Introduction to MATLAB: Applications to Data Science in Finance

2022

University of Glasgow
Adam Smith Business School
Hormoz Ramian[1]

[1]Correspondence: hormoz.ramian@glasgow.ac.uk

Contents

1 Getting Started

Financial and economic decisions are driven by *several* and *intertwined* determinants. A software routine, defined as a collection of machine-executable instructions, assists analysts to examine such multi-dimensional characteristics more efficiently. Assistance provided by software routines has become an exceedingly more valuable part of any empirical or analytical analysis as the number of determinants contributing to desired outcomes of interest are expanding.

This handout serves as a brief overview to introduce basic features needed to develop a software routine in MATLAB language with a focus on working with financial and economic conceptual frameworks and datasets. A more comprehensive description of the software including examples and routines is provided in Attaway (2013).

1.1 Prerequisites

While no prior programming knowledge is needed to access the learning outcomes associated with this text, prior familiarity with matrix calculus and basic statistics is recommended. As the user you will need to install the software either via an institutional subscription such as the university[2], an employer or directly from the *MathWorks*[3].

- Ensure you use university credentials to access the installation resources when completing the installation process.
- Installation process requires around 25GB of free space otherwise it may stop unexpectedly.
- The most recent versions of the software are 2022a and 2021b. Either of these are suitable for any application in data science.

1.2 Environment

Matlab launches for the first time with a default appearance illustrated in Figure (1). The launch window comprises the following 8 main components[4]:

1. **Command Window —** This component is the first channel of interaction with the software where, first, instructions (software commands) can be entered and second, executed outcomes are displayed. This component provides a quick access to software's computational abilities, defining new variables, recalling previously defined variables, and in general interacting with the software in various ways such as accessing help files and documentations.
   1.1. As the first example, users are able to enter `doc` to access the entire documentation home. Alternatively, entering this command followed by the name of any specific command enables the user to access its documentation e.g. `doc clear` prompting the software to navigate to documentations provided for the `clear` command.
   1.2. While the above example leads to Matlab opening an additional window to demonstrate documentations, most commands entered within the command window lead to instant execution and print out of the output in the same window. For instance, entering `pwd` prompts the software to display an output, in this case showing the current default selected working directory. The most recent output in the command window is always stored in a temporary variable labelled as `ans`:

```
>> pwd
ans =
'D:\Users'
```

[2] https://www.gla.ac.uk/myglasgow/it/software/statistics/
[3] https://uk.mathworks.com/products/matlab.html
[4] https://uk.mathworks.com/help/matlab/getting-started-with-matlab.html

