matlab octave : tutorial

basic operations

```
%% Change Octave prompt
PS1('>> ');
%% Change working directory in windows example:
cd 'c:/path/to/desired/directory name'
%% Note that it uses normal slashes and does not use escape
characters for the empty spaces.

%% elementary operations
5+6
3-2
5*8
1/2
2^6
1 == 2  % false
1 ~= 2  % true.  note, not "!="
1 && 0
1 || 0
xor(1,0)


%% variable assignment
a = 3; % semicolon suppresses output
b = 'hi';
c = 3>=1;

% Displaying them:
a = pi
disp(a)
disp(sprintf('2 decimals: %0.2f', a))
disp(sprintf('6 decimals: %0.6f', a))
format long
a
format short
a


%%  vectors and matrices
A = [1 2; 3 4; 5 6]

v = [1 2 3]
```

```
v = [1; 2; 3]
v = [1:0.1:2]  % from 1 to 2, with stepsize of 0.1. Useful for
plot axes
v = 1:6        % from 1 to 6, assumes stepsize of 1 (row vector)

C = 2*ones(2,3)  % same as C = [2 2 2; 2 2 2]
w = ones(1,3)    % 1x3 vector of ones
w = zeros(1,3)
w = rand(1,3)  % drawn from a uniform distribution
w = randn(1,3) % drawn from a normal distribution (mean=0,
var=1)
w = -6 + sqrt(10)*(randn(1,10000));  % (mean = -6, var = 10) -
note: add the semicolon
hist(w)     % plot histogram using 10 bins (default)
hist(w,50)  % plot histogram using 50 bins
% note: if hist() crashes, try "graphics_toolkit('gnu_plot')"

I = eye(4)    % 4x4 identity matrix

% help function
help eye
help rand
help help
```

moving data around

```
%% dimensions
sz = size(A) % 1x2 matrix: [(number of rows) (number of
columns)]
size(A,1)  % number of rows
size(A,2)  % number of cols
length(v)  % size of longest dimension


%% loading data
pwd    % show current directory (current path)
cd 'C:\Users\ang\Octave files'   % change directory
ls     % list files in current directory
load q1y.dat    % alternatively, load('q1y.dat')
load q1x.dat
who    % list variables in workspace
whos   % list variables in workspace (detailed view)
clear q1y       % clear command without any args clears all vars
v = q1x(1:10);  % first 10 elements of q1x (counts down the
columns)
save hello.mat v;   % save variable v into file hello.mat
save hello.txt v -ascii; % save as ascii
% fopen, fread, fprintf, fscanf also work  [[not needed in
class]]

%% indexing
A(3,2)  % indexing is (row,col)
A(2,:)  % get the 2nd row.
        % ":" means every element along that dimension
A(:,2)  % get the 2nd col
A([1 3],:) % print all  the elements of rows 1 and 3

A(:,2) = [10; 11; 12]     % change second column
A = [A, [100; 101; 102]]; % append column vec
A(:) % Select all elements as a column vector.

% Putting data together
A = [1 2; 3 4; 5 6]
B = [11 12; 13 14; 15 16] % same dims as A
C = [A B]  % concatenating A and B matrices side by side
C = [A, B] % concatenating A and B matrices side by side
C = [A; B] % Concatenating A and B top and bottom
```

