

What you will be doing

- No class Monday
- Next class Wed in MLIB 1110
- Read chapters 2 & 3.1-3.3 in text
- Read chapter 2 Notes
- In class Assignment 3 today
- Complete Assignment 4 as really basic introduction to Matlab
 - Due 23rd

Programming Languages

- Programs written in a specific language are just variations on ways to pass instructions to a computer
- Each language has its own syntax (form) and semantics (meaning)
 - **Syntax:** Sequences of text including words, numbers, and punctuation using rules like written languages (grammar)
 - **Semantics:** The meaning given to the syntax
 - a sequence of words that makes sense to a computer



vs.

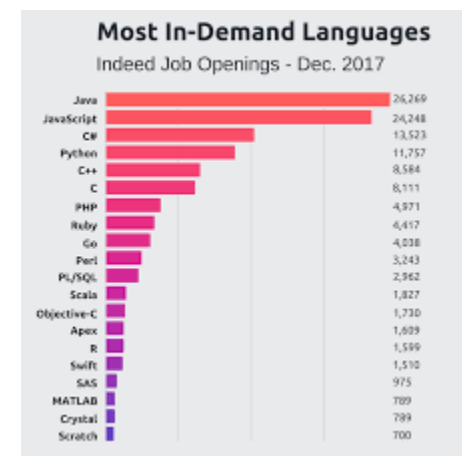


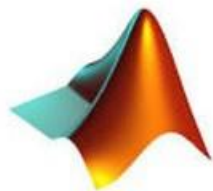
Matlab

- Proprietary
- Inexpensive for educational uses, expensive otherwise
- Flexible graphical user interface
- Many toolboxes
- Matrix oriented

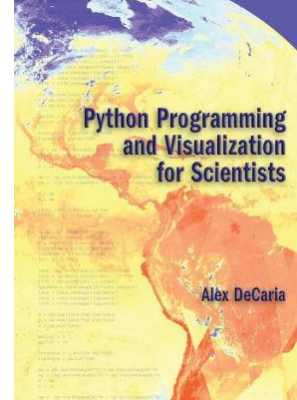
Python

- Open source
- Free, although enterprise releases for commercial applications
- Notebooks are a convenient way to learn
- Many modules
- Object oriented





vs.



In this course:

Matlab

- All codes available in Matlab
- What nearly everyone should use
- Matlab text required

Python

- Nearly all codes available in python
- Some may want to do some exercises using python
- Intro text recommended

- Everything in one github repository (notes, codes, data):
https://github.com/johnhorel/atmos_5040_2019
- Programming language reviews from ATMOS 5020 in repository
- Notes independent of language
- It may be possible to have additional lab time on Wednesdays after the official class period ends

The zen of programming

- Understand what you are expected to do
- Don't start from scratch- use example codes or search for examples
- Read text, look online for techniques and tricks
- There is never only one way to do something- build on what you are comfortable with but seek simpler ways

Debugging

- Programming is iterative
- Start from what works.
 - Get example code working first
- What did you change?
- Have you looked at the underlying data file?
- Have you looked at the workspace variables?
- Did you display some intermediate values?
- Don't assume that if your code runs that it is correct. Does the output make sense?
- Don't expect others to debug your code for you
 - Programming can be like solving a puzzle- it appears hard while you're doing it, but you get some great "ah ha" moments once you've figured it out for yourself
- Being able to work independently and solve problems is critical

Running Matlab

- Find Matlab using the Mac search tool
- What is the directory shown?
 - Change the directory to your desktop
- Becoming comfortable with the Matlab environment
 - Command window
 - Workspace
 - Editor
 - Current Folder

Using github class directory

- Go to https://github.com/johnhorel/ATMOS_5040_2019
- Note the directory “programming help matlab”
- Go to the main class directory
- Select the clone/download tab
 - Select the download zip tab
- On your Mac, move the zip file to the Desktop and then unzip it
- Go to the Chapter 2 subdirectory
- Double click on the jan16_inclass.m file
- Matlab should start up