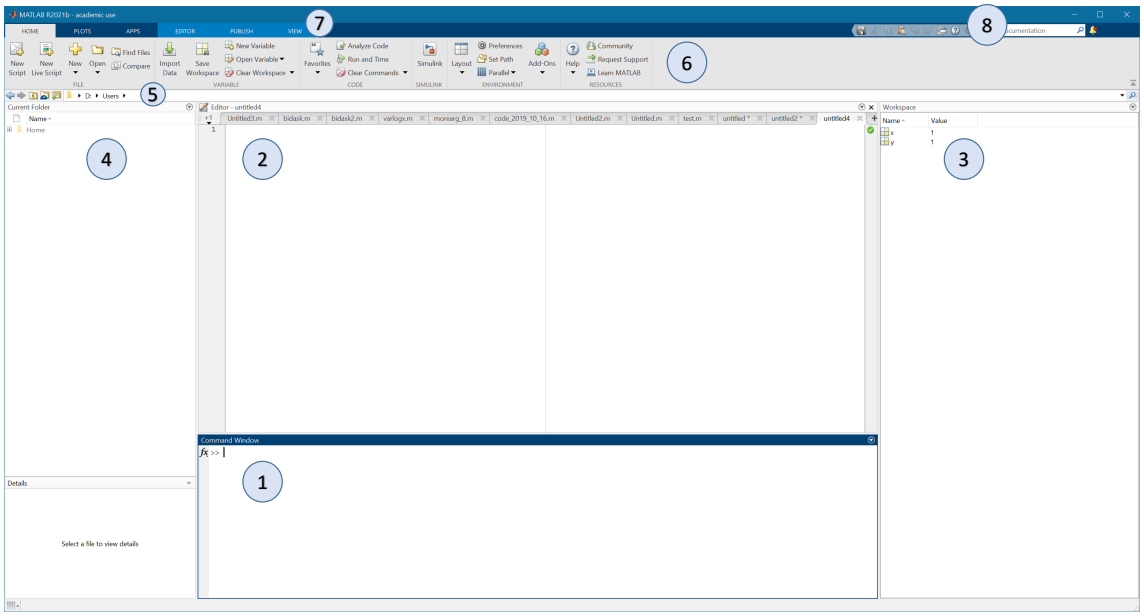


Figure 1: Matlab Default Appearance

where `'D:\Users'` is the answer provided to command `pwd`. This answer is stored in `ans` as a variable available to access for later use under the workspace component (item 3 described below). Entering the name of the variable `ans` inside the command window instructs the software to recall and print out its value on the command window. Note that when a subsequent command in the command window is executed, the result of the new output will replace the original values inside `ans`.

Suppressing the screen print out into the command window can be carried out by using a semi-colon `;` (e.g. `pwd;`) after each entered command. This is a helpful tip since repeatedly printing out results into the command window is often uninformative and slightly slows down the software. Suppressing the results when possible, over many instances can contribute to speed up programmes particularly when a routine is extensively long.

2. **Editor —** This component enables users to enter 'multiple' lines of instructions without executing them one at a time. This component serves an important role to design and develop longer routines which can be executed as a combined set of instructions. For example, a user may wish to develop a routine to import data, apply transformations, generate summary statistics and save the output as an exported file based on multiple executed commands.

   2.1. As a basic example, we are able to write the following five-line programme to instruct the software to clear all existing variables in the memory via the command `clear`. This is an essential way to avoid receiving conflicting computational results when developing a routine in the software and need to *reset* the memory and begin from a clean starting point. `clear` only removes the contents stored in the software memory (specifically everything inside 'Workspace' described under item 3 below) while keeping the routines developed inside the Editor intact. The second line labelled `clc` removes all print outs available on the Command Window while keeping values inside the Workspace and Editor intact. The command on the third line `close all` instructs the software to close all currently open figures produced by the software. The command on the fourth line `cd("c:\")` instructs the software to change the current working directory to a new location specified inside the quotations. This is an essential command partic-

ularly when working with data files, which instructs the software to relocate a default directory where external files e.g. datafiles can be accessed by only calling their files names without specifying their locations. It always is a very good programming habit to check the current working address repeatedly to ensure the software is properly instructed to access and save resources (datafiles, other supplementary routines, exported material, etc.) in an organised fashion.

```
clear
clc
close all
cd('c:\')
```

Note that the above four commands often form the first part of most routines in Matlab since they enable the user to start using the software with a cleared memory, cleared screen, closed figures and at the intended working directory.

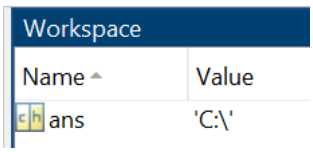
2.2. As a follow up to an earlier note — when a command `cd("...")` is used to instruct the software to relocate to an intended working directory (e.g. `cd("c:\")`), the results can be verified by using `pwd` leading to the the Command Window to show the specified intended location. This is a trivial task but often worth double checking to ensure the command is correctly executed. For example, the intended working directory may be mis-entered or the software may have been unable to successfully switch to the intended location, leading to many subsequent errors since none of the resources in this location can be accessed when the switch is not completed.

3. **Workspace —** This component provides the user with an overview of all existing information stored in the software memory for access (referred to as variables). Variables can be created in several ways. Each variable is a useful resource to save ongoing information on the side to access and use on subsequent steps. Variables in Matlab can be defined as numerical values (numbers, percentages, binary values, or arrays of numbers such as vectors and matrices), text (characters), tables (collection of numerical and textual) and many other types. Workspace provides a complete overview of everything available in memory which can be stored in the following ways:

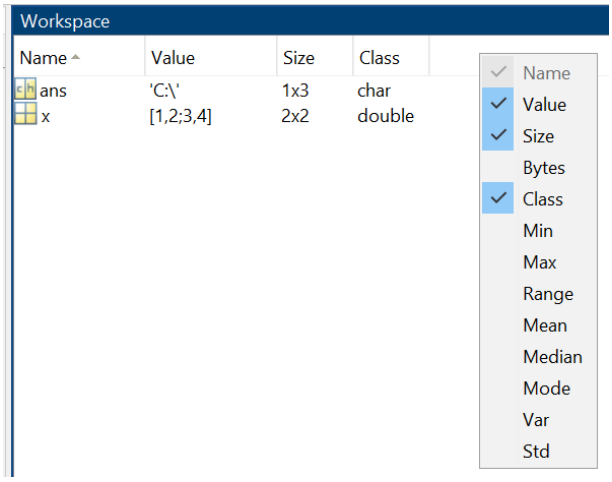
3.1. Similar to an earlier illustration, values directly entered via the Command Window can be stored into the Workspace when any output is stored in the default variable `ans`:

```
pwd
```

leading to the Workspace to show the following where the variable name is `ans`, its value is `cd("c:\")`, and its icon  demonstrates its type as a character type storing a directory address.



3.2. For further illustration, additional variable characteristics can be accessed from the Workspace Window by checking other features provided. For example *size* is an informative characteristic that informs the user about the layout of numerical arrays (e.g. a 2×2 matrix `x` where its icon  indicates the type as a numerical variable).



in the illustration above, the second variable is manually entered into the Workspace Window by entering the following command in the Command Window:

```
1  x=[1,2;3,4]
```

3.3. Importing a dataset into the software also leads to appearance of new variables available to access via the Workspace Window. Furthermore, when carrying out computations, generated results are stored into this window to examination and later use.

3.4. An individual variable can be removed from the Workspace by using the clear command followed by the variable name `clear x` or `clear x ans` to simultaneously remove multiple variables.

4. **Current Directory —** This component provides an overview of contents inside the selected working directory which the user may need to interact with such as datafiles, supplementary routines, and any other resources.
5. **Address Field —** This component is a graphical interface to navigate between local resources via the software and is linked to the earlier command `cd("c:\")` and `pwd`.
6. **Main Toolbar —** This section provides graphical user interfaces to built-in facilities such and *Import Data* used to browse the local resources and access data, or creating a new page within the Editor Window via *New Script*.
7. **Menu Tabs —** This section provides a wider access to graphical user interfaces such as *Editor* tab which shows additional essential features such as *Run* used to execute commands inside the Editor Window.
8. **Quick Access Toolbar —** Lastly, this section enables the user to create shortcuts to any desired interfaces.

1.3 Packages

Packages are independent toolboxes designed to carry out specialized computational tasks (neural networks, deep learning, etc.). The installation process provides an option to select certain Matlab packages or install all available packages. While a package offers a pre-defined framework to carry out computations, all such computations can also be obtained based on a user developed routine. For example, one may rely on a package to obtain a regression coefficients or alternatively develop the regression's computational framework to obtain the same results. Packages offer a quicker way to obtain results, however, given their pre-defined natures, they may not offer adequate flexibility needed for particular tasks thus a user may need to develop a computational framework with the purpose of maintaining intended flexibility.

Nevertheless, it is considered a good programming practice to develop routines and check the results against pre-defined packages whenever possible as a validation tool before further developing a routine towards a more complex purpose.