computing on data

```
%% initialize variables
A = [1 2;3 4;5 6]
B = [11 12;13 14;15 16]
C = [1 1;2 2]
v = [1;2;3]
%% matrix operations
A * C  % matrix multiplication
A .* B % element-wise multiplication
% A .* C  or A * B gives error - wrong dimensions
A .^ 2 % element-wise square of each element in A
1./v   % element-wise reciprocal
log(v)  % functions like this operate element-wise on vecs or
matrices
exp(v)
abs(v)

-v  % -1*v

v + ones(length(v), 1)
% v + 1  % same

A'  % matrix transpose

%% misc useful functions
% max  (or min)
a = [1 15 2 0.5]
val = max(a)
[val,ind] = max(a) % val -  maximum element of the vector a and
index - index value where maximum occur
val = max(A) % if A is matrix, returns max from each column

% compare values in a matrix & find
a < 3 % checks which values in a are less than 3
find(a < 3) % gives location of elements less than 3
A = magic(3) % generates a magic matrix - not much used in ML
algorithms
[r,c] = find(A>=7)  % row, column indices for values matching
comparison
% sum, prod
sum(a)
prod(a)
floor(a) % or ceil(a)
```

```
max(rand(3),rand(3))
max(A,[],1) -  maximum along columns(defaults to columns -
max(A,[]))
max(A,[],2) - maximum along rows
A = magic(9)
sum(A,1)
sum(A,2)
sum(sum( A .* eye(9) ))
sum(sum( A .* flipud(eye(9)) ))
% Matrix inverse (pseudo-inverse)
pinv(A)        % inv(A'*A)*A'
```

plotting data

```
%% plotting
t = [0:0.01:0.98];
y1 = sin(2*pi*4*t);
plot(t,y1);
y2 = cos(2*pi*4*t);
hold on;  % "hold off" to turn off
plot(t,y2,'r');
xlabel('time');
ylabel('value');
legend('sin','cos');
title('my plot');
print -dpng 'myPlot.png'
close;           % or,  "close all" to close all figs
figure(1); plot(t, y1);
figure(2); plot(t, y2);
figure(2), clf;  % can specify the figure number
subplot(1,2,1);  % Divide plot into 1x2 grid, access 1st element
plot(t,y1);
subplot(1,2,2);  % Divide plot into 1x2 grid, access 2nd element
plot(t,y2);
axis([0.5 1 -1 1]);  % change axis scale
%% display a matrix (or image)
figure;
imagesc(magic(15)), colorbar, colormap gray;
% comma-chaining function calls.
a=1,b=2,c=3
a=1;b=2;c=3;
```

control statements: for, while, if statement

```
v = zeros(10,1);
for i=1:10,
    v(i) = 2^i;
end;

% Can also use "break" and "continue" inside for and while loops
to control execution.

i = 1;
while i <= 5,
  v(i) = 100;
  i = i+1;
end

i = 1;
while true,
  v(i) = 999;
  i = i+1;
  if i == 6,
    break;
  end;
end

if v(1)==1,
  disp('The value is one!');
elseif v(1)==2,
  disp('The value is two!');
else
  disp('The value is not one or two!');
end
```

functions

to create a function, type the function code in a text editor (e.g. gedit or notepad), and save the file as "functionname.m"

example function:

```
function y = squareThisNumber(x)

y = x^2;
```

to call the function in octave, either:

1) navigate to the directory of the functionname.m file and call the function:

```
    % Navigate to directory:
    cd /path/to/function

    % Call the function:
    functionName(args)
```

2) add the directory of the function to the load path and save it:
should not use addpath/savepath for any of the assignments in this course. instead use 'cd' to change the current working directory.

```
    % To add the path for the current session of Octave:
    addpath('/path/to/function/')

    % To remember the path for future sessions of Octave, after
executing addpath above, also do:
    savepath
```

octave's functions can return more than one value:

```
    function [y1, y2] = squareandCubeThisNo(x)
    y1 = x^2
    y2 = x^3
```

call the above function this way:

```
   [a,b] = squareandCubeThisNo(x)
```

vectorization

vectorization is the process of taking code that relies on **loops** and converting it into **matrix operations**. it is more efficient, more elegant, and more concise.

as an example, compute the prediction from a hypothesis. theta is the vector of fields for the hypothesis and x is a vector of variables.

with loops:

```
prediction = 0.0;
for j = 1:n+1,
  prediction += theta(j) * x(j);
end;
```

with vectorization:

```
prediction = theta' * x;
```

recalling the definition multiplying vectors, this one operation does the element-wise multiplication and overall sum in a concise notation.