1.1 Starting MATLAB, MATLAB Windows

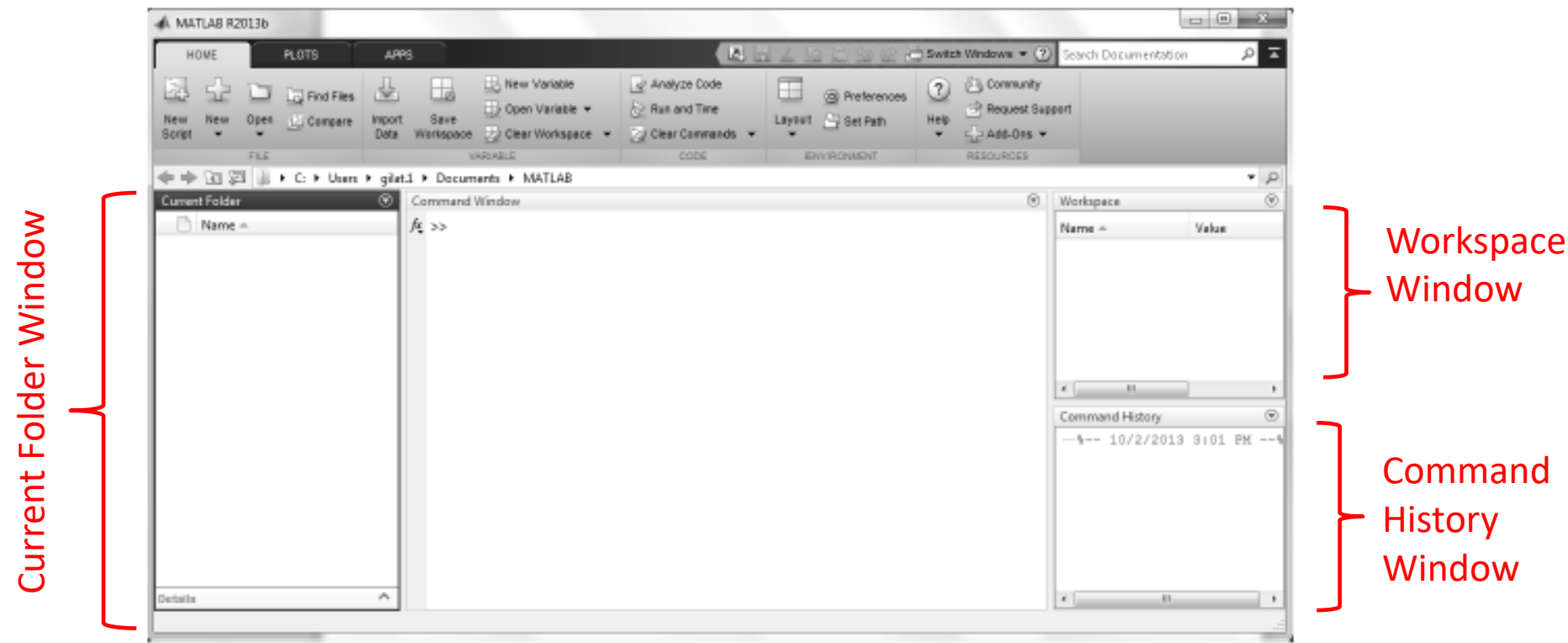


Figure 1-1: The default view of MATLAB desktop.

Command Window

1.1 Starting MATLAB, MATLAB Windows

| Window | Purpose |
|------------------------|---|
| Command Window | Main window, enters variables, runs programs. |
| Figure Window | Contains output from graphic commands. |
| Editor Window | Creates and debugs script and function files. |
| Help Window | Provides help information. |
| Command History Window | Logs commands entered in the Command Window. |
| Workspace Window | Provides information about the variables that are stored. |
| Current Folder Window | Shows the files in the current folder. |

Command Window is MATLAB's main window. Use it to:

- Execute commands
- Open other windows
- Run programs that you've written
- Manage the MATLAB software
 - ver- version. What version of Matlab is installed and toolboxes available

1.2 Working in the Command Window

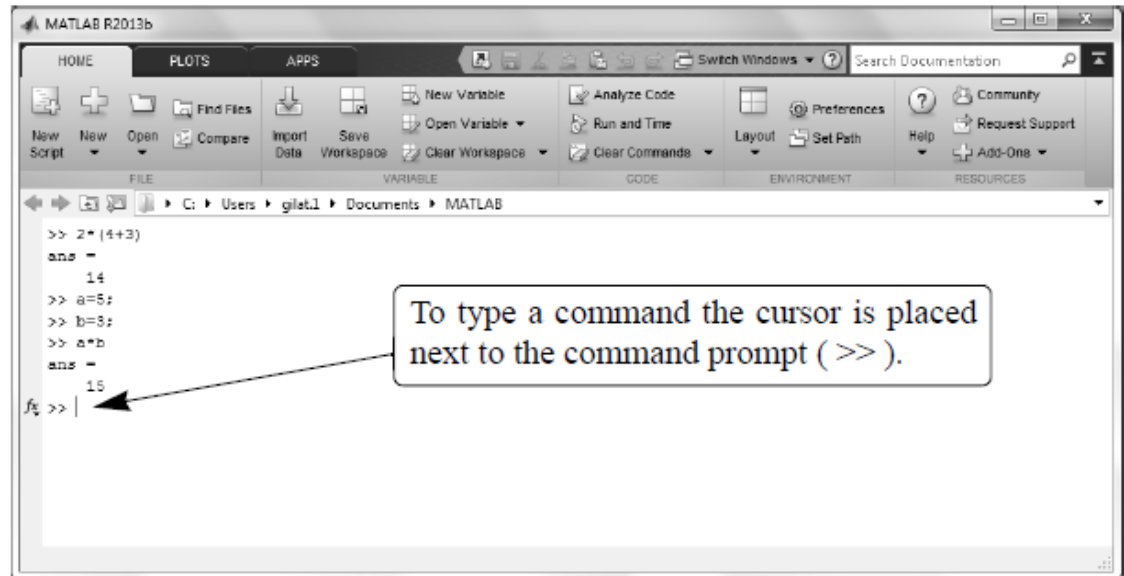


Figure 1-5: The Command Window.

Basic procedure

1. At prompt (`>>`), type in MATLAB command
2. Press ENTER key
3. MATLAB displays result in Command Window, followed by a prompt
4. Repeat from step 1

