1 Introduction to Matlab

1. **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.

2. **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.
>>
```

3. **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 seperate 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 seperate text file (called an **m-file**).

4. **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.

5. 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.

2 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. To take the number e to a power, use the functional form: 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`.

3 Arrays

In our computations, we will be producing large sets of numbers that we will want the computer to remember. This is what an array is used for: *An array is a set of storage spaces that can be used to store numbers.* (Side remark: If you've seen matrices, an array is the computer equivalent of a matrix).

3.1 One dimensional arrays: Rows and Columns

For example, suppose I want the computer to remember the numbers: $1.1, 4, 3.2, 5$. In Matlab, I would type:

```
x = [ 1.1, 4, 3.2, 5];
```

In this case, we say that the variable x is a one-dimensional array of 4 numbers. The semicolon at the end of the command tells Matlab not to echo out the command. See what happens if you leave the semicolon off. (HINT: You don't have to re-type the numbers, just use the "up" arrow on the keyboard!)

To access the i^{th} number from an array, use parentheses. For example, if I want the third number from the array named x above, I would type: `x(3)`

If I want to change the second number in the example from a 4 to a 7, I would type: `x(2)=7`

Matlab is also sensitive to how the data is typed in. In the first example, we typed our data in as a *row*, but we could have produced a column:

```
x = [ 1.1; 4; 3.2; 5];
```

The semicolon inside the brackets tells Matlab that this is a column instead of a row. Accessing column elements looks the same as accessing row elements; that is, the third element of the column is still `x(3)`.

Some special commands associated with one dimensional arrays:

- `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. 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.

3.2 Two dimensional arrays

The one dimensional array is really a special case of a two dimensional array. A two dimensional array is a table of data. For example, we might have a table of 4 rows of data, each 2 columns long, which we could enter into Matlab as:

```
x = [ 1.1, 4; 3.2, 5; 1, 3; 4, 2.1];
```

or

```
x = [ 1.1, 4
3.2, 5
1, 3
4, 2.1];
```

In either case, we say that the *size* of the array is 4×2, read as "Four by Two". So, one dimensional arrays are either $1 \times n$ (for a row of n elements), or $n \times 1$ (for a column of n elements).

NOTE: The array is the basic data type on which Matlab operates.

To access an element of a two dimensional array, you must specify two numbers: The row and the column. In our example above, the number 2.1 is in the 4th row, 2d column, so its position is $(4, 2)$. The number 3.2 is in the 2d row, 1st column, so its position is $(2, 1)$.

In general, the (i, j) element is accessed by writing `x(i,j)`. For example, if we want to change the $(3, 2)$ position from its current value of 3 to 7.212, we would type: `x(3,2)=7.212;`

It is possible to define multidimensional arrays, but we will only use 1 and 2 dimensional arrays.

3.3 Matlab commands associated with Arrays

1. Creating Arrays.

   - `A=rand(m,n)` Produces an $m \times n$ array of random numbers between 0 and 1.
   - `A=zeros(m,n)` Produces an $m \times n$ array of zeros.
   - `A=ones(m,n)` Produces an $m \times n$ array of ones.

2. Manipulating Arrays.

   - Transposition. Transposition changes rows to columns, and vice versa. It is denoted by the single quote character `'`. For example, if x is a row of n elements, then `x'` is a column of n elements. If A is an $m \times n$ array, then A' is an $n \times m$ array. It does this by making the first column of the old matrix the first row of the new matrix, and so on. Try it with a small matrix.
   - Addition and Subtraction. These operations are defined **only** for arrays of the same size. If A and B are $m \times n$, then $A + B$ is also $m \times n$. The new array is obtained by adding the terms, element by element, of the old arrays. For example, using Matlab notation:

     ```
     A=[1, 2, -1; 3, 4, 0];
     B=[4, -1, 0; 2, 1, 1];
     ```

     Then $A + B$ is a 2×3 array:

     $$\begin{bmatrix} 1+4 & 2-1 & -1+0 \\ 3+2 & 4+1 & 0+1 \end{bmatrix} = \begin{bmatrix} 5 & 1 & -1 \\ 5 & 5 & 1 \end{bmatrix}$$

   - Scalar addition. If we want to add a constant to every item in an array, we do it in the usual way. In this example, A is an $m \times n$ array, and we add 5 to every element: `A+5`

- 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`
- Array Multiplication. We can multiply and divide the elements of an array A and an array B elementwise (so A and B must be the same size): `A.*B` and `A./B` (NOTE: see that we used `.*` and `./`. This is because the regular `*` and `/` will have special meaning.)

  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.

  We can apply regular functions to arrays. For example, if x is the array containing the numbers $1, 2, 3, 4$, then `sin(x)` is the array containing the numbers $\sin(1), \sin(2), \sin(3), \sin(4)$. Similarly, `exp(x)` is an array containing the numbers e^1, e^2, e^3, e^4.

  If we apply the function $x^2 + 3x + 5$ to the array A, the Matlab command would be:

  ```
  A.^2+3*A+5
  ```

3. Accessing Blocks of Values.

   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
$A(2 : 5, :)$	The rows 2-5, all columns
$A(:, j)$	The entire jth column
$A(:, 2 : 5)$	The 2d to fifth columns, all rows
$A(1 : 4, 2 : 3)$	A 4×2 submatrix

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

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

   Answer: A 3×2 matrix consisting of the elements are:

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

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

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

I would use the follows sequence of commands:

```
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.

