

MATLAB[®] Tutorial

Introduction

MATLAB[®] is a high-level language and interactive environment for numerical computation, visualization, and programming. You can also use Octave, which is an open source version of MATLAB[®]

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.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.

```
1  %%% elementary operations
2  5+6
3  3-2
4  5*8
5  1/2
6  2^6
7  1 == 2  % false
8  1 ≠ 2  % true.  note, not "!="
9  1 && 0
10 1 || 0
11 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.

```
1  % variable assignment
2  a = 3; % semicolon suppresses output
3  b = 'hi';
4  c = 3≥1;
5
6  % Displaying them:
7  a = pi
8  disp(sprintf('2 decimals: %0.2f', a))
9  disp(sprintf('6 decimals: %0.6f', a))
10 format long
11 a
12 format short
13 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.

```
1 %% vectors and matrices
2 A = [1 2; 3 4; 5 6]
3 v = [1 2 3]
4 v = [1; 2; 3]
5 v = [1:0.1:2] % from 1 to 2, with stepsize of 0.1. Useful for ...
    plot axes
6 v = 1:6 % from 1 to 6, assumes stepsize of 1
7
8 C = 2*ones(2,3) % same as C = [2 2 2; 2 2 2]
9 w = ones(1,3) % 1x3 vector of ones
10 w = zeros(1,3)
11 w = rand(1,3) % drawn from a uniform distribution
12 w = randn(1,3) % drawn from a normal distribution (mean=0, var=1)
13 w = -6 + sqrt(10)*(randn(1,10000)) % (mean = 1, var = 2)
14 hist(w)
15 e = []; % empty vector
16 I = eye(4) % 4x4 identity matrix
```

Dimensions

```
1 %% dimensions
2 sz = size(A)
3 size(A,1) % number of rows
4 size(A,2) % number of cols
5 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>.

```
1 %% indexing
2 A(3,2) % indexing is (row,col)
3 A(2,:) % get the 2nd row. %% ":" means every elt along that ...
    dimension
4 A(:,2) % get the 2nd col
5 A(1,end) % 1st row, last elt. Indexing starts from 1.
6 A(end,:) % last row
7 A([1 3],:) % use arrays as indices
8 A(1:3,1:2) % use sequences as indices
9 A(:,2) = [10 11 12]' % change second column
10 A = [A, [100; 101; 102]]; % append column vec
11 A = [ones(size(A,1),1), A]; % e.g bias term in linear regression
12 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 (^) require a "." before to explicitly to perform the operation element-by-element. Most other functions, including local functions, are element-wise by default.

```

1 %% matrix operations
2 C = [C ; C]
3 A * C % matrix multiplication
4 B = [5 6; 7 8; 9 10] % same dims as A
5 A = A(1:3,1:2);
6 A .* B % element-wise multiplication
7 % A .* C or A * B gives error - wrong dimensions
8 A .^ 2
9 1./v
10 log(v) % functions like this operate element-wise on vecs or ...
    matrices
11 exp(v) % e^4
12 abs(v)
13
14 -v % -1*v
15
16 v + ones(1,length(v))
17 % v + 1 % same
18
19 A' % (conjugate) transpose

```

Reshape and Replication

```

1 %% reshape and replication
2 A = magic(3)
3 A = [A [0;1;2]]
4 reshape(A,[4 3])
5 reshape(A,[2 6])
6 v = [100;0;0]
7 %A+v
8 A + repmat(v,[1 4])
9 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>.

```

1 %% plotting
2 t = [0:0.01:0.98];
3 y1 = sin(2*pi*4*t);
4 plot(t,y1);
5 y2 = cos(2*pi*4*t);
6 hold on; % "hold off" to turn off

```

```

7 plot(t,y2,'r--');
8 xlabel('time');
9 ylabel('value');
10 legend('sin','cos');
11 title('my plot');
12 close; % or, "close all" to close all figs
13
14 figure(2), clf; % can specify the figure number
15 subplot(1,2,1); % Divide plot into 1x2 grid, access 1st element
16 plot(t,y1);
17 subplot(1,2,2); % Divide plot into 1x2 grid, access 2nd element
18 plot(t,y2);
19 axis([0.5 1 -1 1]); % change axis scale

```

Display Matrix or Image

```

1 %% display a matrix (or image)
2 figure;
3 imagesc(magic(15)), colorbar, colormap flag;
4 % 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/qlx.dat`).

```

1 %% loading and saving data
2 load qly.dat
3 load qlx.dat
4 who % displays all workspace variables
5 whos % displays all workspace variables with details
6 %clear qly % clear w/ no argt clears all
7 v = qlx(1:10);
8 save hello v; % save variable v into file hello.mat
9 save hello.txt v -ascii; % save as ascii
10 % fopen, fprintf, fscanf also work
11 % ls %% cd, pwd & other unix commands work in matlab; to ...
    access shell,
12 % 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.

