MATLAB: Introduction
Part 1

Bruno Abreu Calfa

Last Update: August 9, 2011

Contents

1 MATLAB as a Calculator

The following operators are supported:

- + (Addition)
- - (Subtraction)
- * (Multiplication)
- / (Division)

```
1 + 2
2*3 + 4
4/3 - 3/4 + 2^3

% Beware of operator precedence rules!
% Use parentheses to enforce the desired order

2*3 + 4
2*(3 + 4)
4.2/3 + 1.2
4.2/(3 + 1.2)
15/(2 + 3)*(4 - 1)
15/((2 + 3)*(4 - 1))
2^3/2
2^(3/2)

ans =

     3


ans =

    10


ans =

    8.5833


ans =

    10


ans =

    14


ans =

    2.6000


ans =
```

```
     1

ans =

     9

ans =

     1

ans =

     4

ans =

    2.8284
```

2 MATLAB Classes

"Everything" in MATLAB is a matrix!

```
% Scalars
a = 1 % The scalar variable 'a' stores the value 1
% This is a comment and is ignored by the interpreter
sin(a) % Sine of 'a' = 0.8415
sin(a); % ';' avoids displaying the result of the command
size(a) % = [1,1], i.e. 1-by-1 matrix
b = a + 2 % b = 3
c = cos(b*pi/.2) % 'pi' is the predefined constant
d = rand % A random scalar

% List variables in workspace
w1 = who
w2 = whos
```

```
a =

     1
```

```
ans =

    0.8415


ans =

     1     1


b =

     3


c =

    -1


d =

    0.7655


w1 =

    'a'
    'ans'
    'b'
    'c'
    'd'
    'opts'


w2 =

7x1 struct array with fields:
    name
    size
    bytes
    class
    global
    sparse
    complex
    nesting
```

```
    persistent
```

3 1-D Arrays (Vectors)

Use [...,...] or [... ...] for horizontal stacking and [...;...] for vertical stacking.

```
v1 = [1 2 3] % Row vector, same as v1 = [1,2,3]
v2 = [4;5;6] % Column vector
v3 = v2 - v1.' % Transpose a real matrix with .'
v4 = v1*v2 % Dot product, also dot(v1,v2)
v7 = .1*v4 % Scalar-vector multiplication
v7(1) % First element of array 'v7'
v8 = exp(v7) % Element-wise operation
sz8 = size(v8) % = [1 3]
v9 = rand(1,5) % Random 1-by-5 array
p = prod(v1) % Product of elements = 6
```

```
v1 =

     1     2     3


v2 =

     4
     5
     6


v3 =

     3
     3
     3


v4 =

    32


v7 =

    3.2000
```

```
ans =

    3.2000


v8 =

   24.5325


sz8 =

     1     1


v9 =

    0.7952    0.1869    0.4898    0.4456    0.6463


p =

     6
```

4 2-D Arrays (Matrices)

Use horizontal stacking and vertical stacking likewise

```
m1 = [1 2 3; 4 5 6] % 2-by-3
m1p = [1,2,3; 4,5,6] % 2-by-3, same as m1
m2 = rand(2,3) % Random 2-by-3 matrix
m3 = m1 + m2 % Matrix addition
m4 = m1*m2.' % OK! Transpose a real matrix with .'
m4(1,2) % Element in row 1 and column 2 of 'm4'
len4 = length(m4) % Size of longest dimension
m5 = m3/2 % Element-wise division
m6 = tan(m5) % Element-wise operation
```

```
m1 =

     1     2     3
     4     5     6


m1p =
```

```
     1     2     3
     4     5     6


m2 =

    0.7094    0.2760    0.6551
    0.7547    0.6797    0.1626


m3 =

    1.7094    2.2760    3.6551
    4.7547    5.6797    6.1626


m4 =

    3.2267    2.6019
    8.1482    7.3929


ans =

    2.6019


len4 =

     2


m5 =

    0.8547    1.1380    1.8275
    2.3773    2.8399    3.0813


m6 =

    1.1491    2.1645   -3.8088
   -0.9586   -0.3112   -0.0604
```

