MATLAB Onramp Course

2 – Commands

Enter Commands

* Execute commands by typing them into the **Command Window** after the MATLAB prompt (>>) and pressing **Enter** key
* Unless you specify an output variable 🡪 MATLAB stores results in a variable named **ans**
  + The equal sign (=) is the **assignment operator**
    - The value of the expression on the right of the equal sign is assigned to the variable on the left
* **Workspace browser** (on the right) shows all variables you created
  + Add a semicolon (;) to the end of a command so that the result is not displayed
    - The command will still be executed
    - The variable will still be visible in the Workspace browser
* When you enter just a variable name at the command prompt, MATLAB displays the current value of that variable

Name variables

* Rules:
  + Start with a letter
  + Only contain letters, numbers, and underscores (\_)
* Invalid variable name 🡪 MATLAB will display an error message & a suggested correction

Save and Load Variables

* MAT-file = file format specific to MATLAB
* Save all variables in your workspace to a MAT-file with the **save** command:
  + To save all variables in the workspace to a MAT-file named filename.mat, use the command:  
    >> save filename
  + To save only some variables to a MAT-file, use additional inputs with the commands:  
    >> save filename var1
* Remove all variables from your workspace with the **clear** command:  
   >> clear
* Clean up the Command Window with the **clc** command:  
   >> clc
* Load all variables from a MAT-file with the **load** command:  
   >> load filename
  + To load only some variables from a MAT-file, use additional inputs with the commands:  
    >> load filename var1
* See the contents of any variable by entering the **variable** **name** in the Command Window:  
   >> myvar

Use Built-in Functions and Constants

* Built-in Constants:
  + pi = π = 3.1416
  + i = imaginary number
  + \**note* – the command window output only shows 4 decimal places, but more decimal places are stored internally in MATLAB
    - Can control the displayed precision with the **format** function:  
       >> format long 🡪 Displays 15 decimal places  
       >> format short 🡪 Displays 4 decimal places
* Built-in Functions:
  + abs = Absolute value
  + eig = Eigenvalues
  + sin(x) = sine of x
  + sqrt = Square root

3 – MATLAB Desktop and Editor

MATLAB Desktop and Editor

* Default layout contains 3 main windows:
  + Command Window
  + Workspace
  + Current Folder
* Live script in MATLAB Editor
  + Create: Click ‘New Live Script’ in the Toolstrip
  + Able to run multiple commands at once
    - Enter commands in the gray code box
    - Switch to text mode to add descriptions   
      A close-up of a computer screen

      Description automatically generated
    - Click run 🡪 Will evaluate all commands in order
  + Output:
    - Can customize where output will appear (*via buttons on right side*) – either **(a)** in the right pane, or **(b)** underneath the command that created it in the live script code
    - Click on a result in the output pane 🡪 The command that created it is shown in the editing pane
      * \*Also works vise versa! (click on command in editing pane, will show the result in the output pane)
  + Break script into sections – press ‘**Section Break**’ button
    - Allows you to run just a portion of your code at a time
  + Add a slider: Control 🡪 Numeric Slider

Debug MATLAB Code

* **Errors** – prevent code from running
  + Error icon appears in top right;  
    Code issue is underlined in redd   
    A screenshot of a computer

    Description automatically generated
    - Hover mouse over the message indicator OR the underlined code 🡪 See a description of the error
* **Warning** – script will still run, but warning gives potential improvements to code
  + Warning icon appears in top right;  
    Code with the warning is underlined in orange  
    A screenshot of a computer

    Description automatically generated

4 – Vectors and Matrices

Manually Enter Arrays

* **Arrays** = Basic programming tool in MATLAB language
  + All MATLAB variables are arrays  
    A diagram of array and rows

    Description automatically generated with medium confidence
  + **scalar** = A single number
    - 1-by-1 array (1 row, 1 collum)
    - Ex. x = 4
  + Arrays with multiple elements
    - Create using square brackets [ ]
    - **Row Vector**
      * 1-by-n (1 row, multiple columns)
      * Separate numbers by spaces (and/or commas)
      * x = [3 5 7] = [3, 5, 7] = [3,5,7]
    - **Column Vector**
      * n-by-1 (n rows, 1 column)
      * Separate number with semicolons
      * x = [3;5;7]
    - **Matrix**
      * n-by-m (n rows, m columns)
      * Separate by spaces & semicolons – enter elements row by row
      * x = [3 4 5; 6 7 8]
* Other options:
  + Can perform calculations within the square brackets
    - Example – x = [sqrt(10) pi^2]

