MATLAB Tutorial

written by
Peter Blossey and James A. Rossmanith
Department of Applied Mathematics
University of Washington
Seattle, WA

MATLAB (short for MATrix LABoratory) is a very useful piece of software for numerical analysis. It provides an environment for computation and visualization. Learning MATLAB is not the goal of this course, but a working knowledge of MATLAB will allow you to implement and test the algorithms that form the heart of this course. Seeing these algorithms at work will hopefully enable a deeper understanding of the mechanics, strengths and pitfalls of each algorithm.

Starting MATLAB

To start MATLAB, click on the MATLAB icon or type `matlab` at the command line prompt. To exit, type `quit`.

Entering Variables

There are three types of variables that you can use in matlab: scalars, vectors and matrices. To assign a value to a scalar variable, type

```
>> x = 0.2
```

After hitting return, MATLAB will echo the value of the variable back to you:

```
x =

    0.2000
```

If you don't want to see the value of the variable, add a semicolon at the end of the line:

```
>> x = 0.2;
```

To enter a vector or matrix, use square brackets to indicate the start and end of the vector/matrix. For example a row vector may be entered:

```
>> y = [ 0 1 2 3 4]

y =

     0     1     2     3     4
```

To enter a variable with more than one row, the rows of the vector or matrix are separated by semicolons or by carriage returns. For example a column vector may be entered:

```
>> z = [ 0; 1; 2; 3; 4]

z =

     0
     1
     2
     3
     4
```

A matrix may be entered similarly, with the rows of the matrix separated in this case by carriage returns and the whole expression enclosed in square brackets:

```
>> A = [ 0 2
         3 7
        12 8]

A =

     0     2
     3     7
    12     8
```

The (complex conjugate) transpose of a vector or matrix may be obtained by placing an apostrophe after the expression:

```
>> w = z′

w =

     0     1     2     3     4

>> B = A′

B =

     0     3    12
     2     7     8
```

To give a variable a set of evenly-spaced values, use the colon operator:

```
>> t = 0:6

t =

     0     1     2     3     4     5     6

>> u = 0:0.3:1.5

u =

   0    0.3000    0.6000    0.9000    1.2000    1.5000
```

The first and last numbers are the starting and ending points for the series. The middle number is the spacing between the members of the series. If no spacing is given, MATLAB assumes a spacing of 1.

Accessing elements of a vector/matrix

The individual elements of a vector or matrix may be accessed and/or changed individually by specifying the row and/or column number of the element. In a row or column vector, only a single number is required. In a matrix, both the row and column number must be specified with `A(i,j)` choosing the ith row and jth column.

```
>> y(3)

ans =
     2

>> A(3,2)

ans =
     8
```

However, you must specify a position within the matrix or vector. Otherwise, MATLAB will complain:

```
>> A(2,3)
???  Index exceeds matrix dimensions.
```

You may change individual elements of a matrix or vector in this way:

```
>> y(3) = y(3) + 2

y =
     0     1     4     3     4
```

```
>> A(3,2) = A(3,2) - 3

A =
     0     2
     3     7
    12     5
```

You may also select parts of a matrix or vector, using `B(2,1:2)` to specify the first two elements of the second row of B or `B(2,:)` for the entire second row of B.

```
>> B(2,:)

ans =

     2     7     8
```

Often you need to access elements at or near the end of a vector. You can do this by typing `y(end)` or `y(2:end)` or `y(2:end-1)`. This is very handy, especially when you want to plot part of a vector.

Getting information about variables

To see the size of a variable or its length (number of rows), type `size(A)` or `length(A)`. To see all of the variables that are currently in the MATLAB workspace, type `whos`:

```
>> whos
  Name      Size         Bytes  Class

  A         3x2             48  double array
  B         2x3             48  double array
  ans       1x3             24  double array
  t         1x7             56  double array
  u         1x7             56  double array
  w         1x5             40  double array
  x         1x1              8  double array
  y         1x5             40  double array
  z         5x1             40  double array

Grand total is 45 elements using 360 bytes
```

Elementary computations

Variables may be added, subtracted, multiplied and divided as long as the rules of arithmetic and linear algebra are obeyed, i.e. you can't divide by zero or multiply a 1x2 matrix by a 3x4 matrix. Multiplying a vector or matrix by a scalar will scale each element of the vector or matrix by the value of the scalar.

```
>> C = 2*[1 2; 3 4]

C =

     2     4
     6     8

>> v = 2*[1 2 3 4]

v =

     2     4     6     8
```