In some cases, a user may need to explicitly check whether a specific package is properly installed within the software. This is done by entering `ver` command in the Command Window leading to the software to print out all installed packages in the same window.

```
>> ver
-----------------------------------------------------------------------------------------------------
MATLAB Version: 9.11.0.1769968 (R2021b)
MATLAB License Number: 294300
Operating System: Microsoft Windows 10 Pro for Workstations Version 10.0 (Build 19044)
Java Version: Java 1.8.0_202-b08 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode
-----------------------------------------------------------------------------------------------------
MATLAB                                                Version 9.11        (R2021b)
Simulink                                              Version 10.4        (R2021b)
AUTOSAR Blockset                                      Version 2.5         (R2021b)
Bioinformatics Toolbox                                Version 4.15.2      (R2021b)
Control System Toolbox                                Version 10.11       (R2021b)
Curve Fitting Toolbox                                 Version 3.6         (R2021b)
DDS Blockset                                          Version 1.1         (R2021b)
DSP System Toolbox                                    Version 9.13        (R2021b)
Data Acquisition Toolbox                              Version 4.4         (R2021b)
Database Toolbox                                      Version 10.2        (R2021b)
Datafeed Toolbox                                      Version 6.1         (R2021b)
Deep Learning HDL Toolbox                             Version 1.2         (R2021b)
Deep Learning Toolbox                                 Version 14.3        (R2021b)
Econometrics Toolbox                                  Version 5.7         (R2021b)
Embedded Coder                                        Version 7.7         (R2021b)
Filter Design HDL Coder                               Version 3.1.10      (R2021b)
Financial Instruments Toolbox                         Version 3.3         (R2021b)
Financial Toolbox                                     Version 6.2         (R2021b)
Fixed-Point Designer                                  Version 7.3         (R2021b)
Fuzzy Logic Toolbox                                   Version 2.8.2       (R2021b)
Global Optimization Toolbox                           Version 4.6         (R2021b)
Image Acquisition Toolbox                             Version 6.5         (R2021b)
Image Processing Toolbox                              Version 11.4        (R2021b)
Instrument Control Toolbox                            Version 4.5         (R2021b)
MATLAB Coder                                          Version 5.3         (R2021b)
MATLAB Compiler                                       Version 8.3         (R2021b)
Mapping Toolbox                                       Version 5.2         (R2021b)
Model Predictive Control Toolbox                      Version 7.2         (R2021b)
Model-Based Calibration Toolbox                       Version 5.11        (R2021b)
Navigation Toolbox                                    Version 2.1         (R2021b)
Optimization Toolbox                                  Version 9.2         (R2021b)
Parallel Computing Toolbox                            Version 7.5         (R2021b)
Partial Differential Equation Toolbox                 Version 3.7         (R2021b)
Risk Management Toolbox                               Version 1.10        (R2021b)
Signal Processing Toolbox                             Version 8.7         (R2021b)
Spreadsheet Link                                      Version 3.4.6       (R2021b)
Statistics and Machine Learning Toolbox               Version 12.2        (R2021b)
Symbolic Math Toolbox                                 Version 9.0         (R2021b)
```

Each package includes a number of individual routines (functions) accessible to use for specialized computations within that field. In data science we often use the following package:

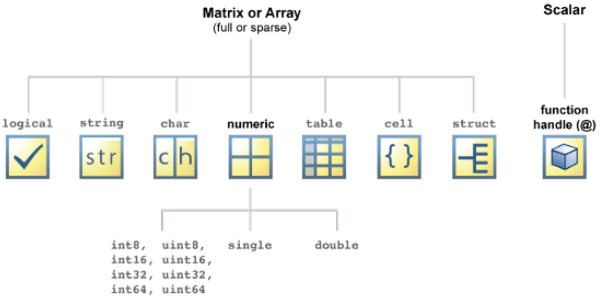
- Curve Fitting Toolbox
- Data Acquisition Toolbox
- Database Toolbox
- Econometrics ToolBox
- Financial Instrument Toolbox
- Financial Toolbox
- Global Optimization Toolbox
- Optimization Toolbox
- Parallel Computing Toolbox

- Statistics and Machine Learning Toolbox

If a tool box is not installed, you are able return to the operating system's software installation section and modify the software to include an individual package without altering the rest of software.

2 Variable Types

Economic and financial data are defined in various types such as numerical values and texts. Matlab enables users to store information in the Workspace in the following ways that include numerical and textual values and additional types based a combination of numbers and texts.