Create Evenly Spaced Vectors

* Long Row Vector:
  + **(a)** Type them out individually:  
     >> y = [1 2 3 4 5 6 ….]
  + **(b)** Use the **colon operator** (:) – specify the start value, end value, (optional spacing value)
    - >> y = 1:20
    - Don’t need square brackets!
    - Output = Row vector
    - Spacing between values:
      * Default = spacing of 1
      * Specify different spacing  
         >> y = startnum:spacingnum:endnum
        + Ex. Row vector from 1-5 with a spacing of 0.5  
          >> z = 1:0.5:5
  + **(c)** Use the **linspace** **function** – specify the start value, end value, # elements
    - For when you know the number of elements you want in a vector (rather than the spacing between each element)
    - linspace(first,last,number\_of\_elements)
      * Use commas to separate inputs
      * *Ex*. – create a row vector from 0:1 with 5 elements (spacing of 0.25)  
         >> x = linspace(0,1,5)
    - Output = Row vector
* Long Column Vector:
  + Use the **transpose operator** (‘) on a row vector to convert it into a column vector
    - y = y’ #Transpose an existing row vector  
      x = (1:3)’ #Transpose while creating a row vector – use parentheses to   
       specify the order of operations

Create Arrays with Functions

* **rand()** function = Create a matrix of random numbers
  + x = rand(n) 🡪 Creates a n-by-n matrix of random numbers
  + x = rand(n,m) 🡪 Creates a n-by-m matrix of random numbers (n rows, m columns)
* **zeros()** function = Create a matrix of all zeros
  + x = zeros(n) 🡪 Creates a n-by-n matrix of all zeros
  + x = zeros(n,m) 🡪 Creates a n-by-m matrix of all zeros (n rows, m columns)
* **ones()** function = Create a matrix of all ones
  + x = ones(n) 🡪 Creates a n-by-n matrix of all ones
  + x = ones(n,m) 🡪 Creates a n-by-m matrix of all ones (n rows, m columns)
* **size()** function = Determine the size of an existing matrix
  + size(x) 🡪 ans = n x m
  + *Example*:  
     >> x = rand(2,3)  
     >> size(x) 🡪 ans = 2x3  
     2 3
  + \*Note – can create a matrix with the same size as an existing matrix:  
     >> rand(size(x))

5 – Array Indexing and Modification

Indexing

* Indexing = How you extract & modify elements in an array
* Index = The position of a value inside of an array
  + Always starts at 1
* **Vectors** (row or column)
  + >> x(index) #Extract a single value
  + >> x(first\_index:last\_index) #Extract a range of values
* **Matrix**
  + x(row\_index, column\_index) #Extract a single value
  + x(row\_index, :) #Extract an entire row (nth row & all columns)
  + x(:, column\_index) #Extract an entire column (nth column & all rows)

Index into Arrays

* Can use the MATLAB **end keyword** as an index (referencing the last element)
  + >> y = A(end, 2) #Extracts the value in the last row, 2nd column of matrix A
  + Can use arithmetic with the keyword end:  
    >> y = A(end-1, end-2) #Extracts the value 1 row from the end, 2 columns from the   
     end
* Other:
  + Can use 1 index on a matrix
    - MATLAB traverses down each column in order to get the value
    - *Example*:   
      >> A = [5 6; 7 8]  
      >> A(3) #Output = 6
  + Can use a variable as an index
    - *Example*:  
      >> idx = 8  
      >> data(idx)

Extract Multiple Elements

* The color operator (:)
  + Used as an index value 🡪 Specifies all of the elements in that dimension
    - Vector  
      >> y = B(:) #Extracts all values in the vector
    - Matrix  
      >> x = A(:,2) #Extracts all values in the 2nd column (all rows)
  + Used to specify a range of values:
    - Vector  
      >> y = B(n:m) #Creates a vector containing n-m values
    - Matrix  
      >> x = A(n:m,:) #Creates a matrix containing the n-m rows (all  
       columns)
* Indices can be nonconsecutive numbers
  + Vector  
    >> y = B([1 3 6]) #Creates a vector containing the 1st, 3rd, & 6th values

Change Values in Arrays

* Change the value of an element by combining indexing & assignment (=)
  + Vector
    - >> A(2) = 5
  + Matrix
    - >> B(1,5) = 4