Adding a scalar to a vector or matrix will add the value of the scalar to **each** element of the matrix:

```
>> D = 2+[1 2; 3 4]

D =

     3     4
     5     6

>> s = 2 + [0:5]

s =

     2     3     4     5     6     7
```

Matrices and vectors may be added or subtracted as long as they are the same size. They may be multiplied as long as there are the same number of columns in the first as there are rows in the second.

```
>> s + 2*[0:5]

ans =

     2     5     8    11    14    17

>> A*B

ans =
     4    14    16
    14    58    92
    10    71   184

>> B*A

ans =

   153    81
   117    93
```

Putting a period in front of the multiplication, division or power operator performs that operation component-wise, i.e. $(\texttt{x.*y})_i = x_i \cdot y_i$ or $(\texttt{x.}\wedge\texttt{2})_i = x_i^2$.

```
>> [1 2 3 4].^2

ans =

     1     4     9    16

>> [1 2 3 4].*[5 0 5 0]

ans =

     5     0    15     0
```

For loops

To run a command more than once as an index varies, you may use a for loop. In the following example, a for loop is used to compute $n!$ for $n = 1, \ldots, 10$:

```
>> j = 1;
>> for i = 1:10
       j = j*i
end
```

You may also do the loop with the values of `i` in the matrix $[0, 2, 3, 6]$:

```
>> for i = [ 0 2 3 6]
       DO SOMETHING
end
```

Note that MATLAB will not run the for loop until you have hit return after typing `end` to indicate the end of the for loop. I find it useful to use the tab key to indent all commands within the loop. This can make the code much easier to read and understand.

If statements

MATLAB uses a similar structure for if statements:

```
>> if i == 1
      DO SOMETHING
elseif i == 2
      DO SOMETHING ELSE
elseif i == 3
      DO SOMETHING ELSE
else
      DO SOMETHING (IF i >= 4 or i <= 0)
end
```

Plotting

To make a line plot of t versus $\sqrt{t}$, simply type:

```
>> plot(t,sqrt(t))
```

To make a plot to t versus $\sqrt{t}$ and t versus t on the same plot, type

```
>> plot(t,sqrt(t),t,t)
```

putting each x-y pair together. You can add symbols or use symbols instead of lines by adding commands to each pair:

```
>> plot(t,sqrt(t),'*-',t,t,'o-')
```

See `help plot` for more information on plotting and a catalog of the available symbols and line types. Use `legend('sqrt(t)','t')` to label the different lines. Title your plot by typing `title('A plot of t versus sqrt(t)')`. You can add labels to the axes similarly: `xlabel('t')` or `ylabel('sqrt(t)')`. Other useful commands: axis, plot3 and (for a bit of fun) comet.

Clearing variables

You may clear a particular variable by typing

```
>> clear x
```

or all variables with

```
>> clear all
```

Formatting MATLAB output

By default, MATLAB outputs numbers with four digits after the decimal point. If one of the numbers is very large or all of them are very small, MATLAB uses scientific notation. However, the exponent is written only once at the beginning of the output, so be careful. For example:

```
>> [1 2 6 24 120  factorial(20)]

ans =

   1.0e+18 *

    0.0000    0.0000    0.0000    0.0000    0.0000    2.4329
```

To control the format of the output, type

```
>> format long e
```

for scientific notation with 15 digits. Other options include `format short e`, `format short`, `format long`, `format short g`. The default is `format short`.

Importing data

Use the `load` command to bring data into matlab from an external file:

```
>> load filename
```

Getting help

Type `help` followed by the name of the function. For example:

```
>> help plot
```

To get more help, try `helpwin, helpdesk, demo` or `tour`. Also, there is help available on the Math Works website at

`http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml`.

Working with matrices in MATLAB

MATLAB has many useful commands for creating, manipulating or operating on matrices. I hope to show you just a small sample here. To see a more full list, type `help elmat` or `help matfun`.

