A Short Introduction to Matlab

Sheng Xu & Daniel Reynolds
SMU Mathematics, 2015

1 What is Matlab?

Matlab is a computer language with many ready-to-use powerful and reliable algorithms for doing numerical computations. You can use it in an interactive manner like an advanced calculator, or you can build up your own set of functions and programs. Excellent graphics facilities are also available for visulaizing your results.

2 Accesssing Matlab

The simplest way for students to access Matlab is through `http://apps.smu.edu`. Here, SMU hosts a cloud-based version of Matlab, that may be used from Windows, OS X and even Linux computers (although the setup in Linux is challenging). Be careful about how you save files in virtual machine, since the "disk" available to Matlab in this environment may not be on your personal computer.

Alternately, as of the Spring 2015 semester, the following public SMU computer labs have Matlab installed:

- Junkins 342 (Engineering login)
- Junkins 344 (Engineering login)
- Embrey 124 (Engineering login)
- Caruth 116 (Engineering login)
- Fondren Library West 103b (general SMU login)
- Fondren Library West 103c (general SMU login)
- Fondren 124 (general SMU login)
- Heroy 126 (general SMU login)
- LEC (general SMU login)
- Umphrey Lee 301 (general SMU login)
- Clements G15 (general SMU login)
- Junkins 102 (Engineering login)

Some of the above computer labs (Clements G15, Junkins 102, etc.) are only accessible for classroom use, and are not generally available. Those marked as "General SMU login" require your standard SMU ID and SMU email password. Those labs marked as "Engineering login" require an engineering account (all Lyle students should already have an engineering account).

You may purchase and download a personal copy of Matlab at

`https://www.mathworks.com/academia/student_version/`

The student version costs $99, and is available for Windows, Mac, and Linux computers. If you do this, you will not need to purchase any of the additional Matlab toolboxes for this course (all available for an additional fee), so be careful not to check any extra boxes. Since many of you will be using Matlab for future Math and Engineering courses as well, it may be a good idea to purchase it now instead of later.

3 Matlab Tutorials

You can take online Matlab tutorials at

`http://www.mathworks.com/academia/student_center/tutorials/`

These interactive tutorials are deigned for students and are *free*. Please register to start learning Matlab on your own pace.

There are also many books with instructions on using and programming in Matlab. It may be useful to buy one to keep it as a reference.

4 Starting Matlab

On Windows systems, Matlab is started by double-clicking the Matlab icon (see Figure 1 below) on the desktop or by selecting Matlab from the start menu. On OS X systems, Matlab is installed in the main `Applications` folder, and can be opened by double-clicking the Matlab icon there. On Linux or UNIX systems, Matlab may be started from any directory by typing `matlab &` in the terminal.

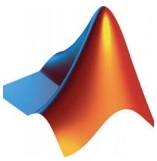

Figure 1: Matlab icon.

On all systems, starting up Matlab will take you to the Command window where the Command line is indicated with the prompt `>>`. You can use Matlab like a calculator by entering and executing Matlab commands in this window. Later, you will learn how to write a Matlab program and run your program like a command.

5 Matlab as an Advanced Calculator

You can use Matlab as an advanced calculator in the Command window by typing commands at the `>>` prompt. Try out each of the follows and see what happens.

```
>> 1+4/2*5
>> 1+4/(2*5)
>> 1+(4/2)*5
>> 9^2
>> sqrt(81)
```

```
>> exp(0)
>> pi
>> sin(pi/2)
>> cos(pi/2)
>> tan(pi/4)
>> cot(pi/4)
```

Note that in the above commands, `pi` is used by Matlab for π, and Matlab's trigonometric functions are permanently set to radians mode.

Here is a useful tip. You can bring up a previous comand by hitting the up-arrow key $\uparrow$ repeatedly until the command you want appears at the `>>` prompt. You can then use keys $\leftarrow$, $\rightarrow$, and 'Delete' to edit the command. If you don't want to cycle through every command you've used, you can start typing the same command and then use the up-arrow to cycle through only your previous commands that started with the same letters.

6 Order of operations

Matlab follows the standard mathematical order of operations in performing calculations. In order of precedence, these are:

- operations in parentheses, `()`
- exponentiation, `^`
- multiplication and division, `*` and `/`
- addition and subtraction, `+` and `-`

For example, try the following operations

```
>> 4 - 2 * 3^2
ans =
   -14
>> (4 - 2) * 3^2
ans =
    18
>> 4 - (2 * 3)^2
ans =
   -32
>> (4 - 2 * 3)^2
ans =
    4
```

7 Variables

Not all calculations can or should be performed on a single line; instead it may be helpful to store the results of calculations:

```
>> 1+4/2*5
ans =
```

```
   11
>> ans*5
ans =
   55
```

As you can see from the above results, if no name is given to a result, Matlab uses the variable `ans` to store the result.