Matlab offers a wide range of variable types suited for various settings such as timeseries, geographical, etc. While this handout describes the following essential types, a full description of all variable types are discussed in the references.[5]

2.1 Numeric

Any information that is quantifiable as a number, where it can be sorted based on a decreasing-increasing order (or vice versa) can be stored as a numerical variable. Numerical variables can be stored as scalars (a single number) or multi-dimensional arrays (vectors and matrices). Consider the following example where a numerical variable collects four features associated with a firm-level dataset: This dataset can be imported directly into the Workspace. However, as an illustration, the following part shows how these values can be manually entered into the software:

Firm	Cash Holding (%)	Financial Constraint (Stress)	Performance (ICR)
1	30.00	2.5	0.2
2	10.00	3.5	0.4
3	30.00	5.5	0.1
4	10.00	-0.5	0.3

```
clear
clc
close all
cd('c:\')
firm = [1;2;3;4]
cashholding = [30;10;30;10]
financialconstraint = [2.5;3.5;5.5;-0.5]
performance=[0.2;0.4;0.1;0.3]
```

[5] https://uk.mathworks.com/help/matlab/data-types.html

where the first four lines are used to prepare the environment with a cleared memory, following by line five `firm = [1;2;3;4]` that instructs the software to create a 4×1-dimensional column vector with numbers 1, 2, 3 and 4 and store them in a variable labelled as `firm`. More specifically, the term column vector indicates that the array is set up as a 4×1 object by using the square brackets `[ ]` and use of semi-colons `;` to separate numbers. Each semi-colon instructs the numerical array to enter the next value on a new row. Contrary to this example, a 1×4-dimensional row vector can be created by following the same steps, except using a comma ',' instead of a semi-colon. For instance entering `[1, 2, 3, 4]` leads to a 4×1-row vector since each comma instructs the software to treat the following number as an entry under a new column. Note that as an exception, commas can be removed — to form an array with numbers separated with simple spaces e.g. `[1 2 3 4]` which instructs Matlab to place each value after a space under a new column.

Variable names such as `firm`, `cashholding`, `financialconstraint` and `performance` can be chosen from any standard naming characters, however, variable names cannot:

- start with a number, e.g. a new variable name `2firm` is unacceptable to the software. As a general rule variable name can include numbers such as `firm2` but may not begin with numbers.
- include spaces, e.g. `cash holding` is an unacceptable naming whereas `cash_holding` including an underscore is acceptable.

Note that assigning values to any existing variable in the workspace leads to replacing the values, without a warning from the software.

Each of the variables defined earlier, `firm`, `cashholding`, `financialconstraint` and `performance` is a 4×1-column vector. We are able to create a new variable that collects all four variables into a new 4×4 matrix. To this end, we treat each existing variable as an input and create a larger array by placing each 4×1 vector next each other:

```
data = [firm  cashholding  financialconstraint  performance]
```

which instructs the software to create a new variable `data` by arranging the existing four variables to show:

```
>> data = [firm  cashholding  financialconstraint  performance]

data =

    1.0000   30.0000    2.5000    0.2000
    2.0000   10.0000    3.5000    0.4000
    3.0000   30.0000    5.5000    0.1000
    4.0000   10.0000   -0.5000    0.3000
```

Note that this is a convenient approach to combine multiple values in a single variable. However, a numerical array is unable to assign column names to identify what is stored under each column and this has to be noted by the user as a side information (meta-data associated with the numerical array). Furthermore, a numerical array, as the name suggests does not allow for any characters or textual entries such as company names that may be of use within financial analysis. A *Table* described later in the text enables the users to both assign column names and store both numeric and textual values at the same time under the same variable.

A numerical array, whether an individual number, a vector (1-dimensional row or column) or a matrix (2 or higher dimensional array), is summarised by the identifiers of its rows, columns and when possible higher dimensions. For instance, `data` defined earlier is a 4×4-dimensional array, where each of its elements can be accessed in the following fashion:

```
>> data(1,1)
ans =
     1
```

```
>> data(1,2)
ans =
    30

>> data(2,1)
ans =
     2

>> data(4,4)
ans =
    0.3000
```

as a generalization, multiple values also are accessible at once, where line number 1 in the following case is accessing the first row and simultaneously both first and second column values.

```
>> data(1,[1 2])
ans =
     1    30

>> data(1,[1 2 3])
ans =
    1.0000   30.0000    2.5000

>> data([2 3],[1 2 3])
ans =
    2.0000   10.0000    3.5000
    3.0000   30.0000    5.5000
```

the fifth line accesses elements on the first row, and 1st-3rd columns at the same time and the last case on line nine, accesses six elements placed on rows 2-3 and columns 1-3 at the same time.