You can create a 3×4 matrix full of zeros:

```
>> A = zeros(3,4)
A =
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

or a matrix full of ones:

```
>> B = ones(3,4)
B =
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

Note that you can make a row vector full of zeros by typing `zeros(1,5)` or a column vector of zeros using `zeros(5,1)`. (Typing `zeros(5)` will give a 5×5 matrix full of zeros.) Perhaps more useful than a matrix full of ones is the identity matrix, which has ones on the main diagonal and zeros elsewhere. Remember that identity matrices are always square, so that you only need to specify the number of rows in the matrix:

```
>>I = eye(4)
I =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

Identifying rows and columns of matrices

The 3^{rd} row of a matrix can be seen in MATLAB by typing:

```
>> I(3,:)
ans =
     0     0     1     0
```

The third column can be seen in a similar fashion:

```
>> I(:,3)
```

```
ans =
     0
     0
     1
     0
```

You can even select sub-matrices in this way. Here we pick off a 3x3 matrix from the upper right corner of the identity matrix

```
>> I(1:3,2:4)
ans =
     0     0     0
     1     0     0
     0     1     0
```

Diagonals and triangles

Before introducing these, let's create a 4x4 random matrix:

```
>> A = rand(4)
A =
    0.9501    0.8913    0.8214    0.9218
    0.2311    0.7621    0.4447    0.7382
    0.6068    0.4565    0.6154    0.1763
    0.4860    0.0185    0.7919    0.4057
```

You can select the upper triangular part of A using `triu`:

```
>> upper = triu(A)
upper =
    0.9501    0.8913    0.8214    0.9218
         0    0.7621    0.4447    0.7382
         0         0    0.6154    0.1763
         0         0         0    0.4057
```

The lower triangular part may be picked off using `tril`:

```
>> lower = tril(A)
lower =
    0.9501         0         0         0
    0.2311    0.7621         0         0
    0.6068    0.4565    0.6154         0
    0.4860    0.0185    0.7919    0.4057
```

The main diagonal may be extracted (as a vector) using `diag`:

```
>> d = diag(A)
d =
    0.9501
    0.7621
    0.6154
    0.4057
```

One can build matrices from diagonals. For example, useful matrices may be constructed from their diagonals:

```
>> D = diag([1 1 1 1],-1)
D =
     0     0     0     0     0
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0

>> D = D + diag([1 1 1 1],1)
D =
     0     1     0     0     0
     1     0     1     0     0
     0     1     0     1     0
     0     0     1     0     1
     0     0     0     1     0

>> D = D - 2*eye(5)
D =
    -2     1     0     0     0
     1    -2     1     0     0
     0     1    -2     1     0
     0     0     1    -2     1
     0     0     0     1    -2
```

After constructing the matrix with the main diagonals, you may alter the first and last row if they differ from the rest:

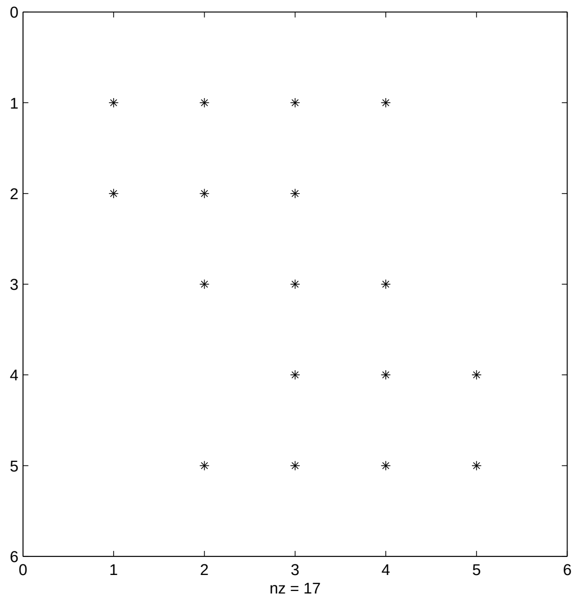
```
>> D(1,1:4) = [2 -5 4 -1]
D =
     2    -5     4    -1     0
     1    -2     1     0     0
     0     1    -2     1     0
     0     0     1    -2     1
     0     0     0     1    -2
```