You can use a custom name for a variable. For example

```
>> a = 1+4/2*5
a =
   11
>> A = a*5
A =
   55
>> a
a =
   11
```

As shown above, Matlab is case-sensitive, in that the variable `a` is different from the variable `A`. Also as seen above, you can use a stored result in subsequent calculations.

These are examples of assignment statements: values are assigned to variables. Each variable must be assigned a value before it can by used on the right of another assignment statement. Please use meaningful names for your variables, such as `weight`, `postion`, `balance` etc., and avoid using names that correspond with Matlab's built-in functions and variables, such as `pi` and `cos`. Finally, Matlab variable names **must not** include spaces; if you wish to separate portions of a variable name you may use an underscore (_), as in `x_velocity`.

You can use the Matlab command `whos` to show you the active variables and their sizes currently used in Matlab.

```
>> whos
  Name      Size            Bytes  Class     Attributes

  A         1x1                 8  double
  a         1x1                 8  double
  ans       1x1                 8  double
```

8 Suppressing Output

You may not want to see the results of intermediate calculations, especially if those calculations occur repeatedly within loops (more on loops later). If this is the case, you can suppress the output of an assignment statement or an expression by adding a semi-colon `;` at the end of the statement or expression. For example,

```
>> x = 11; y = 5*x, z = x^2
y =
   55
z =
   121
```

With `;` the value of `x` is hidden. Without `;`, the values of `y` and `z` are displayed. Note also that you can place several statements on one line separated by commas or semi-colons.

9 Keeping a Record

Issuing the `diary` command,

```
>> diary mysession.txt
```

will cause all subsequent text that appear on the screen, including commands and their outputs, to be recorded into the file `mysession.txt` under the current working directory. If no filename is supplied, this information will be sent to the file `diary` under the current working directory. The recording may be stopped by the command

```
>> diary off
```

It can be resumed by

```
>> diary on
```

It is recommended that if you wish to create a diary, you should use the file manager buttons at the top of the Matlab desktop to change directories to the location where you want the diary *before* you issue the `diary` command.

10 Matlab Built-in Functions and Getting Help

Many mathematical functions are already built into Matlab and can be directly used as on a calculator. The following is a list of some common functions.

```
sin(x), cos(x), tan(x), cot(x), sec(x), csc(x)
asin(x), acos(x), atan(x), atan2(y,x)
exp(x), log(x), log10(x), log2(x)
sqrt(x), factorial(x), abs(x), max(x), min(x)
```

In the above list, the trigonometric functions preceded by an `a` are the inverse trigonometric functions, i.e. `asin(x)` is the same as `arcsin(x)` is the same as $\texttt{sin}^{-1}\texttt{(x)}$. Also, `log(x)` is the natural log function, while `log10` is the customary base 10 logarithmic function, and `log2` is the base 2 logarithmic function. Finally, `factorial(x)` corresponds with $x!$ and `abs(x)` computes $|x|$.

When using Matlab to write programs, the following functions are also useful:

```
sign(x), ceil(x), floor(x), round(x), mod(x,y)
```

You will see some useful functions associated with vectors and matrices later.

You can learn how to use a function if you know its Matlab name. Just type `help` plus the function name at the prompt `>>`. For example

```
>> help atan2
```

A help document will appear in the Command window telling you how to use `atan2`. For more lengthy and interactive help on a given function, type `doc` at the prompt, e.g.

```
>> doc atan2
```

11 Inputing Vectors and Matrices

The name "MATLAB" stands for "MATrix LABoratory", since it was initially designed to simplify matrix arithmetic. Here is an example of to input a row vector `r`.

```
>> r = [1 3, sqrt(49)]
r =
   1 3 7
```

The entries of the row vector may be separated by either a space or a comma, and are enclosed in square brackets.

Here is an example to input a column vector `c`.

```
>> c = [1; 3
        sqrt(49); 5]
c =
   1
   3
   7
   5
```

The entries of the column vector must be separated by semi-colons or by pressing `Enter` between rows of the vector.

Try the following Matlab commands.

```
>> r = [1; 3; sqrt(49)];
>> length(r)
ans =
    3
>> norm(r)
ans =
   7.6811
```

Note that the Matlab command `length(r)` gives the number of entries in the vector `r`. To compute the mathematical length/magnitude of the vector, you can use the command `norm(r)`, which computes $\|r\|$ as the standard square root of the sum of the squares of each vector entry. The `length` and `norm` functions work in the same manner for row and column vectors.

