also edit the colors and symbols of your plot, and the title, legend and axis labels. Try them out!

Plotting in Three Dimensions

Matlab uses the `plot3` command to plot in three dimensions. We won't be using this feature here. To get more information, either type `help plot3` or refer to the Matlab Graphics Manual.

M-Files: Functions and Scripts

What is a Matlab Function? A Matlab function is a sequence of commands that uses some input variables and outputs some variables. The following is a very simple Matlab function:

```
function y=square(x)
%FUNCTION Y=SQUARE(X)
%This function inputs a number or an array, and
%  outputs the squares of the numbers.

y=x.^2;
```

You would type this in the editor, then save it as `square.m` (the filename must be the same name as the function, and it must use the `.m` extension).

You'll notice that the first line states "function". This is always the first line of a Matlab function. The remarks that follow the first line are very important. When you type `help square`, these three lines appear. So when you write your own functions, you should include comments so that you can remember how to use it.

The rest of the first line defines what the input variable is (x), and what the output variable is (y).

A Matlab function can produce multiple outputs. For example:

```
function [A,b]=randmatrix(n)
%FUNCTION [A,b]=RANDMATRIX(N)
%Produces an 2n x 2n random matrix A and a random
%column vector b.
nn=2*n;
A=rand(nn,nn);
b=rand(nn,1);
```

To call this function from Matlab, you would write, for example,

`[X,y]=randmatrix(10);`

You'll notice that after running this program, the variables internal to the function (in this case `nn`) disappear. This is one major difference between a *script* and a *function*:

- A **script file** is a text file with a sequence of Matlab commands. It is used by Matlab just as if you were typing the commands in from the keyboard. You should use a script file whenever you are experimenting in Matlab- it makes life a lot easier!
- A **function** in Matlab is like a subroutine in programming. That is, once the function has been called, all of its variables are local to that function- you cannot access them from the keyboard, and the variables are erased once the function is finished.

The Bisection Method

To see how functions and scripts work together, we will solve a sample problem:

> Use the bisection method to locate the solutions to the following equation: Sketch the function using Matlab and identify intervals which contain the solutions, find the roots correct to six decimal places.
>
> $$1 + 5x - 6x^3 - e^{2x} = 0$$

SOLUTION:

First we plot the curve to find our intervals. In the command window,

```
x=linspace(-5,5);  %Creates the x-values for the plot
y=1+5*x - 6*x.^3 -exp(2*x);  %Creates the y-values
plot(x,y)
```

After some trial and error, you might find that all the solutions are in the interval $[-1, 1]$. In fact, $x = 0$ is a solution (we will let the bisection algorithm find it anyway).

We will be using `bisect.m` from the textbook. Notice the first line of code (starts with the word **function**).

All functions start with the word function, followed by the sample command. In this case,

```
function xc = bisect(f,a,b,tol)
```

We see that f will be the function, the interval on which we can find the solution is $[a, b]$, and we will look until the given tolerance (error) is reached.

Now, construct a script file that will call the bisection algorithm for us, since we need to run it three times. Open up the editor in Matlab, and type the following. From the graph, we see that the zeros were contained within the following intervals: $[-1, -0.6], [-0.2, 0.1], [0.3, 0.6]$

```
Intervals=[-1,-0.6; -0.2,0.1; 0.3,0.6];

f=inline('1+5*x-6*x.^3-exp(2*x)');

for j=1:3
  a=Intervals(j,1);
  b=Intervals(j,2);
  tol=0.5 * 10^(-6);
  y(j)=bisect(f,a,b,tol);
end
y    %Without the semicolon, this prints the values
```

Save this as something with a .m extension, for example: `bisectExample.m` Run it in the command window by typing the filename (without the .m):

```
bisectExample
```

The solutions are in the array y.

Turning in Matlab Code

Script files can be converted into HTML files with output, then printed and turned in. Try converting the file we just ran.