1.1 Starting MATLAB, MATLAB Windows

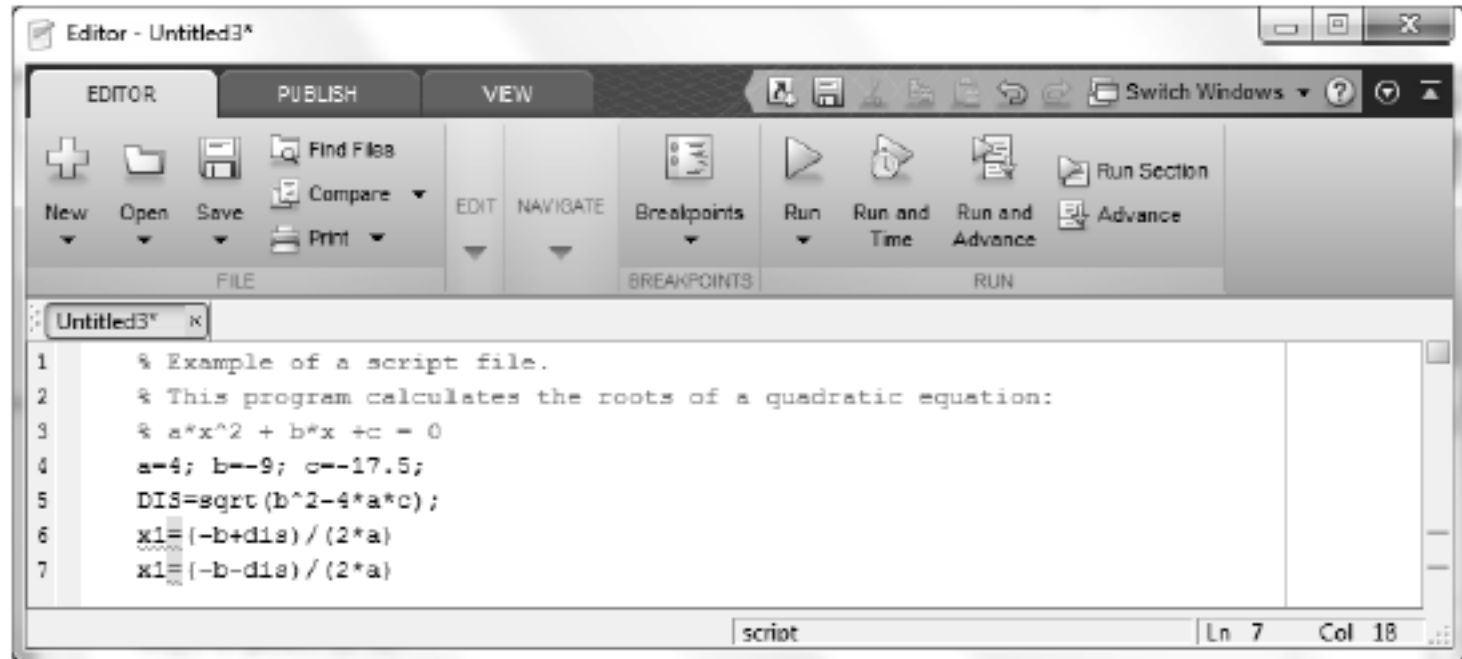


Figure 1-3: Example of an Editor Window.

Use Editor Window to write and debug MATLAB scripts. Open with `edit` command

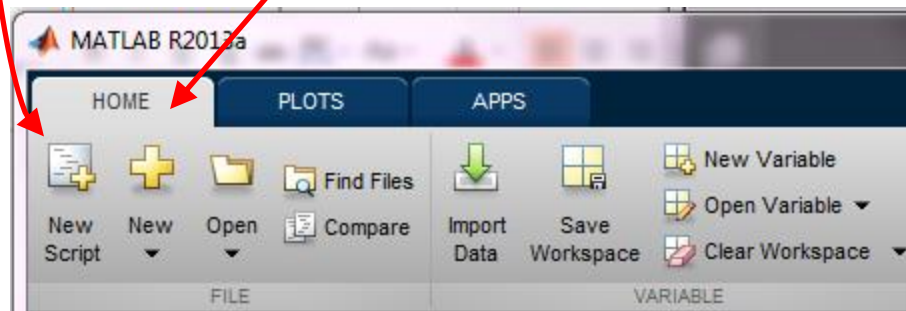
A script file/program/m file is a sequence of MATLAB commands

- When a program runs (is executed), MATLAB performs the commands in the order they are written, just as if they were typed in the Command Window
- When a script file has a command that generates an output (e.g. assignment of a value to a variable without semicolon at the end), the file displays the output in the Command Window

Use the Editor Window to work with script files

Can open window and create file two ways

1. Click on New Script icon
2. Click on New icon, select Script
3. In the Command Window, type `edit` and then press ENTER