6 – Array Calculations

Perform Array Operations on Vectors

* Add a scalar value to all elements in an array
  + x = x **+** n # Adds n to each element in array x
* Add 2 arrays of the same size
  + v1 **+** v2
* Multiply or divide all elements of an array by a scalar value
  + z = 2**\***x
  + y = x**/**3
* Apply basic statistical functions to a vector to produce a single output
  + **max**()
  + **min**()
* Perform mathematical operations to each element in an array
  + y = **sqrt**(x) # Find the square root of each element in array x
  + z = **round**(x) # Find the rounded values of each element in array x
* Matrix multiplication
  + The **\* operator** – Performs matrix multiplication
    - Only works if the inner dimensions of 2 matrices agree  
      n-by-m \* m-by-k
    - If multiplying 2 equally-sized vectors 🡪 Get an error message
      * Example – Multiplying a 1x2 matrix by a 1x2 matrix  
        >> z = [3 4] \* [10 20]  
        A white background with red text

        Description automatically generated
  + The **.\* operator** – Performs element-wise multiplication
    - Multiplies the corresponding elements of 2 equally sized arrays
    - Example – Multiplying a 1x2 matrix by a 1x2 matrix  
      >> z = [3 4] \* [10 20]  
       z = 30 80

7 – Function Calls

Request Multiple Outputs in Function Calls

* Use the **size()** **function** on a vector/matrix
  + (a) Get a single output variable with the array size
    - 2-element row vector
      * 1st element = # rows
      * 2nd element = # columns
    - >> s = size(x)
  + (b) Get 2 output variables – each containing the size of one dimension (# rows or # columns) of the input array
    - Use square brackets [ ] to request more than 1 output
    - >> [xrow, xcol] = size(x)
* Use the **max() function** on a vector/matrix
  + (a) Get a single output variable with the max value & index value
    - 2-element row vector
      * 1st element = Maximum value
      * 2nd element = Index value of that maximum value
    - >> m = max(x)
  + (b) Get 2 output variables – One with the maximum value & one with the index value of that maximum value
    - Use square brackets [ ] to request more than 1 output
    - >> [xMax, idx] = max(x)
* \*If you only want the second output from a function 🡪 Ignore the first output by using a **tilde** (~) in its place
  + Example – just want the # columns in a matrix  
    >> [~, xcol] = size(x)

8 – Documentation

MATLAB Documentation

* Click the ‘**Help’** button (top panel)
  + Can search for content – all MATLAB functions have a supporting documentation page
    - Supported calling syntaxes
    - Description of those syntaxes
    - Examples of the function
* Use the **doc function** to open the documentation for a MATLAB function
  + >> doc function\_name
  + If you don’t know the name of the function – can search the documentation using phrases
    - Example – search for a function that creates normally distributed numbers (rather than uniformly distributed numbers)  
      >> doc normally distributed numbers

9 - Plots

Plot Vectors

* **plot() function** = Creates a line plot
  + Plot a single vector by itself
    - Uses the vector values as the y-axis data;  
      Sets x-axis data to the range of 1-n (with n= # elements in the vector)
    - >> plot(y)
  + Plot 2 vectors of the same length against each other
    - >> plot(x, y, line\_specifications, line\_preperties\_byName)
  + **Line Specs**: Specify the color, line style, & marker of a plot
    - Ex. “r--o”   
       r = red line  
       -- = dashed line  
       o = circle markers
    - Use different symbols in double quotes as another input
      * The symbols can appear in any order
      * Don’t need all 3 characters
        + Ex. Specify the marker, omit the line style 🡪 Plot shows only the marker (no line)
      * Line Styles:  
        A screenshot of a computer

        Description automatically generated
      * Colors:  
        A screenshot of a computer program

        Description automatically generated
        + RGB triplet = 3-element row vector, elements specify the intensities of the red/green/blue components ranging from   
          0-1
        + Hexadecimal color code = String scalar or character vector, starts with #, followed by 3-6 hexadecimal digits ranging form 0-F
      * Markes:  
        A screenshot of a computer

        Description automatically generated
  + **Line Properties**: Control the appearance & behavior of a Line object
    - Specify optional pairs of arguments as Name-Value arguments
      * Name-Value arguments must appear after other arguments   
        (\*after the line specification)
        + >> plot(v1, Name=Value)
      * If using older MATLAB version (before R2021a):
        + Use commas (,) to separate each name & value argument;  
          Enclose Name in quotes “ “
        + >> plot(v1, “Name”, Value)
    - Arguments for line:
      * **Color** – line color
        + Value options – RGB triplet, hexadecimal color code, color name, or short name

“none” = no color

Or specify a color (same rules/color options as under linespecs)

* + - * **LineStyle** – linestyle
        + Value options

“none” = no line

Or specify a linestyle (same rules/options as under linespecs)

* + - * **LineWidth** – line width
        + Values = A positive value in points (1 point = 1/72 of an inch)

0.5 (default)

