MATLAB ® Tutorial

Anand Sampat, asampat@stanford.edu

October 10, 2014

Introduction

MATLAB ® is a high-level language and interactive environment for numerical computation, visualization, and programming. In CS 229 we'll use it to prototype machine learning algorithms using standard MATLAB ® functions (no special libraries). Since we don't require external libraries, you can also use Octave, which is an open source version of MATLAB ® .

Accessing MATLAB® and/or Octave

As a Stanford student, you have 2 options to use MATLAB ®

1) Download the program (for more information see 'https://itservices.stanford.edu/service/softwarelic/matlab'. You can buy MATLAB® 2013 or 2014.

2) Use MATLAB® on the corn machines on Farmshare. In order to use it you can use the command: `ssh -CY username@corn.stanford.edu` in the UNIX command prompt (-C compresses your data over the network which may increase the speediness when using MATLAB® and -Y provides a more secure version of X11 forwarding). Try `matlab &` to open up MATLAB®. If it is not already installed please visit `https://web.stanford.edu/group/farmshare/cgi-bin/wiki/index.php/MATLAB` for more information on how to load MATLAB® modules on Farmshare.

If you prefer to use Octave, you can download it for multiple platforms @ `https://www.gnu.org/software/octave/download.html`.

Basics MATLAB® is a scripting language and thus has an interactive environment and a proper functional backbone. Scripts can be written into ".m" files either as a list of interactive commands, or as a single function. Technically it is also possible to have multiple functions in one file by defining classes using OOP, but for this class we won't need such functionality. (See `logistic_grad_ascent.m` for a good example of this).

Below I'll go through certain classes of basic functions and finally will give a few examples of learning algorithms and functions associated with those matrix manipulations. All of the commands can be found in `matlab_session_2014.m`.

Elementary Functions Similar to Python and other interactive languages, MATLAB® can be used to do simple math and if no semicolon is used to end the line, the result will be displayed and stored in the "ans" variable.

```
%%% 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 Assignment is done via a single equals sign. You can also explicitly display variables in specific formats (float, int, etc.) using sprintf and disp functions.

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

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

Vectors and Matrices Vectors and matrices can be defined using square brack-

ets. Values separated by a space or comma are within the same row. Values separated by a semicolon are within the same column.

```
%%  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

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 = 1, var = 2)
hist(w)
e = []; % empty vector
I = eye(4)    % 4x4 identity matrix
```

Dimensions

```
%% dimensions
sz = size(A)
size(A,1)  % number of rows
size(A,2) % number of cols
length(v)  % size of longest dimension
```

Indexing Indices can be just numbers, sequences of numbers, or matrices themselves. To find out about all of the possibilities please refer to `http://www.mathworks.com/help/matlab/math/matrix-indexing.html`.

```
%% indexing
A(3,2) % indexing is (row,col)
A(2,:)  % get the 2nd row. %% ":" means every elt along that ...
    dimension
A(:,2)  % get the 2nd col
A(1,end) % 1st row, last elt. Indexing starts from 1.
A(end,:) % last row
A([1 3],:) % use arrays as indices
A(1:3,1:2) % use sequences as indices
A(:,2) = [10 11 12]'  % change second column
A = [A, [100; 101; 102]]; % append column vec
A = [ones(size(A,1),1),  A];  % e.g bias term in linear regression
A(:) % Select all elements as a column vector.
```

Matrix Operations Matrix operations can be either matrix-wise or element-

wise. Functions that are defined for matrices such as multiplication(*), division (/), or the power operator ($\hat{}$) require a "." before to explicitly to perform the operation element-by-element. Most other functions, including local functions, are element-wise by default.

```
%% matrix operations
C = [C ; C]
A * C % matrix multiplication
B = [5 6; 7 8; 9 10] % same dims as A
A = A(1:3,1:2);
A .* B % element-wise multiplcation
% A .* C  or A * B gives error - wrong dimensions
A .^ 2
1./v
log(v)  % functions like this operate element-wise on vecs or ...
    matrices
exp(v) % e^4
abs(v)

-v  % -1*v

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

A'  % (conjuate) transpose
```

Reshape and Replication

```
%% reshape and replication
A = magic(3)
A = [A [0;1;2]]
reshape(A,[4 3])
reshape(A,[2 6])
v = [100;0;0]
%A+v
A + repmat(v,[1 4])
bsxfun(@plus, A, v)
```

Plotting There are a number of different plots you can create in MATLAB ®. For the purposes of this class especially focus on "plot" and "scatter" - read more at `http://www.mathworks.com/help/matlab/ref/plot.html` and `http://www.mathworks.com/help/matlab/ref/scatter.html`.

