Introduction to Matlab

What is Matlab?

Matlab is a computer program designed to do mathematics. You might think of it as a super-calculator. That is, once Matlab has been started, you can enter computations, and Matlab will produce the results. Matlab has many built-in programs and has some great graphics features that we'll discuss later.

Starting the Matlab Program

The Matlab program physically resides on the computer named "Hope" in the math lab. To run Matlab, you do not have to be physically sitting at Hope; you can be at any of the computer terminals.

Once you have logged into your particular machine, you'll need to log into Hope to make Matlab work. To do this, in your command window type: `ssh hope` and you'll be on that machine. Now just type `matlab` and you'll see the program start. The splashscreen will come up, and you'll get a command window that looks like:

```
                              < M A T L A B >
                  Copyright 1984-2001 The MathWorks, Inc.
                       Version 6.1.0.450 Release 12.1
                                May 18 2001

  To get started, select "MATLAB Help" from the Help menu.
>>
```

How does Matlab work?

You can make Matlab do computations three different ways:

- Type commands directly into the keyboard.
- Have your Matlab commands typed into a separate text file (called a **script file**), and then have Matlab read these commands in. This is very nice- it gives you documentation and allows you to run similar computations several times without having to re-type the commands.
- Define your own functions by typing a separate text file (called an **m-file**).

Saving your work

If you haven't written a script file, but are doing your computations "live", you may want to begin the session by typing: `diary filename`

All subsequent keyboard input and output will then not only be on the computer monitor, but will also be saved as "filename". For example, if you're using

Matlab for homework problem 3.1, you may use the command: `diary hw3_1` to save your work.

Important: The "diary" command *must* be used *prior* to typing in the commands you want to save.

Matlab has a very nice text editor that you can use to type out and save Matlab functions and scripts- To access the editor, type `edit` in the Matlab command window.

1 Introductory Commands

1. **Arithmetic**

   Matlab understands all of the basic arithmetic functions, `+, -, *, /, ^` are addition, subtraction, multiplication, division and exponentiation. Type them in just as you would write them. For example, 2^5 would be typed as `2^5`.

2. **Trigonometric Functions**

   Matlab understands the basic trig functions sine, cosine and tangent as `sin , cos , tan` . So, for example, the sine of 3.1 would be typed as: `sin(3.1)`

   The number π is used so frequently that Matlab has its (approximate) value built-in as the constant `pi`. For example, $\sin(\pi)$ is typed as `sin(pi)`. Note that π uses a lowercase "P".

3. **Exponential and Logarithmic Functions**

   Matlab does not have the number e built-in as a constant (like π). To take the number e to a power, use the functional form: $\mathrm{e}^x =$ `exp(x)` So if I want the number e, I would type `exp(1)`, and so on.

   For the natural log (log base e), use the notation `log`. For example, $\ln(3)$ is written as `log(3)`. We will only use the natural log- if in the future you want a different base, look up the log command by typing `help log`.

4. **Complex Numbers and Arithmetic**

   Matlab has complex arithmetic built-in. Either the letter i or j can be used to represent $\sqrt{-1}$, but a word of caution is in order here: You can only use i or j for $\sqrt{-1}$ ONLY if you have not previously defined them. If you think you're going to use complex numbers, do not use the letter i for anything but complex arithmetic! Example: `(0.2+3*i)*(5+2*i)` will multiply the two complex numbers together (using complex arithmetic).

1.1 Helpful Administrative Commands

The following commands are useful as you begin to use Matlab more and more:

`who` List all variables currently in use.

`whos` List all variables, and their sizes.

`ls` or `dir` List the contents of the current directory.

`help` *command* List the help file for the function *command*. For example, to get help on the sine function, type `help sin`.

`demo` Lists all the demonstration programs that Matlab came with- This is fun to look at. We don't have all of them; you can go to Matlab's website to look at more: `www.mathworks.com`.