```

1 %% for, while, if statements
2
3 w = [];

```

```

4  z = 0;
5  is = 1:10
6  for i=is
7      w = [w, 2*i] % Same as \
8      %      w(i) = 2*i
9      %      w(end+1) = 2*i
10     z = z + i;
11     break;
12     continue;
13 end
14 % avoid! same as w = 2*[1:10], z = sum([1:10]);
15
16 w = [];
17 while length(w) < 3
18     w = [w, 4];
19     break
20 end
21
22 if w(1)==0
23     % <statement>
24 elseif w(1)==1
25     % <statement>
26 else
27     % <statement>
28 end

```

Other Useful Functions

```

1  %% misc useful functions
2
3  % max (or min)
4  a = [1 15 2 0.5]
5  val = max(a)
6  [val,ind] = max(a)
7
8  % find
9  find(a < 3)
10 A = magic(3)
11 [r,c] = find(A>=7)
12
13 % sum, prod
14 sum(a)
15 prod(a)
16 floor(a) % or ceil(a)
17 max(rand(3),rand(3))
18 max(A,[],1)
19 min(A,[],2)
20 A = magic(9)
21 sum(A,1)
22 sum(A,2)
23 sum(sum( A .* eye(9) ))
24 sum(sum( A .* flipud(eye(9)) ))
25
26 % pseudo-inverse

```

```

27 pinv(A) % inv(A'*A)*A'
28
29 % check empty
30 isempty(e)
31 numel(A)
32 size(A)
33 prod(size(A))

```

Logistic Regression Example

```

1 % logistic_grad_ascent.m
2 function [theta, ll] = logistic_grad_ascent(X,y)
3
4 % rows of X are training samples
5 % rows of Y are corresponding 0/1 values
6
7 % output ll: vector of log-likelihood values at each iteration
8 % output theta: parameters
9
10 alpha = 0.0001;
11
12 [m,n] = size(X);
13
14 max_iters = 500;
15
16 X = [ones(size(X,1),1), X]; % append col of ones for intercept term
17
18 theta = zeros(n+1, 1); % initialize theta
19 for k = 1:max_iters
20     hx = sigmoid(X*theta);
21     theta = theta + alpha * X' * (y-hx);
22     ll(k) = sum( y .* log(hx) + (1 - y) .* log(1 - hx) );
23 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.

```

1 [th,ll] = logistic_grad_ascent(qlx,qlly);
2 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.

```

1 % linear_regress.m

```

```

2 function wlearned = linear_regress(X,y)
3 % linear regression model
4
5 % Plot the original data
6 epsilon = 0.0001;
7 max_iters = 5000;
8
9 % Use gradient descent to to learn a set of parameters wlearned
10 % initialize wlearned randomly
11 wlearned = randn(2,1);
12 % iterate for max_iters # of iterations (could use other convergence
13 % criteria)
14 for iteration = 1:max_iters
15     grad = 2*sum(repmat(wlearned'*X-y,size(X,1),1).*X,2);
16     wlearned=wlearned-0.0001*grad;
17     err=sum((y-wlearned'*X).^2);
18 end

```

In the `matlab_session.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.

```

1 % Create a random normal model (think of it as original data)
2 X=randn(2,1000);
3 w= [1;1];
4 n=randn(1,1000)+0.1;
5 y=w'*X + n;
6
7 % Run linear regression
8 th = linear_regress(X,y);
9
10 % Calculate y_hat
11 y_hat = th'*X;
12
13 % Plot the original data along with the y_hat
14 scatter3(X(1,:),X(2,:),y);
15 hold on; % keeps the same figure active - can plot multiple ...
16         plots on one axis
17 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>