5 Element-wise Operations

The following are element-wise mathematical operators:

- .* (Element-wise Multiplication)
- ./ (Element-wise Division)
- .^ (Element-wise Exponentiation)

```
v1 = [1 2 3] % 1-by-3
v2 = [2 4 6] % 1-by-3
v3 = v1.*v2 % = [2 8 18]
v4 = v2./v1 % = [2 2 2]
v5 = v1.^v4 % = [1 4 9]
m1 = [0 1; 1 0] % 2-by-2
m2 = [3 5; 7 2] % 2-by-2
m3 = m1.*m2 % = [0 5; 7 0]
```

```
v1 =

     1     2     3


v2 =

     2     4     6


v3 =

     2     8    18


v4 =

     2     2     2


v5 =

     1     4     9


m1 =

     0     1
     1     0
```

```
m2 =

     3     5
     7     2


m3 =

     0     5
     7     0
```

6 The Colon (:) Operator

Use it extensively!

```
v1 = 1:10 % Same as v1 = [1,2,3,...,10]
v2 = 0:.1:1 % Same as v2 = [0,.1,.2,...,1]
m1 = rand(5) % Random 5-by-5 matrix
v3 = v1(5:end) % v3 = [5,6,7,8,9,10]
v4 = m1(:,3) % 'v4' has the elements in column 3 of 'm1'
v5 = m1(1,:) % 'v5' has the elements in row 1 of 'm1'

% Do not forget linspace to generate linearly spaced vectors!
v6 = linspace(0,1,10) % = [0, 0.1111, 0.2222, ..., 1]
v7 = linspace(0,10,5) % = [0, 2.5, 5, 7.5, 10]
v8 = linspace(0,1,100) % = [0, 0.0101, 0.0202, ..., 1]

v1 =

     1     2     3     4     5     6     7     8     9    10


v2 =

  Columns 1 through 9

         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
 0.7000    0.8000

  Columns 10 through 11

    0.9000    1.0000
```

```
m1 =

    0.1190    0.2238    0.8909    0.2575    0.9293
    0.4984    0.7513    0.9593    0.8407    0.3500
    0.9597    0.2551    0.5472    0.2543    0.1966
    0.3404    0.5060    0.1386    0.8143    0.2511
    0.5853    0.6991    0.1493    0.2435    0.6160


v3 =

     5     6     7     8     9    10


v4 =

    0.8909
    0.9593
    0.5472
    0.1386
    0.1493


v5 =

    0.1190    0.2238    0.8909    0.2575    0.9293


v6 =

  Columns 1 through 9

         0    0.1111    0.2222    0.3333    0.4444    0.5556    0.6667
 0.7778    0.8889

  Column 10

    1.0000


v7 =

         0    2.5000    5.0000    7.5000   10.0000


v8 =
```

```
 Columns 1 through 9

         0    0.0101    0.0202    0.0303    0.0404    0.0505    0.0606
0.0707    0.0808

 Columns 10 through 18

   0.0909    0.1010    0.1111    0.1212    0.1313    0.1414    0.1515
0.1616    0.1717

 Columns 19 through 27

   0.1818    0.1919    0.2020    0.2121    0.2222    0.2323    0.2424
0.2525    0.2626

 Columns 28 through 36

   0.2727    0.2828    0.2929    0.3030    0.3131    0.3232    0.3333
0.3434    0.3535

 Columns 37 through 45

   0.3636    0.3737    0.3838    0.3939    0.4040    0.4141    0.4242
0.4343    0.4444

 Columns 46 through 54

   0.4545    0.4646    0.4747    0.4848    0.4949    0.5051    0.5152
0.5253    0.5354

 Columns 55 through 63

   0.5455    0.5556    0.5657    0.5758    0.5859    0.5960    0.6061
0.6162    0.6263

 Columns 64 through 72

   0.6364    0.6465    0.6566    0.6667    0.6768    0.6869    0.6970
0.7071    0.7172

 Columns 73 through 81

   0.7273    0.7374    0.7475    0.7576    0.7677    0.7778    0.7879
0.7980    0.8081

 Columns 82 through 90
```

```
    0.8182    0.8283    0.8384    0.8485    0.8586    0.8687    0.8788
 0.8889    0.8990

  Columns 91 through 99

    0.9091    0.9192    0.9293    0.9394    0.9495    0.9596    0.9697
 0.9798    0.9899

  Column 100

    1.0000
```