```
>> D(5,2:5) = [-1 4 -5 2]
D =
     2    -5     4    -1     0
     1    -2     1     0     0
     0     1    -2     1     0
     0     0     1    -2     1
     0    -1     4    -5     2
```

The command `diag(d,n)` takes the vector `d` and places it on the `n`th diagonal of the matrix (relative to the main diagonal. Typing `diag(d)` will put `d` on the main diagonal. Typing `diag(d,1)` will put it one diagonal above the main diagonal. `diag(d,-1)` will put it one diagonal below.

The `Spy` Command

Typing `spy(D)` will bring up the figure window and show the location of all the non-zero elements in `D`. This can help you learn a lot about the structure of sparse (and other types of) matrices. Try it with triangular and other matrices just to see. It works best with large matrices. If your matrix is small, you might try `spy(D,'*')` (as I did to generate the figure below) to put asterisks at each non-zero location so that they are more visible.



M-files in MATLAB

Defn: M-files are MATLAB's version of **MAIN ROUTINES**. All variables in an M-file script will still be there after you run it.

Although all the MATLAB commands can be run on the command line, for developing code that is several lines long it is useful to write your program as an m-file script. I will show you how do to this below.

HOW TO MAKE AN M-FILE:

1. Open up any text editor that you want: **vi**, **emacs**, **pico**, the **MATLAB editor**, etc....
2. Write all the commands in this m-file that you want to execute. I suggest always beginning with the statement `clear all`.
3. Comments can be added as follows:

```
% This lets me add comments.
% Please comment your code so that anyone
% can easily see what you are doing

% Here is an example of a loop:
j = 0;
for i = 1:100
  j = j + i;
end
```

4. Save this file as `somefilename.m`.
5. In MATLAB, use the `cd` command to change directories so that you are in same directory as this file.
6. To run this script in MATLAB type:

```
>> somefilename
```

NOTE: DO NOT USE THE NAME OF AN INTRINSIC MATLAB FUNCTION AS THE TITLE OF YOUR SCRIPT.

BIG NOTE: If your homework requires you to write code, write it an m-file or function and include it in your homework submission.

Functions in MATLAB

Defn: Functions are MATLAB's version of **SUBROUTINES**. All variables in a function are deleted after the function returns to them main routine.

HOW TO MAKE AN FUNCTION:

1. Open up any text editor that you want: **vi**, **emacs**, **pico**, the **MATLAB editor**, etc....
2. The function declaration should be of the form:

   ```
   % This is a sample function declaration
   function [out1,out2,out3] = somefunction(in1,in2)
   ```

3. Comments can be added as follows:

   ```
   % This lets me add comments.
   % Please comment your code so that anyone
   % can easily see what you are doing

   % Here is an example of a loop:
   j = 0;
   for i = 1:100
     j = j + i;
   end
   ```

4. Save this file as `somefunction.m`.
5. In MATLAB, use the `cd` command to change directories so that you are in same directory as this file.
6. To run this function in MATLAB type:

   ```
   >> [y1,y2,y3]=somefunction(x1,x2);
   ```

   NOTE: DO NOT USE THE NAME OF AN INTRINSIC MATLAB FUNCTION AS THE TITLE OF YOUR FUNCTION.

BIG NOTE: If your homework requires you to write code, write it an m-file or function and include it in your homework submission.

Passing functions to functions

We can pass one function `fun1.m` to another function fun2.m in MATLAB by using the `@` symbol:

```
>> fun2(@fun1)
```

Example:

The built-in function `quad(f,a,b)` in MATLAB approximately integrates the function $y = f(x)$ from $x = a$ to $x = b$. How can we use this built-in function to approximately integrate $y = \cos(\pi x) + 2$ from $x = -1$ to $x = 1$?

1. We begin by writing a MATLAB function called `fun.m`:

```
% fun.m - Here is a simple function

function y = fun(x)

   y = cos(pi*x) + 2;
```

2. If we are now in the correct directory, we type in MATLAB:

```
>> quad(@fun,-1,1)

ans =

     4.000000016674772e+00
```

The `@` symbol thus allows us to send any input function to `quad(f,a,b)` to be integrated.

Evaluating input functions inside of functions

In order to evaluate functions which are passed into a function we must utilize the `feval` function from MATLAB.

Example:

We would like to write a function that computes the slope of the line between the two points $(a, f(a))$ and $(b, f(b))$. Recall that the slope between these two points is give by

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

1. We begin by writing a MATLAB function called `slope.m`:

```
% slope.m - compute the slope between (a,f(a)) and (b,f(b))

function m = slope(func,a,b)

   fa = feval(func,a);
   fb = feval(func,b);

   m = (fb-fa)/(b-a);
```

2. We again use the `fun.m` function from the previous page. If we are now in the correct directory, we type in MATLAB:

```
>> slope(@fun,0.4,0.6)

ans =

   -3.090169943749475e+00
```