AN INTRODUCTION TO MATLAB

MELVIN LEOK

MATLAB is an interactive environment for numerically manipulating arrays and matrices, as well as providing tools for visualizing data. It is particularly appropriate for implementing simple numerical algorithms as it provides the necessary data structures, matrix operations, and visualization tools, thereby allowing one to concentrate on the algorithmic structure of the numerical methods we study.

Scalars, Arrays, and Matrices

Scalars: To assign a scalar value to a variable,
`>> x = 0.2`
By default, MATLAB will echo the value you entered,
```
x =
0.2000
```
unless you suppress output by adding a semicolon to the end of your command,
`>> x = 0.2;`

Row vectors: To enter a row vector,
`>> y = [1 2 3]`
where each term in row is separated by either a space or a comma.

Column vectors: To enter a column vector,
`>> y = [1;2;3]`
Notice that the rows are separated by semicolons. Alternatively, we can take the matrix transpose of a row vector,
`>> y = [1 2 3]'`
where the ' denotes the transpose operation.

Matrices: As is the case for row and column vectors, we separate terms in each row by a space, and we separate the rows by semicolons,
`>> A = [1 2 3;4 5 6; 7 8 9]`

Special Matrices: It is often convenient to construct the following special vectors and matrices,

Equally spaced elements in a vector: To create a vector starting at 0 and ending at 2 with intervals 0.5, we enter,
```
>> v = 0:0.5:2
v = 0 0.5000 1.0000 1.5000 2.0000
```
We could also have constructed a linear space that starts at 0 and ends at 2, with 5 evenly spaced entries,
`>> v = linspace(0,2,5);`

Matrix of zeros: `>> Z = zeros(3,5);`

Matrix of ones: `>> X = ones(3,5);`

Identity matrix: `>> I = eye(3);`

Diagonal matrices: If you wish to create a diagonal matrix with entries 1,2,3, we enter,
`>> diag([1,2,3]);`
To create a 4 x 4 matrix with entries 1,2,3 in the diagonal above the main diagonal, we enter,
`>> diag([1,2,3],1);`

Selecting parts of a matrix: Given a matrix,
```
A =
1 2 3
4 5 6
7 8 9
```
To select the entry in row 1, column 2, we enter,
```
>> A(1,2)
ans = 2
```
To select a submatrix, say rows 1 and 2, and columns 2 and 3,
```
>> A(1:2,2:3)
ans =
2 3
5 6
```

Concatenating matrices: Matrices can be combined together. Given a row vector,
`>> v=[1 2];` we can either construct a 2 x 2 matrix,
```
>> [v ; v]
ans =
1 2
1 2
```
or a row 4 vector,
```
>> [v v]
ans =
1 2 1 2
```

Matrix operations: The follows are some matrix level operations in MATLAB,

+	Addition
-	Subtraction
*	Multiplication
^	Power
'	Conjugate transpose

If you would like to operate on each term in the matrix individually, say given a matrix,
`>> A = [1 2;3 4];` we would like to obtain a new matrix consisting of squares of each term, we would enter,

```
>> A.^2
ans =
1 4
9 16
```

By prefixing the `^2` with a period, we cause MATLAB to apply the operation elementwise. In contrast,

```
>> A^2
ans =
7 10
15 22
```

Programming

Relational operators: These operators provide a means of comparing the values of two objects, and return a boolean value.

`==`	Equal
`~=`	Not equal
`<`	Less than
`>`	More than
`<=`	Less than or equal
`>=`	More than or equal

Logical operators: These operators implement boolean gates, and other logical operations.

`~`	NOT
`&`	AND
`|`	OR
`xor`	Exclusive OR
`any`	True if any elements is nonzero
`all`	True if all elements are nonzero

For loops: To loop through a set of commands with an index variable `i` varying from initial value `i0`, final value `i1`, and step `delta`, we have,

```
for i=i0:step:i1
...
end
```

Note that the step can be negative if the final value is less than the initial value. If the step size is `1`, we can use the more compact notation, `for i=i0:i1`.

If statements: These have the form,

```
if (logical-expression)
...
elseif (logical-expression)
...
else
...
end
```

where the `elseif` and `else` terms are optional.

While loops: The while loop executes the inner loop as long as the condition is true.

```
while (while-expression)
...
end
```

Functions: It is often helpful to construct user-defined functions. These are saved as individual M-files, which have the form,

```
function [out1,out2,out3] = f (in1,in2,in3)
...
```

which defines a function `f` that takes `in1, in2, in3` as inputs, and returns `out1, out2, out3` as outputs. The M-file is saved as `f.m`, and it is called by the following command,

```
>> [a,b,c]=f(x,y);
```

Where we notice that we require separate variables `a,b,c` to store the output of the function.