Here is an example of how to input a 3×2 matrix `A`.

```
>> A = [1 5; 7 9
        -3 -7]
A =
       1    5
       7    9
      -3   -7
```

As with row vectors, entries in each row are separated by spaces or commas. As with column vectors, different rows are separated by semi-colons or by pressing `Enter`.

Try the commands

```
>> A = [5 7 9; 1 -3 -7]
A =
       5    7    9
       1   -3   -7
>> size(A)
ans =
   2   3
```

The command `size(A)` gives a row vector, the first entry shows the number of rows and the second the number of columns in the matrix `A`.

12 Special Matrices

Some matrices are used frequently in scientific computing, so Matlab provides functions to construct such matrices directly. To create a $n \times n$ identity matrix:

```
>> I = eye(3);
I =
       1    0    0
       0    1    0
       0    0    1
```

To create a $m \times n$ matrix of zeros:

```
>> A = zeros(2,3);
A =
       0    0    0
       0    0    0
```

To create a $m \times n$ matrix of ones:

```
>> A = ones(3,2);
A =
       1    1
       1    1
       1    1
```

To create a $m \times n$ matrix full of uniformly-distributed random numbers between 0 and 1:

```
>> A = rand(1,5);
A =
    0.8147    0.9058    0.1270    0.9134    0.6324
```

Finally, to create a $m \times n$ matrix full of normally-distributed random numbers with mean 0 and standard deviation 1:

```
>> A = randn(3,1);
A =
   -1.3077
   -0.4336
    0.3426
```

13 The Colon Notation

Matlab provides a quick approach for creating a row vector with evenly-spaced entries. Try the commands.

```
>> a = 2:5
a =
       2   3   4   5
>> b = 1:2:8
b =
       1   3   5   7
>>
```

Generally, `s:c:e` produces a row vector of entries starting with the value `s`, incrementing by the value `c` until it gets to the value `e` (it will not produce a value greater than `e`, though the last entry may come before `e` in the sequence). If the increment `c=1`, you can omit it and just write `s:e` as in the first example above.

Here is another example:

```
>> -1.4: -0.3: -2
ans =
      -1.4000    -1.7000   -2.0000
>>
```

Note that the increment need not be an integer, nor must it be positive.

Another quick way to generate a row vector with evenly-spaced entries is to use the command `linspace`. For example

```
>> linspace(1.1,1.5,5)
ans =
      1.1000    1.2000   1.3000    1.4000   1.5000
>>
```

In general, `linspace(s,e,n)` generated a row vector with `n` evenly-spaced entries that start with `s` and end with `e` (the increment is computed automatically from these inputs).

14 Accessing Entries in a Vector or Matrix

The i-th entry of either a row or column vector `v` can be accessed through the index `i` by `v(i)`. For example

```
>> a = 2:5
a =
       2   3   4   5
>> a(3)
ans = 4
```

The entry at row `i` and column `j` of a matrix `A` can be accessed as `A(i,j)`. For example

```
>> A = [5 7 9; 1 -3 -7; 6 8 10]
A =
       5    7    9
       1   -3   -7
       6    8   10
>> A(2,3)
ans =
    -7
```

Not only can you retrieve a specific value of a matrix or vector, but this same notation may be used to change vector or matrix values. For example, following the previous commands, we may do

```
>> A(2,3) = 5
A =
       5    7    9
       1   -3    5
       6    8   10
```

You can access a range of entries of a vector or matrix by using index vectors generated by colon notation. For example

```
>> A = [5 7 9; 1 -3 -7; 6 8 10]
A =
       5    7    9
       1   -3   -7
       6    8   10
>> A(2:3,1:2:3)
ans =
       1    -7
       6    10
```

In this example, the accessed rows are specified by the vector `2:3`, which are row 2 and 3, and the accessed columns are specified by the vector `1:2:3`, which are columns 1 and 3.

To access a whole row or column, refer to the following example

```
>> A = [5 7 9; 1 -3 -7; 6 8 10]
A =
       5    7    9
       1   -3   -7
       6    8   10
>> A(2,:)
ans =
       1   -3   -7
>> A(:,3)
ans =
      9
     -7
     10
```

Note that `A(2,:)` retrieves the second row of the matrix `A`, and `A(:,3)` returns the thrid column of the matrix `A`.

We note that the use of colon notation to access subsets of a vector or matrix is called *array slicing*, and is a unique benefit of Matlab over most other programming languages.

