ML:Octave Tutorial

From Coursera

Contents

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

Data files used in this section: featuresX.dat (https://raw.githubusercontent.com/tansaku/py-coursera/master/featuresX.dat) , priceY.dat (https://raw.githubusercontent.com/tansaku/py-coursera/master/priceY.dat)

```
%% 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 w/ no argt clears all
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] or [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

% find
a < 3
find(a < 3)
A = magic(3)
[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` statements

```
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, do 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:

```
   % 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, let's compute our 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;
```

If you recall the definition multiplying vectors, you'll see that this one operation does the element-wise multiplication and overall sum in a very concise notation.

Working on and Submitting Programming Exercises

1. Download and extract the assignment's zip file.
2. Edit the proper file 'a.m', where a is the name of the exercise you're working on.
3. Run octave and cd to the assignment's extracted directory
4. Run the 'submit' function and enter the assignment number, your email, and a password (found on the top of the "Programming Exercises" page on coursera)

Video Lecture Table of Contents

Basic Operations

```
0:00    Introduction
3:15    Elementary and Logical operations
5:12    Variables
7:38    Matrices
8:30    Vectors
11:53   Histograms
12:44   Identity matrices
13:14   Help command
```

Moving Data Around

```
0:24    The size command
1:39    The length command
2:18    File system commands
2:25    File handling
4:50    Who, whos, and clear
6:50    Saving data
8:35    Manipulating data
12:10   Unrolling a matrix
12:35   Examples
14:50   Summary
```

Computing on Data

```
0:00    Matrix operations
0:57    Element-wise operations
4:28    Min and max
```

```
5:10    Element-wise comparisons
5:43    The find command
6:00    Various commands and operations
```

Plotting data

```
0:00    Introduction
0:54    Basic plotting
2:04    Superimposing plots and colors
3:15    Saving a plot to an image
4:19    Clearing a plot and multiple figures
4:59    Subplots
6:15    The axis command
6:39    Color square plots
8:35    Wrapping up
```

Control statements

```
0:10    For loops
1:33    While loops
3:35    If statements
4:54    Functions
6:15    Search paths
7:40    Multiple return values
8:59    Cost function example (machine learning)
12:24   Summary
```

Vectorization

```
0:00    Why vectorize?
1:30    Example
4:22    C++ example
5:40    Vectorization applied to gradient descent
9:45    Python
```

Next: Logistic Regression Back to Index: Main

External Resources

Octave Quick Reference (http://enacit1.epfl.ch/octave_doc/refcard/refcard-a4.pdf)

An Introduction to Matlab (http://www.maths.dundee.ac.uk/ftp/na-reports/MatlabNotes.pdf)

Learn X in Y Minutes: Matlab (https://learnxinyminutes.com/docs/matlab/)