**Basic Operation**

|  |
| --- |
| **Operator** |
| +, -, \*, /, ^ |
| **Logical operator** |
| ==, ~=, >=, <=, >, <, &&, || |
| **Change prompt output format** |
| PS1(‘>> ’);  >>  >>  … |
| **Assign** |
| a = 3, b = ‘hi’, c = (3 > 1), A = [1 2; 3 4; 5 6] (a 3\*2 matrix), vh = [1 2 3], vv = [1; 2; 3]  v = 1:0.2:2 = [1, 1.2, 1.4, 1.6, 1.8, 2]  Note: ; avoid printing result |
| **Print variable** |
| >> a = pi  >> a  >> disp(a)  >> disp(sprint(‘2 decimal: %0.2f’, a)  >> format long  >> a  Print approx. 14 decimal points  >> format short  >> a  Print approx. 4 decimal points |
| **Matrix** |
| A = [1 2; 3 4; 5 6] (a 3\*2 matrix)  vh = [1 2 3]  vv = [1; 2; 3]  v = 1:0.2:2 = [1, 1.2, 1.4, 1.6, 1.8, 2]  v = priceY(1:10) # slice of first 10 elements  ones (2,3) a 2\*3 matrix with all 1’s  zeros (2,3) a 2\*3 matrix with all 0’s  eye (4) a 4\*4 identity matrix  rand(3,3) a 3\*3 matrix with random number E [0, 1]  randn(3,3) a 3\*3 matrix with random gaussian distribution E [0, 1] |
| **Plotting** |
| Hist (randn(100,100), 50) #plot a histogram of random value, with batch of 50 |
| **Help** |
| Help eye  Help rand  Help help |
| **Matrix Size** |
| A = [1 2; 3 4; 5 6]  s = size(A) = [3, 2] # a row vector, meaning 3 rows, 2 columns  size (s) = [1 2]  size(A, 1) = size(A)(1) = [3, 2](1) = 3  length(s) = 2  length(A) = max(size(A)) = max([3,2]) = 3 |
| **Working directory** |
| Display current directory  >> pwd  Change directory  >> cd ‘C:\Users\deyun\Desktop’  List directories of current directory  >> ls |
| **Load data** |
| Ex. load featureX.dat, priceY, files that contain all features data and h(x) data  >> load (‘featureX.dat’)  >> load (‘priceY.dat’) |
| **Display all variable** |
| >> whos |
| **Clear variable** |
| >> clear featureX #delete featureX variable  >> clear #clear all variables |
| **Save variable** |
| >> save y.mat priceY  >> load y.mat  >> save hello.txt priceY -ascii # save as test(ASCII) |
| **Matrix Manipulation** |
| A = [1 2; 3 4; 5 6]  Indexing  A(2,1) #= 3  A(3,2) #= 6  Columns and rows  A(2, :) #= [3 4] # ‘:’ means all elements within that row/ column  A(: 2) #= [2; 4; 6]  A([1, 3], :) #= [1 2; 5 6] # get all element of first and third row  replacing  A(:, 2) = [3; 5; 7] # assign second column as 3, 5, 7 (replace 2,4,6) |
| **Join 2 matrix** |
| A = A[A, [5; 6; 7]] # append column to the right of A  A (:) #put all element in a matrix into a single column vector  C = [A b] # Join b to the right of A    C = [A; b] #join b at the bottom of A  1 2  3 4  5 6  11 12  13 14  15 16 |
| **Matrix Arithmetic** |
| Matrix scaling  2 \* A; scale by 2  A .^ 2; Square  1 ./ A; Reciprocal  log(A); Element wise logarithm  abs(A); Absolute value of elements in A  A + ones(rows(A), columns(A)); increment every element of A by 1  A + 1; same function as the one above  Matrix Multiplication  A \* B  Matrix Element Multiplication  A .\* B; element in A \* corresponding element in B, A(1, 1) \* B(1, 1)  Matrix Transpose  A’  Solve System of Linear Equations  A \ b; where A is the expression, and b is the answer    Magic(3); # return a 3\*3 matrix that each row / column / diagonal sum to same thing |
| **Matrix Logical Operation** |
| A < 3; = [0 / 1] on each element, prior to the comparison result  find(A<3); #=A row vector, contain elements in A that satisfy < 3  [r, c] = find(A > 3) # r return the row index, c return column index of which element sat. |
| **Matrix Max/Min** |
| **Max/Min in vector:**  Max(A) # = the row that contain the max value, if A is a vector  Val = max(A) # only return the max value if A is a vector  [Val, ind] = max(A) # return max value and the index of the max value if A is a vector  **Max/Min in Matrix:**  Max(rand(3), rand(3)) # return 3\*3 matrix of element-base comparison between 2 matrix  max(A, [], 1) # a row vector contain max value in each column  max(A, [], 2) # A column vector contain max value in each row  max(A(:)) or max(max(A)) # find single max value in a matrix |
| **Sum/product/floor Matrix** |
| Sum(A) # sum of all element  Sum(A, 1) #return a row vector, sum elements of each column  Sum(A, 2) # return column vector, sum elements of each row  sum(sum(magic(9) .\* eye(9))) # sum the diagonal  Prod(A) #product of all element  Floor(A) # round down all element ceil(A) otherwise |
| **Matrix Inverse** |
| pinv(A) |
| **Plot / Visualize Data** |
| **Plot both equation on the same graph**  Plot(xset1, yset1)  Hold on;  Plot(xset2, yset2, ‘r’) #’r’ = display in red  **Plot in different windows**  Figure(1); plot(x1,y1)  Figure(2); plot(x2,y2)  **Graph Labels**  Xlabel(‘time’)  Ylabel(‘J’)  Legend(‘sin’, ’cos’)  Title(‘title’)  **Display matrix in color**  Imagesc(A)    Imagesc(A), colorbar, colormap gray; (use comma chaining of commands)    **Save figure**  Cd ‘desktop’;  Print -dpng ‘myplot.png’  **Clear Figures**  >> clf  **Close figure**  >> close |
| **For Loop** |
| For i = 1 : 10, #from 1 to 10  V(i) = 2 ^ i  end;  index = 1:10  for i = index:  …  End; |
| **While loop and break/continue** |
| i = 1  While i <= 5,  v(i) = 100;  i = i + 1  if i == 6:  break; (or continue)  elseif i == 4  disp(‘another 1’)  else  disp(‘nope’)  end;  End; |
| **Functions** |
| Predefine a file called ‘functionName.m’  **Inside the function**  Function y = functionName(x) # y is what is going to be returned, can be array  y = x ^ 2  **Call the function**  Cd to the directory  functionName(5) |
| **Example: The Cost Function** |
| X = [1 1; 1 2; 1 3]  y = [1 ; 2; 3]  theta = [0; 1]  Function J = costFunction (X, y, theta)  H = X \* theta  squarErrors = (H-Y) .^ 2  J = 1/ (2\*rows(X)) \* sum(squarErrors) |

**Octave: Vectorization – Improve runtime of Octave**

|  |
| --- |
| **Vectorized Simultaneous Update** |
| **General**  Theta = theta-(alpha / m)(X \* (X’ \* theta - y)) |