Matlab Tutorial 1: Basics

Time

We estimate this tutorial will take 20-30 minutes. That includes time for a bit of playing around with the commands.

MathWorks Help Website

MathWorks maintains a very useful website for MatLab help. Between that and search engines, it's fairly easy to find out how to do what you need.

Getting Started

We'll assume you have Matlab installed and have started it.

Starting the desktop: If you have a window with several sections then the desktop started automatically. If you have a simple commandline window with the Matlab command prompt you should start the desktop with the command
>> desktop [return]

Once the desktop is started you should figure out which of the windows is for entering commands.

Conventions

% This is an edited transcript of a Matlab session.
% We've inserted comment lines, which begin with a '%'.

Command lines begin with a '>>'. When copying a command you should enter everything after the '>>' and hit return. Matlab's response will be shown in the line or lines below the command.

Trick: You can use the up arrow to find previous commands. If you type the first letter of the command then the up arrow will only show previous commands that start with that letter.

Matlab as a Calculator

% The basic operations are * + - / ^. Try the following:

```
>> 2+3
ans =  5
>> 2*3
ans =  6
>> 2/3
ans =  0.66667
>> 2^3
ans =  8
>> 2*(3+1)^2
```

```
ans =  32
```

Using Variables

```
% You can store results in variables and use them in calculations. Try the
following:

>> x = 2+3
x =  5
>> x
x =  5
>> y = 1+2
y =  3
>> x*y
ans =  15
>> z = x^y
z =  125
```

The Variable *ans*

```
%If you don't name an expression Matlab names it ans for you. This is useful if
you forget to name something.

>> 7+15
ans = 22
>> ans
ans = 22
>> x = ans
x = 22

% Of course Matlab changes the meaning of ans the next time you
don't name an expression.
>> 7+15
ans = 22
>> ans
ans = 22
>> 9 + 2
ans = 11
>> x = ans
x = 11
```

Functions

```
% Matlab has all the functions you know and love.
% We'll start with functions on numbers.

>> sin(1)
ans =  0.84147

>> sin(1.4)
ans =  0.98545

>> sin(3)
ans =  0.14112
```

```
% Matlab knows about pi.
>> pi
ans =  3.1416

>> sin(pi/2)
ans =  1

% The exponential function is given by 'exp'.
>> exp(0)
ans =  1

>> exp(1)
ans =  2.7183
```

Arrays

% An array is the same thing as a matrix, i.e. a table of numbers. (Matlab is short for Matrix Laboratory.)
% You create arrays by using square brackets. You put commas between the entries in rows and semicolons between the rows.

```
% 1 x 4 array
>> [1, 2, 3, 4]
ans =
   1   2   3   4

% 2 x 3 array
>> [1, 2, 3; 4, 5, 6]
ans =
   1   2   3
   4   5   6

% You can store arrays in variables.
>> x = [1, 2, 3, 4]
x =
   1   2   3   4

% Turning rows into columns is called transposing. In mathematics
we use a T, e.g. A^T. You read this as 'A transpose'. In Matlab we
use a single quote after the bracket.
>> x = [1, 2, 3, 4]'
x =
   1
   2
   3
   4

>> y = [1, 2, 3; 4, 5, 6]
y =
   1   2  3
   4   5  6

>> y = [1, 2, 3; 4, 5, 6]'
y =
   1   4
   2   5
   3   6
```

```
% If the entries are simple enough you can use spaces instead of
commas between entries in a row. (You still need a semicolon
between rows.)
>> x = [1 2 3; 4 5 6]
x =
   1   2   3
   4   5   6
```

Accessing entries in an array.

```
% Arrays are accessed with the notation A(row,col).
% Note:Indexing in Matlab is 1-based. Not 0-based as in Python.
>> A = [1, 2, 3; 4, 5, 6]
A =
   1   2   3
   4   5   6

>> A(1,1)
ans =  1
>> A(2,1)
ans =  4
>> A(2,3)
ans =  6

% For one dimensional arrays you only need one index.
>> x = [1, 2, 3, 4, 5]
x =
   1  2  3  4  5  6
>> x(1)
ans = 1
>> x(4)
ans = 4
```

Matrix Arithmetic

% You can add or subtract matrices (arrays) of *the same size*.

```
>> x = [1, 2, 3]
x =
   1   2   3

>> y = [4, 5, 6]
y =
   4   5   6

>> x + y
ans =
   5   7   9

>> x = [1 2; 3 4]
x =
   1   2
   3   4

>> x + [6, 5; 1, 2]
ans =
   7   7
```

```
   4   6

% You can scale or divide matrices by a number.
>> x = [1, 2, 3]
x =
   1   2   3

>> 2*x
ans =
   2   4   6

>> x/2
ans =
   0.50000   1.00000   1.50000

% You can multiply compatibly sized matrices.
% 3x2 times 2x4 is 3x4.
>> A = [1 2; 3 4; 5 6]
A =
   1  2
   3  4
   5  6
>> B = [1 2 3 4; 5 6 7 8]
B =
   1  2  3  4
   5  6  7  8
>> A*B
ans =
   11   14   17   20
   23   30   37   44
   35   46   57   68

% If the sizes aren't compatible you get an error message.
>> A*A
Error using *
Inner matrix dimensions must agree.


% You can raise square matrices to powers.
% Try the following:
>> A = [6, 5; 1, 2]
A =
   6   5
   1   2

>> A^2
ans =
   41   40
    8    9

>> A^3
ans =
    286   285
     57    58

% Given how much we love exponentials, it's nice that Matlab can
compute the exponential of a square matrix.
% The name of the function is expm. Be careful here, exp(A) does
elementwise exponentiation (see below)
>> expm(A)
ans =
```

```
   914.31   911.60
   182.32   185.04

% See what happens if the matrix is not square.
>> B = [1, 2, 3; 4, 5, 6]
B =
   1  2  3
   4  5  6
>> B^2
% My version of Matlab produces the following error message:
Error using ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
```