Editor has tool strip on top with three tabs – EDITOR, PUBLISH, VIEW

- MATLAB used most often with EDITOR tab selected

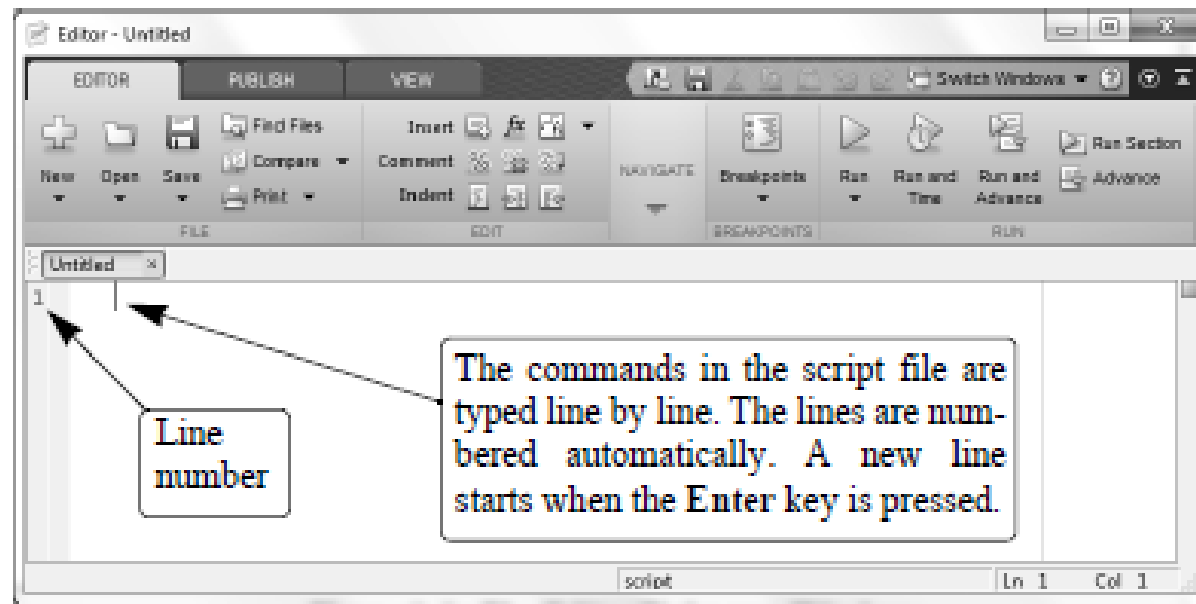


Figure 1-6: The Editor/Debugger Window.

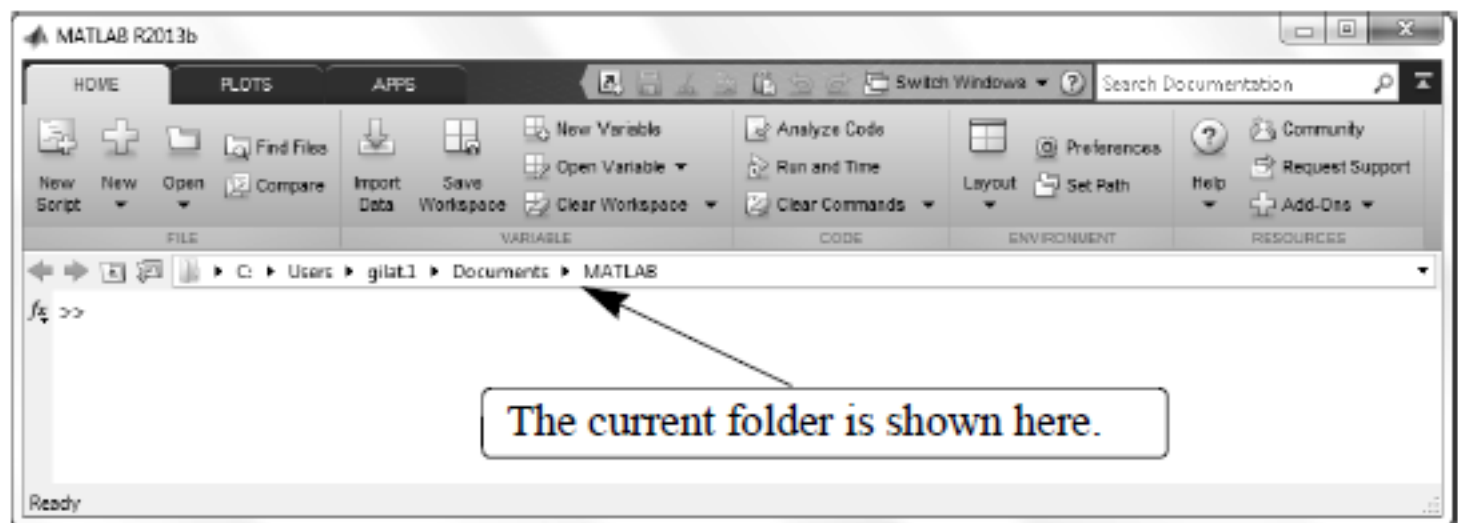


Figure 1-8: The Current folder field in the Command Window.

The *current folder* is the folder that MATLAB checks first when looking for your script file

- Can see current folder in desktop toolbar
- Can also display current folder by issuing MATLAB command `pwd`