15 Matrix Arithmetic

As long as the dimensions of a matrix arithmetic operation are allowable, Matlab can perform matrix operations using natural mathematical notation.

15.1 Scalar multiplication

Scalar multiplication refers to the product of a scalar and a matrix or vector. For example

```
>> A = [5 7 9; 1 -3 -7; 6 8 10]
A =
       5    7    9
       1   -3   -7
       6    8   10
>> -A
ans =
       -5   - 7    -9
       -1     3     7
       -6    -8   -10
>> 0*A
ans =
       0    0    0
       0    0    0
       0    0    0
>> 2*A
ans =
     10   14   18
      2   -6  -14
     12   16   20
```

15.2 Matrix addition

Matrices or vectors may be added or subtracted as long as each object has the same size and shape. For example

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

```
>> A+A
ans =
    10   14    18
     2   -6   -14
    12   16    20
>> A+B
ans =
     6   10    11
     3    0    -1
     2    6    21
>> A-B
ans =
     4    4    7
    -1   -6  -13
    10   10   -1
```

15.3 Matrix multiplication

Matrices and vectors may be multiplied as long as the inner dimensions of the operands agree. For example

```
>> A = [5 7 9; 1 -3 -7], B = [1 3; 2  3; -4 -2], x = [1; 2; 3], y = [4 5]
A =
       5    7    9
       1   -3   -7
B =
       1    3
       2    3
      -4   -2
x =
       1
       2
       3
y =
       4    5
>> A*B
ans =
   -17    18
    23     8
>> A*x
ans =
    46
   -26
>> y*A
ans =
    25   13    1
```

Matlab requires that all matrix arithmetic operations adhere to the correct mathematical rules, meaning that when multiplying two matrices, the number of columns of the first must equal the

number of rows in the second. When you violate this rule, Matlab will issue an error message:

```
>> B*x
Error using  *
Inner matrix dimensions must agree.
```

The error message text is usually red, and Matlab will typically beep on errors.

15.4 Transpose

The transpose of a matrix A, denoted as A^T in linear algebra, is obtained in Matlab by `A'`. For example

```
>> A = [5 7 9; 1 -3 -7]
A =
       5    7    9
       1   -3   -7
>> A'
ans =
       5    1
       7   -3
       9   -7
>> (A')'
ans =
       5    7    9
       1   -3   -7
```

You can combine the transpose with matrix multiplication in interesting ways. For example, you can compute the inner product of the column vectors a and b using the command $a' * b$ (or equivalently $b' * a$). Similarly, you can compute the outer products of these column vectors as $a * b'$ and $b * a'$. For example

```
>> a=[1;2;3]
a =
     1
     2
     3
>> b=[4;5;6]
b =
     4
     5
     6
>> a'*b
ans =
    32
>> b'*a
ans =
    32
>> a*b'
ans =
```

```
     4     5     6
     8    10    12
    12    15    18
>> b*a'
ans =

     4     8    12
     5    10    15
     6    12    18
>>
```

15.5 Entry-wise convenience operations

In addition to supporting standard matrix arithmetic operations, Matlab also supports a variety of convenient operators that are not standard mathematical operations. However, while these do not correspond to correct mathematics, they may be very useful for some programming purposes.

For example, to add a number to every entry of a vector or matrix:

```
>> A = [5 7 9; 1 -3 -7]
A =
       5    7    9
       1   -3   -7
>> A+100
ans =
     105   107   109
     101    97    93
```

If you want to apply entry-wise operations such multiplication, division or power on matrices, you can supply a `.` preceding the arithmetic operator. For example

```
>> A = [5 7 9; 1 -3 -7]
A =
       5    7    9
       1   -3   -7
>> B = [1 7 3;-1  3  7]
B =
       1    7    3
      -1    3    7
>> A.*B
ans =
       5   49   27
      -1   -9  -49
>> A./B
ans =
      5     1    3
     -1    -1   -1
>> A.^2
ans =
```

```
    25   49   81
     1    9   49
```

Note that `A.*B` produces a matrix (with the same size as `A` and `B`) with each entry being the product of the corresponding entries in `A` and `B`.

Here is another example of a "dot" operation.

```
>> r = [1 0.5 0.25 2]
r =
        1.0000   0.5000   0.2500    2.0000
>> r1 = 1./r
r1 =
        1.0000   2.0000   4.0000    0.5000
```

This example shows how to generate a vector with each entry being the reciprocal of the corresponding entry of another vector.