* + - * + If line has markers 🡪 LineWidth also affects the marker edges
    - Arguments for markers:
      * **Marker** – marker symbol
        + “none” (default)
        + Or specify as one of the values listed above (under linespecs)
      * **MarkerSize** – marker size
        + Values = A positive value in points (1 point = 1/72 of an inch)

6 (default)

* + - * **MarkerEdgeColor** – marker outline color
        + “auto” (default) = Uses the same color as the **Color** property
        + Or specify a color (same rules/color options under linespecs)
      * **MarkerFaceColor** – marker fill color
        + “none” (default) = No fill
        + “auto” = Uses the same color as the **Color** property of the parent axes
        + Or specify a color (same rules/color options under linespecs)
    - Arguments for legend:
      * **DisplayName** – legend label
        + Legend will not display until you call the **legend** command
        + Values:

‘ ‘ (default) = Sets the label using the form ‘dataN’

Specify as a character vector or string scalar

* + - Arguments for parent/child:
      * **Parent** – parent
        + Axes
        + PolarAxes
        + Group
        + Transform
      * **Children** – children
        + Use this property to view a list of data tips that are plotted on the chart

Returned as an empty GraphicsPlaceholder array, or a DataTip object array

* + - * + Cannot add or remove children using the Children property
  + Each **plot() command** will create a separate plot
    - To plot one line on top of another (in the same axes) – use the **hold on** command
      * >> plot(x1, y1)  
        >> hold on  
        >> plot(x2, y2)
    - To return to the default plot behavior (where each plot appears on its own axes) – use the **hold off** command
* Other types of plots:
  + histogram(x, nbins, options)
  + bar(x, y, width, style, color, options)

Annotate Plots

* Plot annotation functions
  + Add a plot label
    - **title**(“Plot Title”)
  + Add axes labels
    - **xlabel**(“text”) – adds an x-axis label
    - **ylabel**("text”) – adds a y-axis label
  + Add a legend
    - **legend**(label1, label2,…, options)function
* Use the value of a variable in plot annotations by joining text with the variable:
  + Example – use the value of an indexed element in the title  
    >> title(“Sample “ + sample(3) + “ Data”)

10 – Data Import

Import Tool

* To import a file into MATLAB:
  + (a) Select the file from the “current folder” tab & click ‘Import Data’ button
  + (b) Double click on file you want to import from the “current folder” tab
* If you open a \_\_\_ file:
  + .mat file 🡪 The variables it contains are imported into the workspace
  + Image (.jpeg, etc.) 🡪 Imported as a numeric array of pixel values
  + Spreadsheet (.csv, .xlsx) & .txt files 🡪
    - Imports the data as a table (default)
    - Missing data filled in with NaN
    - Automatically chooses an appropriate data type for each variable in the table

Import Data as a Table

* Extract a variable (column) from a table using **dot notation**
  + >> table.VariableName
* Extract rows from a table using regular array indexing
  + >> table(row\_index, :)
* Can create new variables with **dot notation**
  + >> table.NewVar = table.OldVar\*3
* In live script:
  + Can interact with a table by clicking on it in the output pane
    - *Ex*. Sort a table using one of its variables
  + Click the “**Update Code**” button to save your table modifications in your script

11 – Logical Arrays

Logical Indexing

* **Relational operators**
  + <, >, ==, ~=
  + Perform comparisons between 2 values
  + Outcome: 1 (true) or 2 (false)
* Compare an array to a single scalar value
  + Result = Logical array the size of the original array
  + **Logical Indexing** – Use a logical array as an array index
    - MATLAB extracts the array elements where the corresponding index is 1 (true)
    - Array & Scalar value:
      * *Example* – Extract elements of v1 that are greater than 6  
         >> v = v1(v1 > 6)
    - 2 Arrays:
      * *Example* – Extract the elements of v1 corresponding to where v2 is less than 4  
         >> v = v2(v1 < 4)
    - Use to reassign values in an array:
      * Example – replace all values in array x that are equal to 999 with the value of 1  
         >> x(x==999) = 1
* Combine logical comparisons by using **logical operators**
  + AND (&); OR ( | )
  + Example – find values less than 6 and greater than 5  
     >> v1(v1 < 6 & v1 > 5)

12 - Programming

Decision Branching

* **If-Elseif-Else blocks**
  + >> if condition  
     # code for if condition is True  
     else   
     # code for anything else  
     end
  + **if** block = Executes only if the condition is true
  + **elseif** block = Executes only if this condition is true
  + **else** block = Executes if the conditions above are not true