Elementwise Arithmetic

Sometimes you want to raise each element of an array to a power. The notation is a bit odd: it requires a period before the caret.

```
% Notice the '.' before the '^'.
>> x = [1 2 3; 4 5 6]
x =
   1   2   3
   4   5   6

>> x.^2
ans =
    1    4    9
   16   25   36

>> x.^0.5
ans =
    1.0000   1.4142   1.7321
    2.0000   2.2361   2.4495

% The same notation works to multiply each element of an array by
the same element in another array of the same size.
>> [1, 2, 3; 4, 5, 6] .* [2, 4, 6; 8, 10, 12]
ans =
     2     8    18
    32    50    72

% It also lets you raise each element of an array to the same
element in another array.
>> [2, 2, 2, 2, 2] .^ [1, 2, 3, 4, 5]
ans =
     2  4  8  16  32

% Example: find the sum of the squares of the integers from 1 to
1024.
>> x = [1:1024];
>> sum(x.^2)
ans =  358438400

% This can be done in one command.
>> sum([1:1024].^2)
ans =  358438400
```

Some Standard Array Operations

% Here we will learn to find the determinant and inverse of a square array and the reduced row echelon form and rank of any array.

```
% Check each of the following computations.
>> A = [6 5; 1 2]
A =
    6   5
    1   2

% Determinant:
>> det(A)
ans = 7

% Inverse:
>> inv(A)
ans =
    0.2857   -0.7143
   -0.1429    0.8571
% Another way to find inverses:
>> A^(-1)
ans =
    0.2857   -0.7143
   -0.1429    0.8571

% Solving systems. Example: Solve the system
       6x + 5y = 4
        x + 2y = 7.
We solve this using matrix methods:

% The coefficient matrix is our matrix A. Note the prime on the
array in the next command.
>> solution = A^(-1)*[4, 7]'
ans =
   -3.8571
    5.4286

% Solving systems is so important, Matlab has another, better way
of doing this.
>> solution = A\[4 7]'
ans =
   -3.8571
    5.4286

% Reduced row echelon form:
>> rref(A)
ans =
    1    0
    0    1

% That was too simple. Here's another example.
>> B = [1 2 3; 4 5 6; 7 8 9]
B =
   1   2   3
```

```
   4   5   6
   7   8   9
>> rref(B)
ans =
     1     0    -1
     0     1     2
     0     0     0

% Rank of a matrix:
>> rank(A)
ans = 2
>> rank(B)
ans = 2
```

Creating Special Arrays

% [3:1:10] is the array from 3 to 10 counting by 1.

```
>> x = [3:1:10]
x =
    3    4    5    6    7    8    9   10

% Changing the 1 to 2 means count by 2's:
>> x = [3:2:10]
x =
   3   5   7   9

% You can count by any number
>> x = [3:0.1:3.5]
x =
   3.0000   3.1000   3.2000   3.3000   3.4000   3.5000

% For counting by 1's, you can leave out the middle number
>> x = [3:10]
x =
    3    4    5    6    7    8    9   10

% Creating the identity matrix
>> x = eye(3)
x =
     1     0     0
     0     1     0
     0     0     1
```

Suppressing output to the screen

% For large arrays you'll want to suppress output to the screen.
% You do this by putting a semicolon at the end of the line.
% We'll demonstrate on small arrays so you can see the difference between ending a command with or without a semicolon.

```
>> x = [1:2:9]
x =
    1    3    5    7    9

% With semicolon (no output)
```

```
>> x = [1:2:9];
>> y = 2*x;

>> z = 2*x & no semicolon, yes output
z =
    2    6    10   14   18

% If you forget the semicolon you can stop the screen output by
typing a q followed by  return. Try this with the following command
>> [1:1000]
```

Functions on Arrays

% Most functions can be used on arrays.

```
% Sin acts on arrays by taking the sin of each element.
>> x = [1, 2, 3; 4, 5, 6]
x =
   1   2   3
   4   5   6

>> sin(x)
ans =
   0.84147   0.90930   0.14112
  -0.75680  -0.95892  -0.27942

% exp also acts elementwise on arrays. (Compare this with expm on
square matrices.)
>> exp(x)
ans =
     2.7183     7.3891    20.0855
    54.5982   148.4132   403.4288
```

Matlab Sums the Elements of Arrays

We don't need it just yet, but the *sum* function will be quite useful.

```
% For a single row or column sum adds them up.
>> x = [1,2,3];
x =
   1  2  3
>> sum(x)
ans = 6

>> y = x'
y =
    1
    2
    3
>> sum(y)
ans = 6

% Find the sum of the first 1000 integers.
>> sum([1:1000])
ans =  500500

% Find the sum of the squares of the first 1000 integers.
>> sum([1:1000].^2)
```

```
ans = 333833500

% For a two dimensional array sum produces the sum of each of the
columns.
>> x = [1, 2, 3, 4, 5; 7, 9, 11, 13, 15]
x =
    1   2   3   4   5
    7   9  11  13  15
>> sum(x)
ans =
      8  11  14  17  20
% Note the result of summing a 2x5 array is a 1x5 array.
```