16 Outputing/Plotting Results

16.1 `disp`

The Matlab function `disp(x)` displays the array `x`, without printing the array name. If `x` is a string, the text is displayed. For example

```
>> r = [1 0.5 0.25 2];
>> disp(r)
   1.0000   0.5000   0.2500    2.0000

>> disp('Hello!')
Hello!
>> s = 'Hello again!';
>> disp(s)
Hello again!
```

16.2 `fprintf`

The Matlab function `fprintf` writes data in a controlled format on the screen (or into a file). If you have ever used the `printf` command in C or C++, the syntax in Matlab is identical. For example

```
>> balance = 50;
>> fprintf('The balance is %d $. \n',balance)
The balance is 50 $.
>>
```

In this example, the value of the variable `balance` is shown in the format controlled by `%d` (meaning decimal number) at the location of the format control `%d`. The special format `\n` means linefeed (another subsequent output will be on a new line). The other texts (except `%d` and `\n`) are shown as they are within the quotation marks. Some other often-used formatting specifiers include:

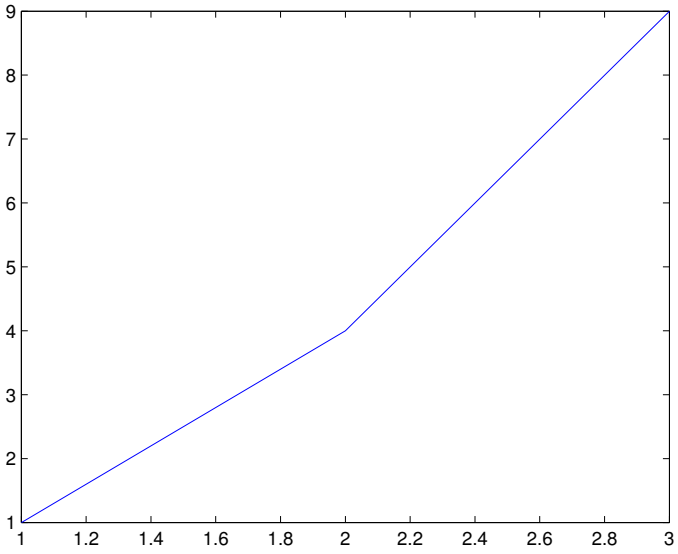

Figure 2: An x-y plot of three points (1,1), (2,4) and (3,9) with Matlab's default color, symbol and line style.

- `%d` – scientific notation for decimal numbers, normal notation for integers
- `%e` – decimal number with seven significant digits, using scientific notation
- `%f` – decimal number with six digits after the decimal, using fixed-point notation
- `%g` – most appropriate notation for decimal numbers (six significant digits), normal notation for integers
- `%i` – same as `%d`
- `\t` – horizontal tab

For more information, type `help fprintf` or `doc fprintf`, or look this up on the web.

16.3 `plot`

The Matlab function `plot` produces x-y plots. For example

```
>> a = [1 2 3];
>> b = [1 4 9];
>> plot(a,b)
```

The plot in Figure 2 is generated. What the Matlab function `plot` does in the example is to draw a line that connects the three points (1,1), (2,4) and (3,9) on an x-y coordinate system. The x-coordinates of the points are given by the vector `a`, and the y-coordinates of the points are given by the vector `b`.

You can change in what shape the points are drawn, how they are connected, and what color is used (by default, the points are drawn as dots and are connected by solid lines in blue). For example

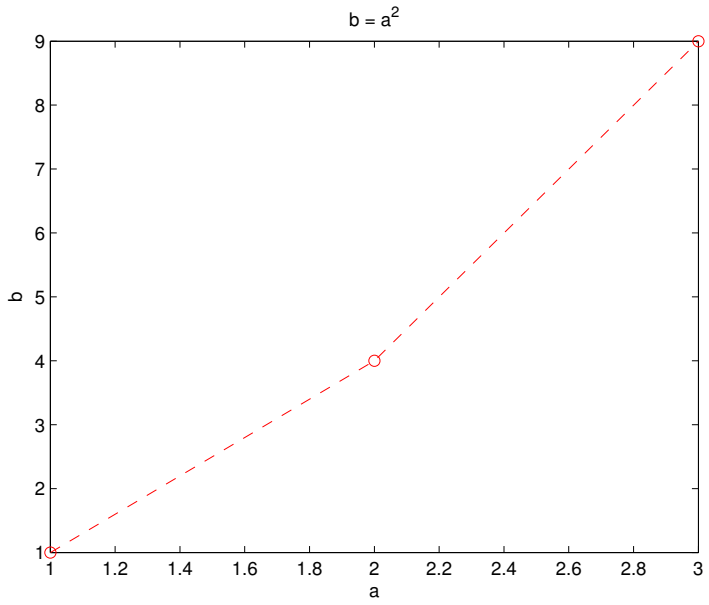

Figure 3: An x-y plot of three points (1,1), (2,4) and (3,9) with changed color, symbol and line style, and added labels and title.

```
>> a = [1 2 3];
>> b = [1 4 9];
>> plot(a,b,'r--o')
>> xlabel('a')
>> ylabel('b')
>> title('b = a^2')
```