4. 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):

   - `A(1,:)=[];` Delete the first row.
   - `A([1,3],:)=[];` Delete rows 1 and 3.
   - `A(:,3)=[];` Delete the third column.
   - `A(:,1:2:5)=[];` Delete the odd columns.
   - `A(1,:)=b;` Put b as the first row.
   - `A(:,6)=c;` Add c as the last column.
   - `d=[c , A(:,1:3)];` Create a new array from c and the first three columns of A.
   - `A=[A(:,1), c, A(:,2:5)];` Insert c as the second column of A, shift the other columns over.
   - `A=[A(1,:); b; A(2:4,:)];` Insert b as the second row of A, shift the other rows down one.

     (NOTE: Note the difference in the use of comma or semicolon in the last two items!)

3.4 Exercises with Arrays

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 (Try it!) 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];`)? For each of those, what happens if you type `x.^2+3`? What happens if you forget the period (e.g., `x^2+3`)

4. Describe what each of the following sets of Matlab commands does (by typing them in). Recall that typing a semicolon at the end of the line suppresses Matlab output. To see the results, leave off the semicolon.

   (a) `a=pi:pi:8*pi;`

   (b)
   ```
   A=rand(3,4);
   A([1,2],3)=zeros(2,1);
   B=sin(A);
   C=B+6;
   D=2*B
   ```

   (c)
   ```
   A=randn(2,4);
   B=3*A';
   ```

   (d)
   ```
   A=ones(3,3);
   B=A./2
   ```

   (e)
   ```
   A=[1,2,3;4,5,6];
   B=sum(A.*A);
   ```

5. 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.

   (c) If A is an array, what will `sin(A)` be?

4 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`.

5 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.

5.1 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!

5.2 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.

5.3 Exercises

1. Let x be a row. What happens if you type `plot(x)`?
2. Write a Matlab command to plot $y = \mathrm{e}^{5x}$, where $-2 \leq x \leq 2$.
3. 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 $x \in [-4, 8]$.

6 M-Files: Functions and Scripts

1. 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 a text 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).

2. Multiple outputs: 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.

3. **A Script File** A script file differs from a function file in that a script file is a sequence of commands that are executed as if you were typing them in at the keyboard. Therefore, all variables stay in accessible storage (unlike when calling a function).

   You should use script files for your homework problems- They give you a record of the commands you used, and also make it easy to make small changes. Here's a sample:

   Suppose my homework assignment is to use Matlab to produce plots of the sine and cosine functions, with a title and a legend. Then my script file would look like:

```
x=linspace(-3,7);
y1=sin(x);
y2=cos(x);
plot(x,y1,x,y2);
title('Homework Problem 1');
legend('Sine','Cosine');
```

I would type the commands into a text editor, then save it using the `.m` extension. For example, I might save it as `hw1.m`

To run the script, in Matlab just type the filename. In this case, I would just type `hw1` at the Matlab prompt.

7 Function Iteration

In our class, it will be important to be able to do function iteration. This will be accomplished using what's called a "for" loop. In words, we'll want to do something like:

```
Do the following set of commands 10 times:
  command;
  command;
  command;
End of the set.
```

In Matlab, this is accomplished by using the following syntax:

```
for i=1:10
  command;
  command;
  command;
end
```

The variable i is called the *index* of the loop. The computer uses this to keep track of what iteration it is currently on. For example, try typing this in:

```
for i=2:7
  i
end
```

When Matlab runs this, it outputs the integers from 2 to 7.

Example: Use a FOR loop to iterate the function $y = \sqrt{x}$ 10 times, using $x = 0.99$.

```
x=0.99;
for i=1:10
  x=sqrt(x)
end
```

The problem with that solution is that we lose all of the intermediate values of the function. A better way might be:

```
x=0.99;
for i=1:10
  y(i)=sqrt(x);
  x=y(i);
end
```

Now the array y holds all 10 numbers that were produced.

Example: Iterate the function $x^2 + 1$ 10 times using the initial conditions $x = -1, x = 0.3, x = 0.9, x = 1.2$ and plot the resulting orbits.

We could run a script similar to the previous example, but we can also do everything at once. Instead of producing an array y for each condition, we'll make y a 10×4 array!

```
x=[-1, 0.3, 0.9, 1.2];
for i=1:10
  y(i,:)=x.^2+1;
  x=y(i,:);
end
%plotting routines:
t=1:10;
plot(t,y(:,1),t,y(:,2),t,y(:,3),t,y(:,4));
legend('x=-1','x=0.3','x=0.9','x=1.2');
title('Orbits for Example');
```

The first column holds the 10 numbers for the iteration starting with $x = -1$, the next column holds the numbers for $x = 0.3$, and so on. Subsequently, we plotted these values.