2 Exercise Set

1. The following script file is an example of Newton's method applied to a function $f(x) = xe^x - \cos(x)$. Recall that Newton's Method solves for x: $f(x) = 0$ by taking an initial guess, x_0, and refines the guess by:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

```
x=0.2;  %Initial guess for solution to f(x)=0
for k=1:5
  y=x*exp(x)-cos(x);
  dy=(x+1)*exp(x)+sin(x);
  x=x-(y/dy)
end
```

Notes about the code:

- We see our first `for` loop. We'll discuss what this does in class.
- Note the use of `%` to make comments.
- Note that `x=x-(y/dy)` does NOT have a semicolon at the end.

(a) Use the `edit` feature to type it in and save it as `newton1.m`

(b) Run the code after you've saved it by typing `newton1` in the command window.

(c) Write down Matlab's output.

(d) To see more significant digits, type `format long`

(e) Type `whos` and write down Matlab's answer. If you're continuing to the next exercise, type `clear` to clear Matlab's memory.

3 Arrays of Data

For those of you that have had linear algebra, Matlab was originally designed as a "front end" to access LINPACK and EISPACK, which are numerical linear algebra packages written in FORTRAN. From this beginning, Matlab's basic data type is the matrix.

If you haven't had linear algebra, simply think of a matrix as an array of numbers- like on a spreadsheet. We'll use arrays to store numbers.

Arrays that have m rows and n columns are said to have size $m \times n$ (remember, rows come first!).

For example, suppose I want to enter the following 2×4 array (or matrix) in Matlab:

$$A = \left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{array} \right]$$

I would type:

```
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 A is written as `A(2,3)` in Matlab.

Special Commands: The colon operator

- `a:b`

  Produces the integers from a to b in a row. 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. 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`

- `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.)

3.1 Matlab commands associated with Arrays

- 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. `A=randn(m,n)` produces an $m \times n$ array of random numbers (with a normal distribution) with zero mean and unit variance.
- `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.

3.2 Solving $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$

To solve $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, Matlab has two basic commands: `x=A\b` or `x=pinv(A)*b`. The command `pinv(A)` computes the *pseudoinverse* of A, which we will discuss later in the section dealing with the Singular Value Decomposition.

In linear algebra, there were three possible outcomes for solving $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$. They were:

1. A unique solution.
2. An infinite number of solutions.
3. No solution.

Matlab will always give exactly one solution. We need to interpret that solution in the second two cases. In the case of an infinite number of solutions (we have free variables in this case, also called *an underdetermined system*), the two methods may give different answers:

- `x=A\b` provides the most zeros in $\mathbf{x}$.
- `x=pinv(A)*b` gives $\mathbf{x}$ with the smallest norm.

Example: Let $A = \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 1 \end{bmatrix}$, with $b = \begin{bmatrix} 9 \\ 3 \end{bmatrix}$. Then the full solution is:

$$\mathbf{x} = \begin{bmatrix} 9+2t \\ 3-t \\ t \end{bmatrix} \tag{1}$$

In Matlab, the result of typing `x=A\b` is $x = [0, -15/2, -9/2]^T$ and the result of typing `x=pinv(A)*b` is $x = [4, 11/2, -5/2]$.

(MATLAB HINT: You can get Matlab to return numbers as fractions by typing `format rat`)

In the case of an overdetermined system (a system with no solution), Matlab will automatically return the *least squares* solution- that is, the answer $\mathbf{x}^*$ will be the minimum of $\|A\mathbf{x} - \mathbf{b}\|$:

$$\|A\mathbf{x}^* - \mathbf{b}\| \leq \|A\mathbf{x} - \mathbf{b}\|, \text{ for all } \mathbf{x}$$

In general, you should always use the forward slash (for help, type `help slash`): `x=A\b` which automatically determines a best numerical method. That is, sometimes its best (numerically) not to explicitly compute the pseudoinverse first.

4 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.

4.1 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!

4.2 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.

4.3 Exercises

1. What is the Matlab command to create the array x which holds the integers: $2, 5, 8, 11, \ldots 89$

2. (Referring to the array above) What would the Matlab command be that zeros out the even-numbered indices (That is, $x(2), x(4), x(6), \ldots$)?

3. What is the difference in Matlab between typing: `x=[1 2 3]` and `x=[1,2,3]` and `x=[1;2;3]`? What happens if you type a semicolon at the end of the commands (i.e., `x=[1 2 3];`)?

4. (Referring to the last question) For each of those, what happens if you type `x.^2+3`? What happens if you forget the period (e.g., `x^2+3`)

5. What do the following commands do: `x=2;3;6;`, `x=2:3:6;`, `a=pi:pi:8*pi;`

6. Describe the output for each of the following Matlab commands. Recall that typing a semicolon at the end of the line suppresses Matlab output- to see the results, leave off the semicolon.