The added string `'ro--'` in the arguments of `plot` changes the:

- color to red with the `'r'`,
- symbol to circle with the `'o'`, and
- line style to a dashed line with the `'--'`.

When plotting data, it is always appreciated if you add labels to the x and y axes, and add a title to the plot, as shown in this example. See the effect in Figure 3.

The final example below demonstrates one way to draw multiple curves in a single plot. Pay attention to `legend`, which is used to name and distinguish each curve from one another.

```
>> t = linspace(-pi,pi,200);
>> s = sin(t);
>> c = cos(t);
>> plot(t,s,'b-',t,c,'r--')
>> xlabel('t')
>> legend('sin(t)','cos(t)')
```

The resulting plot is shown in Figure 4.

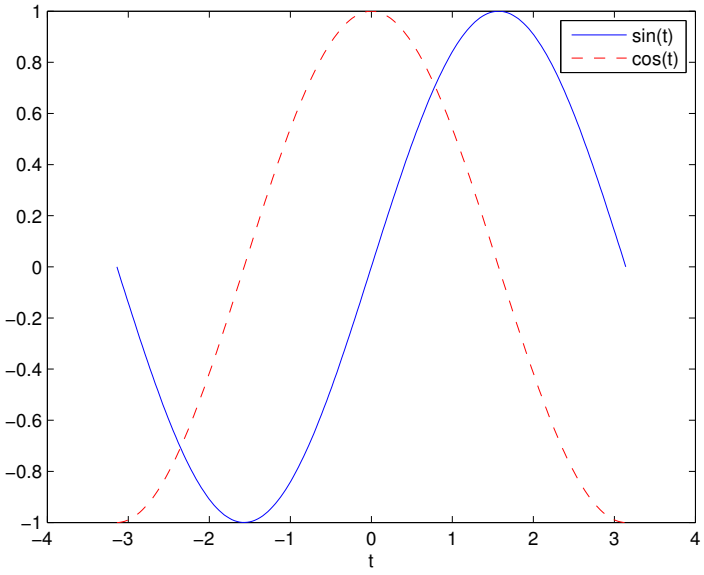

Figure 4: Multiple curves on an x-y plot.

17 `for` Loops

The primary way to do a repeated task in any programming language is through a loop. The idea with a loop is that it will do a task repeatedly at every *iteration* of the loop.

The main loop structure in Matlab is the `for` loop. Below is a simple example:

```
>> for i=1:2:5
fprintf('the value of i at the current step is: i = %d\n',i)
end
the value of i at the current step is: i = 1
the value of i at the current step is: i = 3
the value of i at the current step is: i = 5
>>
```

Note that the first line of the loop begins with the keyword `for`. The contents (i.e. the command `fprintf` in this example) of the loop follow this first line, and cease just before the keyword `end`. The syntax `for i=1:2:5` indicates that the variable `i` takes in turn each value in the vector `1:2:5`, and the contents of the loop are repeatedly executed for each value of `i`.

Below is another example:

```
>> v=[50 40 30 20];
>> for k=v
       disp(k)
   end

    50

    40
```

```
    30

    20
>>
```

Now let's use a loop to compute the sum $100 + 99 + 98 + \cdots + 2 + 1$. The trick here is to define a variable, say s, and then add in turn each entry in the array $100 : -1 : 1$ to s. The code is

```
>> s = 0;
>> for i=100:-1:1
       s = s+i;
end
>> disp(s)
        5050
>>
```

Note that `s` is initialized to be zero (`s = 0`), and then it is updated with the command `s = s+i` by adding the current value of `i` (taking one by one from the array `100:-1:1`) to the current value of `s`. You should understand this trick and know how to use it.