As a more comprehensive way to access element within a numerical array, the following illustrates:

```
>> data(1,1:4)
ans =
    1.0000   30.0000    2.5000    0.2000

>> data(1:end, [2,4])
ans =
   30.0000    0.2000
   10.0000    0.4000
   30.0000    0.1000
   10.0000    0.3000
```

where the first instance `data(1,1:4)` accesses elements on the first row and 1st-to-4th columns at the same time and the second instance `data(1:end, [2,4])` accesses elements identified on the '1st to the last' row, and 2nd & 4th columns.

Similar to the construction of variable `data`, extracted values accessed from this variable can be stored in a new variable. For example, `date2 = data(1:end, [2,4])` is generated as a new 4×2 numerical array collecting all elements from the 1st-to-last rows of `data`, and columns 2 and 4, corresponding to cash holdings and performances of all firms 1-4.

We are able to replace an individual element on a variable e.g. `data` by pointing to the specific element that requires its value to be replaced and equating the expression with the new value:

```
>> data(3,2) = 39.99

data =
    1.0000   30.0000    2.5000    0.2000
```

```
    2.0000   10.0000    3.5000    0.4000
    3.0000   39.9900    5.5000    0.1000
    4.0000   10.0000   -0.5000    0.3000
```

noting that the third element on the second column is changed to 39.99 to amend the cash to total asset ratio previously stored in the data.

Numerical arrays can be added, subtracted, multiplied and inverted according to the elementary matrix operations. This requires the arrays to be consistent in terms of their dimensions, for instance a 4×1-column vector can be added or subtracted from another 4×1-column vector where the addition or subtraction operations applied on an element-by-element correspondence basis. For instance:

```
>> cashholding2 = cashholding + [10;12;18;16]

cashholding2 =

    40
    22
    48
    26
```

leads to adding values 10, 12, 18 and 16 to cash to total assets ratios across firms 1-4. Each numerical variable can be multiplied or divided by an individual multiplier

```
>> cashhodling3 = 0.95 * cashholding

cashhodling3 =

   28.5000
    9.5000
   28.5000
    9.5000
```

where the number `0.95` is multiplied to each individual elements inside `cashholding` to form a new amended variable `cashholding3`.

2.2 Characters and Strings

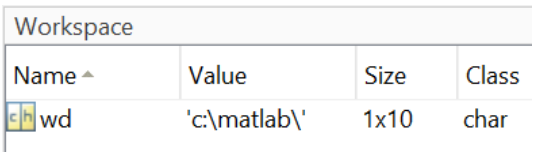
These variable types are designed to work with non-numeric data such as names (individuals, companies, countries) or textual data (government announcements) and any data that is defined within an alphanumeric domain.

For example, a routine can define the working directory as a variable, noting that quotation signs must include the contents:

```
>> wd = 'c:\matlab\'
wd =

    'c:\matlab\'
```

This leads to the workspace to show a new variable called `wd` as a character type storing the directory address. Since this strand of variables is not directly quantifiable, contrary to the numeric type, mathematical operations are not readily applicable.



2.3 Tables

This variable type enables the user to collect a combination of numeric and textual data under the same variable. Aside from accommodating for various types, tables offer additional computational functionalities particularly suited for data scientific and statistical analyses. Example below illustrate a table where inputs are selected from an earlier numeric variable `data` together with manually defined `RowNames` and `VariableNames` leading to `Table1` as a new variable. Note that manually entered 'three-dots' `...` at the end of each line instructs the software to break the line and read the continuation of the same command from the next line. This is a helpful practice to divide longer commands across multiple lines to enhance visual inspection:

```
>> table1 = table(firm,cashholding,financialconstraint,performance, ...
    'RowNames', {'Company A'; 'Company B'; 'Company C'; 'Company D'}, ...
     'VariableNames', {'Firm ID','Cash Holding', 'Financial Constraint', 'Performance'})

table1 =
                  Firm ID    Cash Holding    Financial Constraint    Performance
                  _______    ____________    ____________________    ___________

    Company A       1            30                 2.5                0.2
    Company B       2            10                 3.5                0.4
    Company C       3            30                 5.5                0.1
    Company D       4            10                -0.5                0.3
```