Passing functions to functions: Say we have a function that computes the slope of the line joining the endpoints of another function. This is to say that given $f(x)$, it computes,

$$m = \frac{f(b) - f(a)}{b - a}.$$

It would have to take as one of its inputs the function $f(x)$, which we can do as follows,

```
function m=slope(func,a,b)
fa=feval(func,a);
fb=feval(func,b);
m=(fb-fa)/(b-a);
```

where `feval` allows us to evaluate the function that was passed in at a specific point. Now, say we wish to apply this function to $y = x^2 + 3 * x^3$. We first construct another function,

```
function y=f(x)
y=x^2+3*x^3;
```

We then pass the function `f` to the function `slope` as follows

```
>> slope(@f, 0, 1)
```

Graphics and Visualization

To plot a function, we need to construct two arrays, giving the points on the x-axis, and the corresponding points on the y-axis. Say we wish to plot $\sin(x)$ over the interval $x = 0..\pi$. We first set up the array for x,

```
>> x=linspace(0,pi,50);
```

To evaluate $\sin(x)$ on is array, we simply enter,

```
>> y=sin(x);
```

and to plot the result, we use,

```
>> plot(x,y);
```

To plot both $\sin(x)$, and $\cos(x)$ simultaneously, we would additionally do,

```
>> z=cos(x);
>> plot(x,y, x,z,'o');
```

where the `'o'` is an optional parameter asking MATLAB to plot the second function using circles instead.

To change the viewing axis, we use the `axis` command,

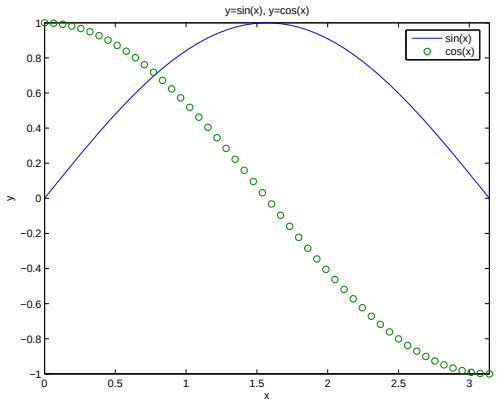
```
>> axis([0 pi -1 1]);
```

A title, axis labels, and legends can be added using,

```
title('y=sin(x), y=cos(x)');
xlabel('x');
```

```
ylabel('y');
legend('sin(x)', 'cos(x)');
```

The resulting plot is shown as follows.



Another approach is to use `fplot`, which allows us to plot a function directly. It is called as follows,

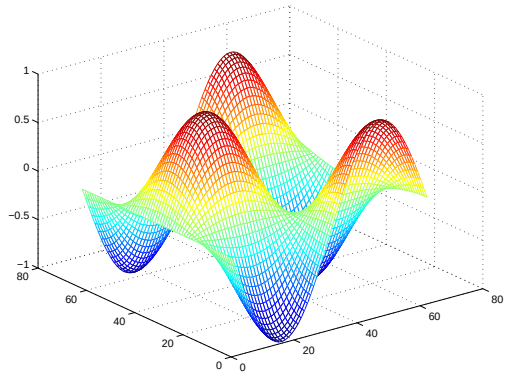
```
>> fplot('sin',[0,pi],50)
```

where the `50` determines the number of sample points to use. It can be omitted, in which case the default is `25`.

To plot in three-dimensions, we need to set up a 2-dimensional mesh as follows,

```
>> x = 0:0.1:2*pi;
>> y = -pi:0.1:pi;
>> [x,y] = meshgrid(x,y);
>> z = sin(x).*cos(y);
>> mesh(z)
```

Note that we needed to use the elementwise multiplication `.*` in defining z.



Additional resources

There are two other more detailed MATLAB tutorials available on the course website. One is a general tutorial by Peter Blossey and James Rossmanith, and the other is an introduction to plotting with MATLAB.

Cleve Moler's book entitled, *Numerical Computing with MATLAB*, is also a useful reference with sample MATLAB codes for many of the numerical methods we will be studying. It is available for download at `http://www.mathworks.com/moler/`

Online help for MATLAB is also available on the web at `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml`, and within MATLAB itself, by entering `help function_name`.

Department of Mathematics, University of Michigan, Ann Arbor, MI 48109, USA.
E-mail address: `mleok@umich.edu`