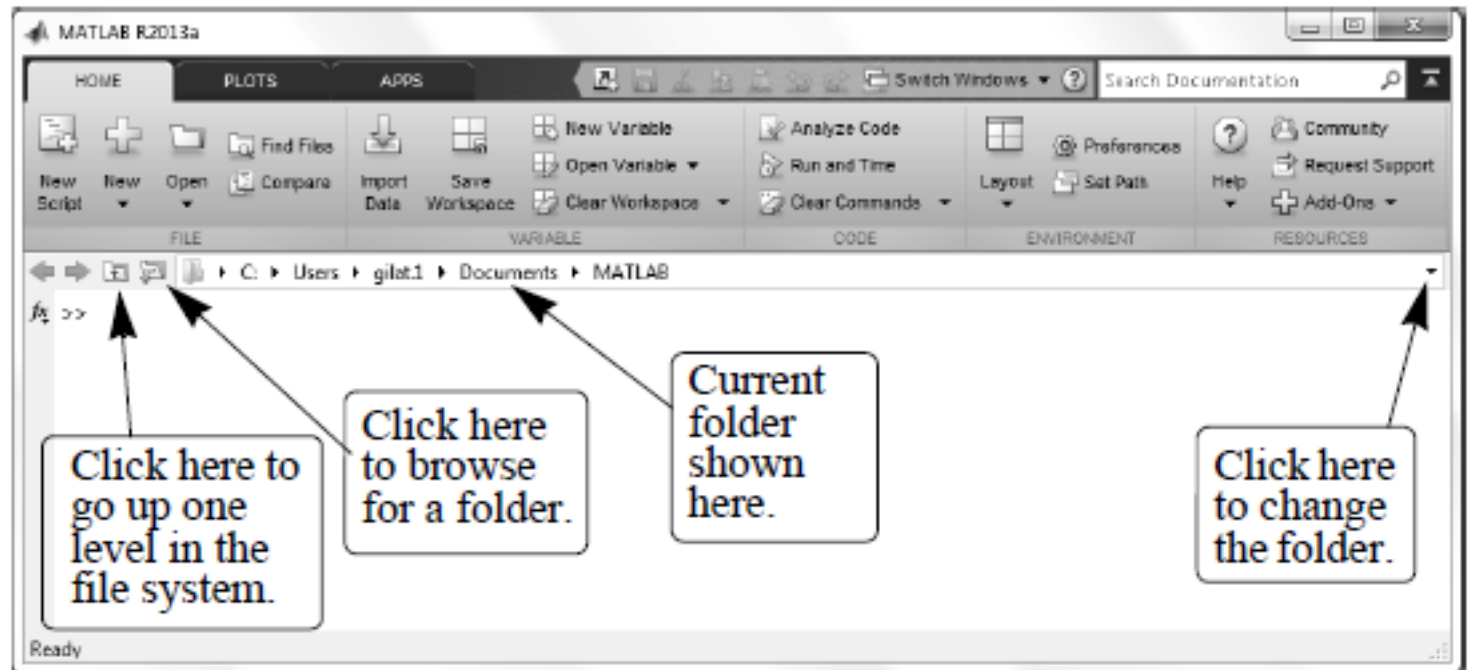


Figure 1-10: The Current Folder Window.

Can change current folder in Current Folder Window

- To show Current Folder Window, click on Layout icon in desktop, then select Current Folder

Matlab File Names and Headers

- Every matlab file you create or edit should contain your name, the date it was created or last edited and a brief description of what the file solves.
 - Remember that many assignments require several files. This guideline applies to all of them.
- Your output in the Command Window should contain your name, the date, course, assignment number, what it does, and what other routines and data files are required. This rule and the previous rule can be accomplished with a single set of commands.
 - `disp('')`
 - `disp('')`
 - `disp(' John Horel')`
 - `disp(' ATMOS 5040/ Spring 2019')`
 - `disp(' Chapter 2')`
 - `disp(' Reproducing code in notes')`
 - `Disp('Requires XXX data file')`

Comments

- Each program should include sufficient comments that the major steps can be followed by a user with a knowledge level equal to your own.
- This doesn't mean each line of code must be commented necessarily. Blocks of code that perform some operation can be commented together.
- Include info on units on numerical values in the code

Graphs

- Basic guidelines of proper graphing (labels, symbol types, titles, legend, etc.) should be followed
- Save in a common format (png)
- Include your name and date in the title of every figure to be turned in

Features for Plotting

- figure
 - plot - Linear plot.
 - subplot
 - loglog - Log-log scale plot.
 - semilogx - Semi-log scale plot.
 - semilogy - Semi-log scale plot.
 - axis - Control axis scaling and appearance.
 - hold - Hold current graph.
 - title - Graph title.
 - xlabel - X-axis label.
 - ylabel - Y-axis label.
 - get - Gets plot properties.
 - set - Sets plot properties.
 - grid
- Color Line Type Marker Type
 - y - yellow - - solid . - point
 - m - magenta : - dotted o - circle
 - c - cyan -. - dashdot x - x-mark
 - r - red -- - dashed + - plus
 - g - green * - star
 - b - blue s - square
 - w - white d - diamond
 - k - black v - triangle (down)
 - ^ - triangle (up)
 - < - triangle (left)
 - > - triangle (right)
 - p - pentagram
 - h - hexagram

Output

- Output should be formatted as to be easily readable.
- Use tabular data when appropriate.
- Include units and report values to the proper number of significant figures.

Can control display of numbers with `format` command

- Once enter command, format stays the same until another `format` command
- Default format is fixed point with four digits to right of decimal point
 - *fixed-point* means decimal point always between one's-digit and one-tenth's digit
- Format only affects display of numbers. MATLAB always computes and saves numbers in full precision

Saving as pdf and uploading to Canvas

- Click on the “Publish” tab
 - Choose “Edit Publishing Options”
 - Change the “Output File Format” to pdf
- Run the program again
 - Verify that the pdf output contains the code and figure with your name on it

January 16 In-class assignment

%clear all the variables

clear all

%close all the figure windows

close all

% jan16 2019 inclass assignment

disp(' ')

disp(' ')

disp(' John Horel')

disp(' ATMOS 5040/ Spring 2019')

disp(' January 16 inclass assignment')

disp(' Requires alta_snow.csv')

January 16 In-class assignment

%for info on the data

%<http://utahavalanchecenter.org/alta-monthly-snowfall>

%download the data from the class github repository

```
data = csvread(' ../data/alta_snow.csv');
```

%column 1 is the year and column 8 is the seasonal total

%create a vector of the years

```
yr = data(:,1);
```

%find out how many years

```
ny = length(yr);
```

%convert the seasonal totals from inches to cm

```
tot = data(:,8)*2.54;
```

January 16 In-class assignment

```
%compute the means of each column
```

```
mmn = mean(data);
```

```
% determine max value and index value of max for the  
winter season
```

```
[maxs,mxi] = max(tot);
```

```
% write to screen the year when max occurred
```

```
fprintf('year of max seasonal snowfall and amount %d %.1f  
cm\n', yr(mxi),maxs)
```

```
%repeat for min
```

```
[mins,mni] = min(tot);
```

```
fprintf('year of min seasonal snowfall and amount %d %.1f  
cm\n', yr(mni),mins)
```

January 16 In-class assignment

```
%plot time series of snow totals as bar chart
```

```
figure(1)
```

```
bar(yr,tot)
```

```
axis('tight')
```

```
title('Alta water year snow total (cm) John Horel  
12/30/2018')
```

```
xlabel('Year')
```

```
ylabel('Snow total (cm)')
```

```
grid
```

```
%then be sure to publish as a pdf and upload to canvas
```

Same thing in python!

