Basic Command

• Use help or doc to see how it works
  ✓ Using google will be helpful for more information or question

• Use clear, clearVars to remove variables
• Use clc to clean command window
• Use whos to see the memory allocation status
• Use Ctrl + C to stop operation
Matrix Initialization

• Use zeros or ones to produce initialized matrix
  ✓ zeros(3,m) produces 3 x m matrix with zero entries

• Use eye to produce identity matrix
  ✓ eye(5) or eye(5,5) produces 5 x 5 identity matrix

• Direct initialization: use space to specify columns and semicolon to specify rows
  ✓ A=[1 2 3; 4 5 6; 7 8 10]
Type Specification

• Use direct type casting to change data types
  ✓ `uint8(zeros(3,4))` produces unsigned 8-bit integer 3 x 4 matrix
  ✓ `single(eye(2,2))` produces single precision 2 x 2 identity matrix
  ✓ Using appropriate data types may reduce memory cost

• Other data types such as cells, strings, etc
  ✓ May enhance and organize your code a little bit but not necessary for final project actually
Matrix Operation

• Basic operator naturally works on matrix
  ✓ +, -, *
  ✓ / will works as inverse
  ✓ ^ works even for complex value

• Use matrix commands to get information
  ✓ inv(A), det(A), tr(A)
  ✓ A’ will return the transpose of A
  ✓ sum(A), mean(A), var(A) will work for columnwise
Matrix Operation

• Add . for elementwise operation
  ✓ A.+B, X.^3.5, etc...

• Use bsxfun for apply row, column operation
  ✓ bsxfun(@minus, A, mean(A)) will subtract every rows of A by mean(A)
  ✓ @ is similar to lambda calculus (ex : @(x,y) x+y^2)
Extraction

• Use row matrix to specify indices
  ✓ A([1 2],[3 4]) will return 2 x 2 matrix with 1\textsuperscript{st}, 2\textsuperscript{nd} row of 3\textsuperscript{rd}, 4\textsuperscript{th} column of A

• Use : to specify range conveniently
  ✓ 3:10 means [3 4 5 6 7 8 9 10]
  ✓ Use like A(a:b,x:y)
  ✓ For full range, leave : itself like A(a:b,:);
High-Dimension Matrix

• Defining is easy
  ✓ zeros(3,4,5), ones(2,6,7,9)

• But regards first two dimension as main part
  ✓ Handles zeros(10,20,30) as 30 matrix of zeros(10,20)

• Operations may be restricted
  ✓ elementwise operation, plus, or minus will be supported
Matrix Resizing

• Use reshape to change dimension
  ✓ reshape(zeros(8,3),6,4)
  ✓ Number of element must be remained same

• Not need to resize for 1-dimension referencing
  ✓ A(3)+A'(4) works

• But might be need for reorganizing dimension
  ✓ A(1:3,4,5:7) may not be treated as 2-dimensional matrix
Matrix Repeating

• Use repmat to repeat matrix
  ✓ reshape(A,3,4) will repeat A 3 times for row with 4 times for column

• Might be useful for use with bsxfun, reshape
Display

• Intentionally omitting semicolon will display the result
  ✓ A+B will display the result while A+B; will not
  ✓ But this is not a good manner

• Use disp or fprintf to display
  ✓ disp(['The result is',num2str(resultVal)]);
  ✓ fprintf('Test accuracy %f with mode %d \n', acc, mm);
  ✓ http://www.mathworks.co.kr/kr/help/matlab/ref/disp.html
  ✓ http://www.mathworks.co.kr/kr/help/matlab/ref/fprintf.html
Tips & Cautions

• When using high dimensional matrix, try toy problem first
  ✓ You may need to grasp the precise behavior of commands like reshape, sum, etc...
  ✓ Debugging high dimensional matrix is much harder

• MATLAB may not announce overflow or underflow
  ✓ Intelligent use of these will be great but only for rare cases
  ✓ Be careful for underflow – MATLAB will regard as zero
  ✓ Use some linear algebra to avoid these – (ex: logdet, chol)
  ✓ Use various applied or improved version of functions on google
Tips & Cautions

• For function debugging variables will not be displayed
  ✓ When being complicated, try it for main code first
  ✓ After ensuring its correctness, then implement as function

• Ambiguous dimension definition will cause disaster
  ✓ zeros(10) may produce 10 x 10 matrix, not 10 x 1 matrix
  ✓ Need to specify dimension even if it is 1
Tips & Cautions

• You may clear memory variables in script
  ✓ MATLAB will not collect garbage that effectively

• MATLAB is optimized for matrix operation – avoid for loop as much as possible
  ✓ Use matrix and elementwise operation even if it has no linear algebraic meaning
  ✓ But cost for memory allocation also should be considered – trying both cases and taking better one is required
Tips & Cautions

• Executing very long time may stop the program
  ✓ Display the progressing status for unexpected stop

• Pre-allocation vs Post-allocation
  ✓ Pre-allocating variables are usually stable
  ✓ But for various reasons like space costs, post-allocation is inevitable for sometimes

• MATLAB works as interpreter
  ✓ Optimized for matrix operation but not additional intelligence-based optimization