For Loops

* **For loops**
  + >> for i = start#:end#  
     # code on each value of i  
     end
  + The loop body executes n times as the loop counter (ex. i) progresses through the values 1:n
  + To loop over a vector of unknown length, use the **length() function**  
     >> for i = 1:length(x)
  + MATLAB works with arrays naturally – don’t need a for loop for simple code
    - *Example* – calculating the square root for numbers 1-5  
       >> x = 1:5  
       >> xSq = x^2

Note: Add a **slider** to select any column in a matrix

* Add a slider: **Control 🡪 Numeric Slider** in the Live Editor tab
* Right click on slider 🡪 Add min, max, and step value

13 – Final Project

Project = Stellar Motion

* spectra matrix = Has star spectrum data collected at evenly-spaced wavelengths
  + Staring wavelength (lambdaStart) = 630.02
  + Spacing (lambdaDelta) = 0.14
  + Number of observations (nObs) = size(spectra,1)
* Goal = Find the speed (km/s) of a star relative to Earth by using its spectrum
* 1) Find the value of the last wavelength in the recorded spectrum
  + >> lambdaEnd = lambdaStart + (nObs-1)\*lambdaDelta
* 2) Create a vector “lambda” containing the wavelengths in the spectrum
  + >> lambda = lambdaStart:lambdaDelta:lambdaEnd
* 3) Extract the 6th column of spectra to a vector named s
  + Each column of spectra is a spectrum of a different star
  + >> s = spectra(:,6)
* 4) Plot the spectra (s) as a function of wavelength, using point markers & a solid line
  + >> plot(lambda, s, “.-“)
  + >> xlabel(“Wavelength”)
  + >> ylabel(“Intensity”)
* 5) Create variables that contain the minimum value of the spectra & the index of that minimum value
  + >> [sHa, idx] = min(s)
  + The index for the minimum value of the spectra (s) corresponds to the location of the hydrogen-alpha line (lambdaHa)
* 6) Find lambdaHa
  + >> lambdaHa = lambda(idx)
* 7) Add the point/location of the hydrogen-alpha line (lambdaHa, sHa) to the existing axes
  + >> hold on
  + >> plot(lambdaHa, sHa, "rs", MarkerSize = 8)
* 8) Calculate the redshift factor (z) and speed (in km/s) at which the star is moving away from the Earth
  + Redshift factor = the speed of the star relative to Earth
  + Speed of light = 299792.458 km/s
  + >> z = (lambdaHa / 656.28) – 1 # Lab hydrogen-alpha line wavelength   
     = 656.28nm
  + >> speed = z\*299792.458

Project = Compare Stellar Spectra

* Goal = Calculate all the stars’ speeds at once & create a plot of the star spectra
* 1) Calculate the speed of all the stars in the spectra
  + Speed of 1 star
    - >> [sHa,idx] = min(spectra(:,2));  
      >> lambdaHa = lambda(idx);  
      >> z = lambdaHa/656.28 - 1;  
      >> speed = z\*299792.458
  + Speed of all stars – using a for loop
    - >> [nrow, ncol] = size(spectra)  
      >> for i = 1:ncol  
      >> [sHa,idx] = min(spectra(:,2));  
      >> lambdaHa = lambda(idx);  
      >> z = lambdaHa/656.28 - 1;  
      >> speed = z\*299792.458  
      >> end
  + Speed of all stars – using **array operations** (delete (:,2))
    - >> [sHa,idx] = min(spectra);  
      >> lambdaHa = lambda(idx);  
      >> z = lambdaHa/656.28 - 1;  
      >> speed = z\*299792.458
    - Get speed as a 7x1 vector
      * (+) speed = star moving away from Earth (redshifted spectrum)
      * (-) speed = star moving towards Earth (blueshifted spectrum)
* 2) Plot the blueshifted spectra using dashed lines;   
   Plot the redshifted spectra with line width of 3
  + >> for v = 1:7  
    >> s = spectra(:,v)  
    >> if speed(v) <= 0  
    >> plot(lambda,s,”--“)  
    >> elseif speed(v) > 0  
    >> plot(lambda,s,LineWidth = 3)  
    >> end  
    >> hold on  
    >> end  
    >> hold off
* 3) Add a legend
  + The string array “starnames” contains the name of each star in spectra
  + >> legend(starnames)
* 4) Note – could’ve used logical indexing instead of a for loop to determine the names of the redshifted spectra
  + >> starnames(speed > 0)
* 5) If you don’t want to differentiate between redshifted/blueshifed spectra, and just want to plot a line for each column in a matrix:
  + >> plot(lambda, spectra)  
    >> legend(starnames)