- Look online in https://github.com/johnhorel/atmos_5040_2019/tree/master/chapter%202
- Double click on: jan16_2019_inclass.ipynb

What you will be doing

- No class Monday
- Next class Wed in MLIB 1110
- Read chapters 2 & 3.1-3.3 in text
- Read chapter 2 Notes
- In class Assignment 3 today: upload your matlab pdf to canvas
- Complete Assignment 4 as really basic introduction to Matlab
 - Due 23rd

Matlab reference material

Symbols for arithmetic are:

| Operation | Symbol | Example |
|----------------|--------|------------------------------|
| Addition | + | $5 + 3$ |
| Subtraction | − | $5 - 3$ |
| Multiplication | * | $5 * 3$ |
| Right division | / | $5 / 3$ |
| Exponentiation | ^ | $5 ^ 3$ (means $5^3 = 125$) |

Order in which MATLAB does arithmetic

| Precedence | Mathematical Operation |
|------------|--|
| First | Parentheses. For nested parentheses, the innermost are executed first. |
| Second | Exponentiation. |
| Third | Multiplication, division (equal precedence). |
| Fourth | Addition and subtraction. |

Boolean and Logical operators

== Equal

~= Not equal

< Less than

> Greater than

& Logical AND

| Logical OR

~ Logical NOT

Syntax

if - Conditionally execute statements.

else - IF statement condition.

elseif - IF statement condition.

end - Terminate scope of FOR, WHILE, SWITCH, TRY and IF statements.

for - Repeat statements a specific number of times.

while - Repeat statements an indefinite number of times.

break - Terminate execution of WHILE or FOR loop.

continue - Pass control to the next iteration of FOR or WHILE loop.

function - Add new function.

return - Return to invoking function.

error - Display error message and abort function.

disp - Display an array.

feval - Execute function specified by string.

Math

sin - Sine.

asin - Inverse sine.

cos - Cosine.

acos - Inverse cosine.

tan - Tangent.

atan - Inverse tangent.

atan2 - Four quadrant inverse tangent.

sqrt - Square root.

exp - Exponential.

log - Natural logarithm.

log10 - Common (base 10) logarithm.

factorial - Factorial function.

abs - Absolute value.

conj - Complex conjugate.

real - Complex real part.

imag - Complex imaginary part.

floor - Round towards minus infinity.

ceil - Round towards plus infinity.

round - Round towards nearest integer.

mod - Modulus.

norm - Matrix or vector norm.

det - Determinant.

inv - Matrix inverse.

eig - Eigenvalues and eigenvectors.

cross - Vector cross product.

Dot – vector dot product

Rounding functions

- `round(x)` – round to nearest integer
- `fix(x)` – round toward zero
- `ceil(x)` – round toward infinity
- `floor(x)` – round toward minus infinity
- `rem(x, y)` – remainder after `x` is divided by `y` (also called modulus)
- `sign(x)` – returns 1 if `x` is positive, -1 if `x` is negative, zero if `x` is zero

A variable is a name that is assigned a numerical value

- Once assigned, can use variable in expressions, functions, and MATLAB statements and commands
- Can *read* the variable (get its value)
- Can *write to* the variable (set its value)

An *array* is MATLAB's basic data structure

- Can have any number of dimensions. Most common are
 - *vector* - one dimension (a single row or column)
 - *matrix* - two or more dimensions
- Arrays can contain numbers or letters

Matlab terminology

- Scalar: 1×1
- Vector: $m \times 1$ (m rows); $1 \times n$ (n columns)
 - Transpose (') flips vectors;
- Matrix: $m \times n$
- Cell arrays: numerically indexed aggregates of info

- All variables are arrays
 - *Scalar* - array with only one element
 - *Vector* - array with only one row or column
 - *Matrix* - array with multiple rows and columns
- Assigning to variable specifies its dimension
 - Don't have to define variable size before assigning to it, as you do in many programming languages
- Reassigning to variable changes its dimension to that of assignment

Variable Names and Code Structure

- Effective use of variable names can minimize the need for comments.
- Variable names that are descriptive should be used whenever possible.
- Use spaces and indents to improve the readability of your code.
- When input is requested from the user, the units need to be included in the request on the screen, and the format of the input should be specified.

= means “assign to” or “store in”
but not “equals”!

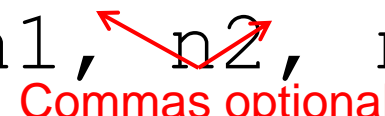
= (equals sign) is MATLAB’s
assignment operator. It evaluates
the expression on its right side and
stores the resulting value in the
variable on its left side

$x = x + 6$ means “take
whatever is in x , add 6 to that and
store the result back into x ”

To create a row vector from known numbers, type variable name, then equal sign, then inside square brackets, numbers separated by spaces and/or commas

```
variable_name = [ n1, n2, n3 ]
```

Commas optional



```
>> yr = [1984 1986 1988 1990 1992 1994 1996]
yr =
    1984    1986    1988    1990    1992    1994    1996
```

Note MATLAB displays row vector horizontally

To create a column vector from known numbers

- Method 1 - same as row vector but put semicolon after all but last number

```
variable_name = [ n1; n2; n3 ]
```

```
>> yr = [1984; 1986; 1988 ]
```

```
yr =
```

```
    1984
```

```
    1986
```

```
    1988
```

Note MATLAB displays column vector vertically

Transpose a variable by putting a single quote after it, e.g., x'

– In math, transpose usually denoted by superscript "T", e.g., x^T

- Converts a row vector to a column vector and vice-versa
- Switches rows and columns of a matrix, i.e., first row of original becomes first column of transposed, second row of original becomes second column of transposed, etc.