Suppose you need to generate an $m \times n$ matrix H with the entry $h_{ij} = 1/(i + j - 1)$. You can do using nested for loops as follows:

```
>> m=3;
>> n=4;
>> H=zeros(m,n);
>> for i=1:m
      for j=1:n
          H(i,j)=1/(i+j-1);
      end
   end
>> H
H =

    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
>>
```

18 Logicals and Decisions

As with other programming languages, Matlab supports logic statements:

```
>> k = 1; l = 2;
>> k == l
   ans =
        0
>> k ~= l
```

```
   ans =
        1
>> k < l
   ans =
        1
>> k > l
   ans =
        0
>> k <= l
   ans =
        1
>> k >= l
   ans =
        0
>> ((k==k) && (l==l))
   ans =
        1
>> ((k~=l) || (k>l))
   ans =
        1
```

In the above statements, "true" equates to `1` and "false" equates to `0`. The comparison operations are

- `==` "is equal to"
- `~=` "is not equal to"
- `<` "is less than"
- `>` "is greater than"
- `<=` "is less than or equal to"
- `>=` "is greater than or equal to"

Moreover, logic statements can be combined with `&&` (and) or `||` (or).

Sometimes whether some commands will be executed depends on certain conditions. The `if` statement can help in this situation. For example, suppose a bank applies 5% annual interest rate on an account if the account balance is above 5000$. Otherwise, the interest rate is only 2%. To compute the new account balances for a set of four accounts after one year, the bank can use a Matlab code as:

```
>>  balance = [6000 5000 20000 200];
>>  for i = 1:length(balance)
       if balance(i)>5000
          balance(i)=balance(i)*(1+0.05);
       else
          balance(i)=balance(i)*(1+0.02);
       end
   end
```

```
>> disp(balance)
        6300        5100       21000         204
>>
```

The general form of the `if` statement is

```
if logical test 1
   Commands to be executed if test 1 is true
elseif logical test 2
   Commands to be executed if test 2 is true but test 1 is false
.
.
.
end
```

For complicated logical statements, it is standard to enclose the logic test in parentheses.

19 `while` Loops

Sometimes you want to repeatedly execute a section of codes until a condition is satisfied, but you do not know in advance how many times you will need to repeat the codes. Since a `for` loop needs to know how many times it will be evaluated beforehand, Matlab supplies another (more advanced) looping mechanism, a `while` loop. For example,

```
>> i=0;
n=11;
while n<100
      n=3*n;
      i=i+1;
      fprintf('at step i = %d: n = %d\n',i,n)
end
at step i = 1: n = 33
at step i = 2: n = 99
at step i = 3: n = 297
```

Note that the commands of the `while` loop are executed 3 times, as shown by the counter `i`, and when the loop finishes, $n = 297$ and the condition `n<100` is no longer satisfied.

20 Making M-Scripts/M-Files

You can store your Matlab commands (such as assignment statements, loops, and other statements) into files called m-scripts. You can then run your files again and again at any time you want to. Basically, your files act as lists of Matlab commands, where the Matlab commands you type into your m-script are executed in the order from top to bottom. An m-file is very useful because it allows you to pack many commands with logical order into one file to fulfill a complicated computational task. Matlab treats m-scripts as if they were typed into the Matlab command window, so any variables included inside a m-script will remain in the Matlab namespace when the m-script finishes. This is Matlab programming. You will learn more about programming later.

To create a new m-script file, click the icon 'New M-File' on the tool bar or select the menu 'File→New→M-File' from the menu list. To edit an old m-script file, click the icon 'Open file' on the tool bar or select the menu 'File→Open'. These operations will open the Matlab Editor for you to input or edit comands. Note that a m-script file must be saved with a ".m" extention. For example, `test.m`. You can run your file (i.e. the commands contained in the file) by typing `test` at the prompt `>>` in the Command window.

Alternatively, a Matlab m-file is just a plain text file with a ".m" extension, so if you prefer you can use a different text editor than the built-in Matlab one.

An m-script can be published to a webpage as a .html file by selecting in the Matlab Editor the menu 'File→Save and Publish to-HTML'. The webpage includes the commands in the m-script followed by the displayed results of these commands.

Lastly, any line in a m-file starting with % is just a comment that is not executed by Matlab. These should be used to help others and yourself understand your code. Similarly, a line starting with %% will show up as a link at the beginning of the published webpage.

21 Function M-Files

You have seen some built-in Matlab functions such as `sin`, `cos` and so on. If you want to use `sin` to compute $\sin(\pi)$, you can do the following

```
>> y=sin(pi)
y =
   1.2246e-16
```

In the above example, the function has the name `sin`, it receives the value `pi` as an input in the parentheses following its name, and it returns the result to the variable `y`.

Also note that in this example the value of $\sin \pi$ given by Matlab is very close to zero, but not exactly zero. This is due to *floating-point roundoff error*, which you will learn more about in Math 3315.