```
%% 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');
close;  % or,  "close all" to close all figs

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 Matrix or Image

```
%% display a matrix (or image)
figure;
imagesc(magic(15)), colorbar, colormap flag;
% comma-chaining function calls.
```

Loading and Saving Files The example below assumes the files are already in the current path. If that isn't the case, either add the directory that contains the files to the path or write out the full path in the command (e.g. `load .../path/to/directory/q1x.dat`).

```
%% loading and saving data
load q1y.dat
load q1x.dat
who % displays all workspace variables
whos % displays all workspace variables with details
%clear q1y  % clear w/ no argt clears all
v = q1x(1:10);
save hello v;   % save variable v into file hello.mat
save hello.txt v -ascii; % save as ascii
% fopen, fprintf, fscanf also work
% ls  %% cd, pwd  & other unix commands work in matlab; to ...
    access shell,
% preface with "!"
```

For/While Loops and If Statements Don't try to use for and/or while loops too much. First try to solve the problem using matrices, then if needed use a for or while loop. Think matrix first.

```
%% for, while, if statements

w = [];
```

```
z = 0;
is = 1:10
for i=is
    w = [w, 2*i] % Same as \/
%     w(i) = 2*i
%     w(end+1) = 2*i
    z = z + i;
    break;
    continue;
end
% avoid! same as w = 2*[1:10], z = sum([1:10]);

w = [];
while length(w) < 3
    w = [w, 4];
    break
end

if w(1)==0
    % <statement>
elseif w(1)==1
    % <statement>
else
    % <statement>
end
```

Other Useful Functions

```
%% misc useful functions

% max  (or min)
a = [1 15 2 0.5]
val = max(a)
[val,ind] = max(a)

% find
find(a < 3)
A = magic(3)
[r,c] = find(A>7)

% sum, prod
sum(a)
prod(a)
floor(a) % or ceil(a)
max(rand(3),rand(3))
max(A,[],1)
min(A,[],2)
A = magic(9)
sum(A,1)
sum(A,2)
sum(sum( A .* eye(9) ))
sum(sum( A .* flipud(eye(9)) ))

% pseudo-inverse
```

```
pinv(A)  % inv(A'*A)*A'

% check empty
isempty(e)
numel(A)
size(A)
prod(size(A))
```

Logistic Regression Example

```
% logistic_grad_ascent.m
function [theta, ll] = logistic_grad_ascent(X,y)

% rows of X are training samples
% rows of Y are corresponding 0/1 values

% output ll: vector of log-likelihood values at each iteration
% ouptut theta: parameters

alpha = 0.0001;

[m,n] = size(X);

max_iters = 500;

X = [ones(size(X,1),1), X]; % append col of ones for intercept term

theta = zeros(n+1, 1);  % initialize theta
for k = 1:max_iters
  hx = sigmoid(X*theta);
  theta = theta + alpha * X' * (y-hx);
  ll(k) = sum( y .* log(hx) + (1 - y) .* log(1 - hx) );
end
```

We can then run the logistic gradient ascent and plot the convergence. Note, you can just call the function as long as the .m file associated with the file is in your MATLAB® path.

```
[th,ll] = logistic_grad_ascent(q1x,q1y);
plot(ll)
```

"3-D" Linear Regression Example Below is a basic linear regression function that uses gradient descent to learn a set of parameters `w_learned`.Again the convergence criteria is hard-coded but you might consider more representative convergence criteria in your own code for homeworks.

```
% linear_regress.m
```

```
function w_learned = linear_regress(X,y)
% linear regression model

% Plot the original data
epsilon = 0.0001;
max_iters = 5000;

% Use gradient descent to to learn a set of parameters w_learned
% initialize w_learned randomly
w_learned = randn(2,1);
% iterate for max_iters # of iterations (could use other convergence
% criteria)
for iteration = 1:max_iters
    grad = 2*sum(repmat(w_learned'*X-y,size(X,1),1).*X,2);
    w_learned=w_learned-0.0001*grad;
    err=sum((y-w_learned'*X).^2);
end
```

In the `matlab_session_2014.m` file, we first define a random set of X's and add some random normal noise to a linear model to find y's. After running the model we can plot `y` and `y_hat`, the estimate, on the same 3-D graph.

```
% Create a random normal model (think of it as original data)
X=randn(2,1000);
w= [1;1];
n=randn(1,1000)+0.1;
y=w'*X + n;

% Run linear regression
th = linear_regress(X,y);

% Calculate y_hat
y_hat = th'*X;

% Plot the original data along with the y_hat
scatter3(X(1,:),X(2,:),y);
hold on; % keeps the same figure active - can plot multiple ...
    plots on one axis
scatter3(X(1,:),X(2,:),y_hat,'r');
```

For further information regarding MATLAB® syntax, you can refer to `http://www.mathworks.com/help/matlab/index.html`