`zeros (m, n)` - makes matrix of m rows and n columns, all with zeros

`ones (m, n)` - makes matrix of m rows and n columns, all with ones

`eye (n)` - makes square matrix of n rows and columns. Main diagonal (upper left to lower right) has ones, all other elements are zero

2.2.1 The zeros, ones and, eye Commands

```
>> zr=zeros(3,4)
```

```
zr = 0    0    0    0  
      0    0    0    0  
      0    0    0    0
```

```
>> ne=ones(4,3)
```

```
ne = 1    1    1  
      1    1    1  
      1    1    1  
      1    1    1
```

```
>> idn=eye(5)
```

```
idn = 1    0    0    0    0  
      0    1    0    0    0  
      0    0    1    0    0  
      0    0    0    1    0  
      0    0    0    0    1
```

To make a matrix filled with a particular number, multiply `ones(m,n)` by that number

```
>> z=100*ones(3,4)
```

```
z =
```

| | | | |
|-----|-----|-----|-----|
| 100 | 100 | 100 | 100 |
| 100 | 100 | 100 | 100 |
| 100 | 100 | 100 | 100 |

Address of element is its position in the vector

- "address" often called *index*
- Addresses in Matlab always start at 1 (not 0)
 - Address 1 of row vector is leftmost element
 - Address 1 of column vector is topmost element
- To access element of a vector represented by a variable, follow variables name by address inside parentheses, e.g., $v(2) = 20$ sets second element of vector v to 20

The colon : lets you address a range of elements

- Vector (row or column)
 - $va (:)$ - all elements
 - $va (m:n)$ - elements m through n
- Matrix
 - $A (: , n)$ - all rows of column n
 - $A (m , :)$ - all columns of row m
 - $A (: , m:n)$ - all rows of columns m through n
 - $A (m:n , :)$ - all columns of rows m through n
 - $A (m:n , p:q)$ - columns p through q of rows m through n

Keeping Things Straight

: Span operator [1:5] = [1 2 3 4 5]

() Operation grouping

[] Vector and matrix delimiter

{ } Cell delimiter

. Decimal point

.. Parent directory

... Continuation of command to next line

, Separator

; End line or row

% Comment

= Assignment operator

' - String delimiter

MATLAB has many built-in functions for working with arrays. Some common ones are:

- `length(v)` - number of elements in a vector
- `size(A)` - number of rows and columns in a matrix or vector
- `reshape(A, m, n)` - changes number of rows and columns of a matrix or vector while keeping total number of elements the same. For example, changes 4x4 matrix to 2x8 matrix

- `diag(v)` - makes a square matrix of zeroes with vector in main diagonal
- `diag(A)` - creates vector equal to main diagonal of matrix

For more functions, click on the Help icon, then in the Help window click on MATLAB, then on “MATLAB functions”, then on “By Category”, then scroll down to the section labeled “Matrices and Arrays”

- Use $+$ to add two arrays or to add a scalar to an array
- Use $-$ to subtract one array from another or to subtract a scalar from an array
 - When using two arrays, they must both have the same dimensions (number of rows and number of columns)
 - Vectors must have the same dimensions (rows and columns), not just the same number of elements

When adding two arrays A and B , MATLAB adds the corresponding elements, i.e.,

- It adds the element in the first row and first column of A to the element in the first row and column of B
- It adds the element in the first row and second column of A to the element in the first row and second column of B , etc.

This called *elementwise addition*

When subtracting two arrays A and B , MATLAB performs an elementwise subtraction

In general, an operation between two arrays that works on corresponding elements is called an *elementwise operation*

EXAMPLE

For $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}$ and $B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \end{bmatrix}$

$$A + B = \begin{bmatrix} A_{11} + B_{11} & A_{12} + B_{12} & A_{13} + B_{13} \\ A_{21} + B_{21} & A_{22} + B_{22} & A_{23} + B_{23} \end{bmatrix}$$

$$A - B = \begin{bmatrix} A_{11} - B_{11} & A_{12} - B_{12} & A_{13} - B_{13} \\ A_{21} - B_{21} & A_{22} - B_{22} & A_{23} - B_{23} \end{bmatrix}$$

When adding a scalar to an array, MATLAB adds the scalar to every element of the array

When subtracting a scalar from an array, MATLAB subtracts the scalar from every element of the array

EXAMPLE

For c a scalar and $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}$

$$A + c = \begin{bmatrix} A_{11} + c & A_{12} + c & A_{13} + c \\ A_{21} + c & A_{22} + c & A_{23} + c \end{bmatrix}$$

$$A - c = \begin{bmatrix} A_{11} - c & A_{12} - c & A_{13} - c \\ A_{21} - c & A_{22} - c & A_{23} - c \end{bmatrix}$$

There are two ways of multiplying matrices – matrix multiplication and elementwise multiplication

MATRIX MULTIPLICATION

- Type used in linear algebra
- MATLAB denotes this with asterisk (*)
- Number of columns in left matrix must be same as number of rows in right matrix

Linear algebra rules of array multiplication provide a convenient way for writing a system of linear equations. For example, the following system of three equations with three unknowns:

$$A_{11}x_1 + A_{12}x_2 + A_{13}x_3 = B_1$$

$$A_{21}x_1 + A_{22}x_2 + A_{23}x_3 = B_2$$

$$A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = B_3$$

can be written in a matrix form by:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

and in matrix notation by:

$$AX = B \quad \text{where } A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \text{ and } B = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix}.$$