7 Strings: char Arrays

Remember that strings are also matrices in MATLAB!

```
str1 = 'Hello, world!' % A simple string
sz1 = size(str1) % = 1-by-13
a = rand; str2 = ['a = ' num2str(a)] % Horizontal stacking
   concatenates strings
b = str2num('500')*rand % MATLAB has many handy *2* functions!

% Format your strings with sprintf
sprintf('Volume of reactor = %.2f', 10.23451) % Floating-point
   format with two decimal digits
str3 = sprintf('A large number = %e', rand*10^5) % Exponential
   notation format
sprintf('Another large number = %g', rand*10^5) % More compact
   format between %e and %f
```

```
str1 =

Hello, world!


sz1 =

     1    13


str2 =

a = 0.47329
```

```
b =

  175.8298


ans =

Volume of reactor = 10.23


str3 =

A large number = 8.308286e+04


ans =

Another large number = 58526.4
```

8 function_handle (@) Class

Used in calling functions indirectly and in creating 'anonymous' functions.

```
Sin = @sin; % The variable 'Sin' points to the function 'sin'
Sin(pi) % Evaluates the sine of

% Anonymous function example
myfun = @(x) 1./(x.^3 + 3*x - 5) % Anonymous function
quad(myfun,0,1) % Adaptive Simpson quadrature to integrate 'myfun
   '


ans =

   1.2246e-16


myfun =

    @(x)1./(x.^3+3*x-5)


ans =

   -0.3644
```

9 Scripts and Functions: M-Files

This file is actually a 'script' M-File (no explicit 'function' definitions). Script M-files are useful to set up the workspace, i.e. define variables, and then call function M-files to execute actions.

```
% Call the M-file 'matlab_intro_part_1_function.m'
matlab_intro_part_1_function
```

```
Methane at 1500000.00 (m/s^2*kg)^-1 and 400.00 K has Z = 0.9931
```

```
function matlab_intro_part_1_function
%MATLAB_INTRO_PART_1_MFILES This function M-file covers the usage
    of MATLAB
%M-files and how to create user-defined functions.
%   In this example, let us calculate the compressibility factor,
    Z, of a
%   pure gas by using the generalized virial equation.
```

The equation is given by:

$$Z = 1 + \frac{P_r}{T_r}(B^0 + \omega B^1)$$

where:

$$B^0 = 0.083 - \frac{0.422}{T_r^{1.6}}$$

$$B^1 = 0.139 - \frac{0.172}{T_r^{4.2}}$$

```
    % unit class
    u = cmu.units;

    % Data for methane
    P = 15*u.bar; % pressure
    Pc = 45.99*u.bar; % Critical pressure
    T = 400*u.K; % Temperature
    Tc = 190.6*u.K; % Critical temperature
    omega = 0.012; % Acentric factor

    Z = virialgen(P,Pc,T,Tc,omega);
    str = sprintf('Methane at %.2f and %.2f has Z = %.4f', P, T,
       Z);
    disp(str);
end

function Z = virialgen(P,Pc,T,Tc,omega)
    Pr = P/Pc;
    Tr = T/Tc;
```

```
    [B0,B1] = virialB(Tr);
    Z = 1 + Pr/Tr*(B0 + omega*B1);
end

function [B0,B1] = virialB(Tr)
    B0 = 0.083 - 0.422/Tr^1.6;
    B1 = 0.139 - 0.172/Tr^4.2;
end
```

```
Methane at 1500000.00 (m/s^2*kg)^-1 and 400.00 K has Z = 0.9931
```