Matlab enables you to create your own functions called function m-files in a similar way as the above example. For example, the following function m-file is created to find the two roots of the quadratic equation $ax^2 + bx + c = 0$, under the assumptions that $a \neq 0$ and that the equation has real-valued roots:

```
function [x1 x2] = quadroots(a,b,c)
sdelta = sqrt(b^2-4*a*c);
x1 = (-b+sdelta)/(2*a);
x2 = (-b-sdelta)/(2*a);
```

As you can see, unlike a Matlab script, the function m-file starts with the keyword `function`. Here, the created function is named `quadroots` – you should use other meaningful names as you like. The function receives the coefficients `a`, `b` and `c`, and it returns the two roots `x1` and `x2`, which are listed between the square brackets at the left-hand side of the equal sign. The main body (the codes after the first line) of the function m-file applies formulas to compute the two roots using the received coefficients. This function m-file must be saved with the file name `quadroots.m`, i.e. the root of the file name must be the same as the function name, such that other m-scripts or function m-files know where (which file) to find the codes when they call/use the created function by the name `quadroots`. Furthermore, as with Matlab variable names, Matlab function names **must not** contain spaces.

After the function `quadroots` is created and saved, you can test it by using another m-script, as long as both of the m-files are located in the same directory on the computer. Below is an example of a test m-script, which can be saved as a file with any name, e.g. `testquadroots.m`

```
aa=1;
bb=0;
cc=-4;
[root1 root2] = quadroots(aa,bb,cc);
fprintf('the roots are: %d and %d\n',root1,root2)
```

Note that when using `quadroots`, the input variables (`aa`,`bb` and `cc`) passed to it can have different names as those in the input list (`a`, `b` and `c`) of the function m-file above, but they must have been assigned values and must be in the correct order (i.e. `aa` corresponds to `a` and so on). Likewise, the returned values of the roots are assigned to the variables `root1` and `root2` in the order determined by the function m-file, but the names of the variables these are assigned to can be different from the names in the function m-file.

Unlike a m-script, the variables declared inside a function m-file exist only within that function (unless they are passed in or returned from the function). This means that in the above example, although `testquadroots.m` declared the variables `aa`, `bb` and `cc`, the `quadroots` function itself cannot reference variables with those names, and must instead reference the variables that were passed in (`a`, `b` and `c`). Similarly, although `quadroots` created a local variable `sdelta`, since it was not included in the output list for the function, `testquadroots.m` cannot access that value since it only existed within the scope of `quadroots`.

A function m-file has the following general form:

```
function [output1, output2, ...] = name(input1, input2, ...)
...
output1 = ...;
...
output2 = ...;
...
```

22 Anonymous Functions

Many functions do not require much space, or you may want them to change as a program proceeds. In such cases, creating an entire function m-file to hold the small function, or creating many function m-files to hold each version of a function, may be unnecessary. In such situations, Matlab allows you to declare an *anonymous function*. These may be defined in a script m-file, a function m-file, or even in an interactive Matlab session. For example, to create a simple function that implements $g(x) = \cos(\pi x)$ you can use the Matlab code

```
g = @(x) cos(pi*x);
```

This function will persist in the namespace of the function, script or session where it is defined, similar to a Matlab variable. A few rules about anonymous functions:

- The entire function contents must consist of a single line of execution.
- The function name is listed at the left of the equal sign to the left of the ampersand.
- All arguments to the function must be listed in the parentheses following the ampersand.

- The function cannot create temporary variables.
- The function can see and use any other variables defined previously within the file.
- While it is possible for anonymous functions to return multiple outputs, it is somewhat challenging and is not recommended.

For example, here is a more complicated example using anonymous functions:

```
a = 2;
b = -3;
c = 1;
d = 8;
f = @(x,y) a*(x-b)^2 + c*(y-d)^2;
clear a b c d
x = linspace(-5,-1,100);
y = linspace(4,12,200);
z = zeros(100,200);
for i=1:100
   for j=1:200
      z(i,j) = f(x(i),y(j));
   end
end
surf(y,x,z)
```

Because `a`, `b`, `c` and `d` were available at the time `f` was defined, the anonymous function handle includes those values *even though they were deleted prior to using* `f`.

Concluding Remarks

Matlab is a powerful computing environment that has been built up over many years as a tool for scientific computing. As a result, no tutorial can give an exhaustive account of all Matlab functionality, though we believe that this tutorial should be enough to help you get started. As with any technology tool, the best way to learn is through practice, through sharing tips with others, and through reading the documentation. Good luck!