```
A=rand(3,4);
A([1,2],3)=zeros(2,1);
B=sin(A);
C=B+6;
D=2*B';
E=A./2;
F=sum(A.*A);
```

7. What will Matlab do if you type in:

```
A=rand(3,4);
A(:)
A(7)
```

   NOTE: This is very bad programming style! Don't do it unless you know what you're doing!!

8. What is the Matlab command to perform the following:

   (a) Given an array x, add 3 to each of its values.

   (b) Given an array A, remove its first column and assign the result to a new array B.

9. What will the following code fragment do?

```
a=1:10;
for k=1:10
  h=ceil(length(a)*rand);
  b(k)=a(h);
  a(h)=[];
end
```

Compare this with `a=ceil(10*rand(10,1))` and `a=randperm(10)`

10. Let x be a row. What happens if you type `plot(x)`?

11. Let A be a 4×3 matrix. What happens if you type `plot(A)`? Compare this with `plot(A')`.

12. Write a Matlab command to plot $y = \mathrm{e}^{5x}$, where $-2 \leq x \leq 2$ using 30 points. Plot both the curve and the actual data points themselves, both in magenta.

13. Write a Matlab function to plot $y = \sin(x)$ in red, $y = \sin(2x)$ in black, and $y = \sin(3x)$ in green, all on the same plot. You can assume that $x \in [-4, 8]$.

5 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.

Both scripts and functions should have the `.m` file extension. We'll get more practice with these a little later.

5.1 Debugging Hints

Sometimes when we write a Matlab function, we'll want to stop its execution to see what the function is actually doing. There are a couple of ways to do this- one way is to use the debugger in Matlab's editor. The method below is also very useful.

The **keyboard** command temporarily halts the execution of a function and returns control to the keyboard, and you can use this to see what's happening in a function. To return the control back to Matlab, you would type **return**.

For example, let's turn the Newton's method script file into a function. Edit the file we created earlier as `newton1.m` so that it looks like this:

```
function [z,y,dy]=newton2(x,n)
%FUNCTION [y,dy]=newton2(x,n)
% performs Newton's Method on x*exp(x)-cos(x)
% using initial value x and n iterations.
% The output z gives the refined solution,
% the output y gives the function values and dy
% the corresponding derivatives.

for k=1:n
  y(k) = x*exp(x)-cos(x);
  dy(k) = (x+1)*exp(x)+sin(x);
  z(k) = x-(y(k)/dy(k));
  x = z(k);
end
keyboard
```

Now save the file as `newton2.m`. In the command window, type `help newton2`. You should see the help lines come up. Now run the function by typing in the command window: `[x,y,z]=newton2(0.2,5);`

The program should stop at the keyboard line. Type in `whos` to see the current active variables. NOTICE that the cursor in the command window has changed to `K>>`. This indicates that the keyboard command is active. Type in `return` to get back.

Something to think about: What will Matlab do if you call

```
[x,y,z]=newton2(0.2,5);
```

and then type in `[x,y,z]=newton2(0.2,5);` at the `K>>` prompt??

Other Hints:

- Always know what the matrix sizes are. If you think the input to a function is a column, you can ensure that it is either by checking the size with `size(x)` or by typing: `x=x(:);`

- You can also type the command: `dbstop if error`

  This will stop Matlab execution and point you to the line in your code where the error occurred. To turn this off, type `dbclear if error`

  To see more options, type `help dbstop`