3.2 Array Multiplication

For example, if A is a 4×3 matrix and B is a 3×2 matrix:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \text{ and } B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix}$$

then the matrix that is obtained with the operation $A*B$ has dimensions 4×2 with the elements:

$$\begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31}) & (A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32}) \\ (A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31}) & (A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32}) \\ (A_{31}B_{11} + A_{32}B_{21} + A_{33}B_{31}) & (A_{31}B_{12} + A_{32}B_{22} + A_{33}B_{32}) \\ (A_{41}B_{11} + A_{42}B_{21} + A_{43}B_{31}) & (A_{41}B_{12} + A_{42}B_{22} + A_{43}B_{32}) \end{bmatrix}$$

A numerical example is:

$$\begin{bmatrix} 1 & 4 & 3 \\ 2 & 6 & 1 \\ 5 & 2 & 8 \end{bmatrix} \begin{bmatrix} 5 & 4 \\ 1 & 3 \\ 2 & 6 \end{bmatrix} = \begin{bmatrix} (1 \cdot 5 + 4 \cdot 1 + 3 \cdot 2) & (1 \cdot 4 + 4 \cdot 3 + 3 \cdot 6) \\ (2 \cdot 5 + 6 \cdot 1 + 1 \cdot 2) & (2 \cdot 4 + 6 \cdot 3 + 1 \cdot 6) \\ (5 \cdot 5 + 2 \cdot 1 + 8 \cdot 2) & (5 \cdot 4 + 2 \cdot 3 + 8 \cdot 6) \end{bmatrix} = \begin{bmatrix} 15 & 34 \\ 18 & 32 \\ 43 & 74 \end{bmatrix}$$

When performing matrix multiplication on two square matrices

- They must both have the same dimensions
- The result is a matrix of the same dimension
- In general, the product is not commutative, i.e., $A * B \neq B * A$

3.2 Array Multiplication

```
>> A = randi(3,3)
```

```
A =
```

| | | |
|---|---|---|
| 3 | 3 | 1 |
| 3 | 2 | 2 |
| 1 | 1 | 3 |

```
>> B=randi(3,3)
```

```
B =
```

| | | |
|---|---|---|
| 3 | 3 | 1 |
| 1 | 2 | 2 |
| 3 | 3 | 3 |

```
>> AB = A*B
```

```
AB =
```

| | | |
|----|----|----|
| 15 | 18 | 12 |
| 17 | 19 | 13 |
| 13 | 14 | 12 |

```
>> BA = B*A
```

```
BA =
```

| | | |
|----|----|----|
| 19 | 16 | 12 |
| 11 | 9 | 11 |
| 21 | 18 | 18 |

```
>> AB == BA
```

```
ans =
```

| | | |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

When performing matrix multiplication on two vectors

- They must both be the same size
- One must be a row vector and the other a column vector
- If the row vector is on the left, the product is a scalar
- If the row vector is on the right, the product is a square matrix whose side is the same size as the vectors

3.2 Array Multiplication

```
>> h = [ 2 4 6 ]  
h =  
      2      4      6  
>> v = [ -1 0 1 ]'  
v =  
     -1  
      0  
      1
```

```
>> h * v  
ans =  
      4  
>> v * h  
ans =  
     -2     -4     -6  
      0      0      0  
      2      4      6
```

Another way of saying *elementwise* operations is *element-by-element* operations

- Addition and subtraction of arrays is always elementwise
- Multiplication, division, exponentiation of arrays can be elementwise
- Both arrays must be same dimension

Do elementwise multiplication, division, exponentiation by putting a period in front of the arithmetic operator

| <u>Symbol</u> | <u>Description</u> | | <u>Symbol</u> | <u>Description</u> |
|---------------|--------------------|--|---------------|--------------------|
| .* | Multiplication | | ./ | Right division |
| .^ | Exponentiation | | .\ | Left Division |

ELEMENTWISE MULTIPLICATION

- Use `. *` to get elementwise multiplication (notice period before asterisk)
- Both matrices must have the same dimensions

```
>> A = [1 2; 3 4];
```

```
>> B = [0 1/2; 1 -1/2];
```

```
>> C = A .* B
```

```
>> C =
```

```
0    1
```

```
3   -2
```

If matrices not same dimension in elementwise multiplication, MATLAB gives error

```
>> A = [ 1 2; 3 4];
```

```
>> B = [1 0]';
```

```
>> A .* B % Meant matrix multiplication!
```

```
??? Error using ==> times
```

```
Matrix dimensions must agree.
```

```
>> A * B % this works
```

```
ans =
```

```
1
```

```
3
```


Be careful – when multiplying square matrices

- Both types of multiplication always work
- If you specify the wrong operator, MATLAB will do the wrong computation and there will be no error!
 - Difficult to find this kind of mistake

Elementwise computations useful for calculating value of a function at many values of its argument

```
>> x=[1:8]
```

Create a vector x with eight elements.

```
x =
```

```
    1    2    3    4    5    6    7    8
```

```
>> y=x.^2-4*x
```

```
y =
```

```
   -3   -4   -3    0    5   12   21   32
```

```
>>
```

Vector x is used in element-by-element calculations of the elements of vector y .

MATLAB has lots of functions for operating on arrays. For a vector v

- `mean(v)` – mean (average)
- `max(v)` – maximum value, optionally with index of maximum
- `min(v)` – minimum value, optionally with index of minimum
- `sum(v)` – sum
- `sort(v)` – elements sorted into ascending order

- `median (v)` – median
- `std (v)` – standard deviation
- `dot (v, w)` – dot (inner product); v, w both vectors of same size but any dimension
- `cross (v, w)` – cross product; v, w must both have three elements but any dimension
- `det (A)` – determinant of square matrix A
- `inv (A)` – inverse of square matrix A