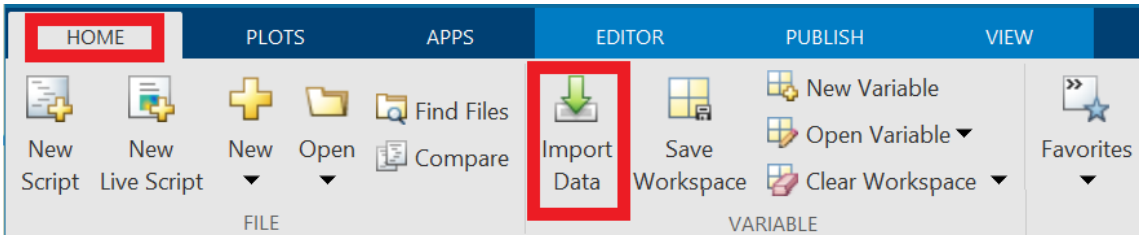
Given the heterogeneous nature of tables, recalling variable values requires the following approach. To recall each column under a table, the name of the table and its column have to be provided and attached using a '.' e.g. `table1.("Cash Holding")`. As an exception, when a column name includes no spaces, the values can be recalled in a simpler way e.g. `table1.Performance` without the quotation marks.

3 Data: Import & Export

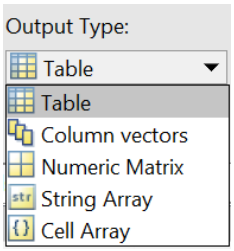
Datasets can be imported to Matlab's workspace both locally (from a hard drive) or directly from a remote source (an online data repository or data warehouse). Each approach offers its own characteristics suited for different purposes. Local access is a more common approach as it offers more flexibility and more security when working with proprietary data whereas a remote access offers a more convenient way and centralised approach particularly for when working as part of a team.

3.1 Local Access

Importing data into Matlab can be carried out via specific commands or a graphical user interface. The nature of a dataset, its structure, and future workflow associated with its purpose determine which approach is more suitable. Matlab's graphical user interface, however, provides an option to import data for a quick use, as well as, auto-generating the associated scripts for a future use. This graphical user interface is accessible via *Import Data* section under *HOME* tab depicted below, followed by a browse window prompting the user to select a data file often provided in a common format such as comma-delimited `(.csv)`, Excel `(.xls)` or `(.xlsx)`, or alternatively Matlab's own storage format `(.mat)`.



The subsequent step involves previewing the dataset within a prompted window displayed in Figure (2). At this stage, the user has the following options to pre-modify the dataset from the *Output Type* section to instruct the software about the intended variable type that the data will be stored in as within the workspace environment. This choice should be made based the nature of data, for example, if the data comprises of only numerical values, a *Numeric Matrix* is an ideal choice, whereas when working with a dataset including both textual and numerical values, then *Table* serves as a more suitable choice.



The section highlighted in the following illustration (Figure 2) as *'Import Selection'* enables the user to either import the data instantly into the workspace by choosing *'Import Data'* that appears as the first item. In this case, the user needs to repeat the steps every time the imported dataset into the software is altered or reset e.g. when using `clear` command. Alternatively, data can be imported together with an auto-generated script associated with these chosen preferences by the software via selecting *'Generate Script'* that appears as the third option in order to use the routine to repeat data importing stage. This option provides a basis for the user to avoid repeating the manual approach particularly when developing an extended computational routine which may require the user to reset and restart the programme multiple times.

The resulting imported data together with an auto-generated scripts will be readily accessible to the user.

3.2 External Data Acquisition

This approach offers a basis to users to bypass downloading and subsequently loading the data into the software via the process described above. Instead, data can be accessed and imported directly into Matlab's workspace based on Matlab's `Data Acquisition Toolbox`. More specifically, each data providing platform offers a *connection* function that is configured to remote access their data repositories, customise certain sub-samples of the available data and load the data. Aside from individual connections, Matlab offers a general purpose connection that is applicable to a wider range of data acquisition via `fetch` command.[6]

The following example demonstrates using `fred(.)` routine to directly access the Federal Reserves St. Louis Economic Data Warehouse. This method requires providing requested variable names, start and end dates and additional search parameters.

```
clc
clear
```

[6] https://uk.mathworks.com/help/datafeed/fred.fetch.html

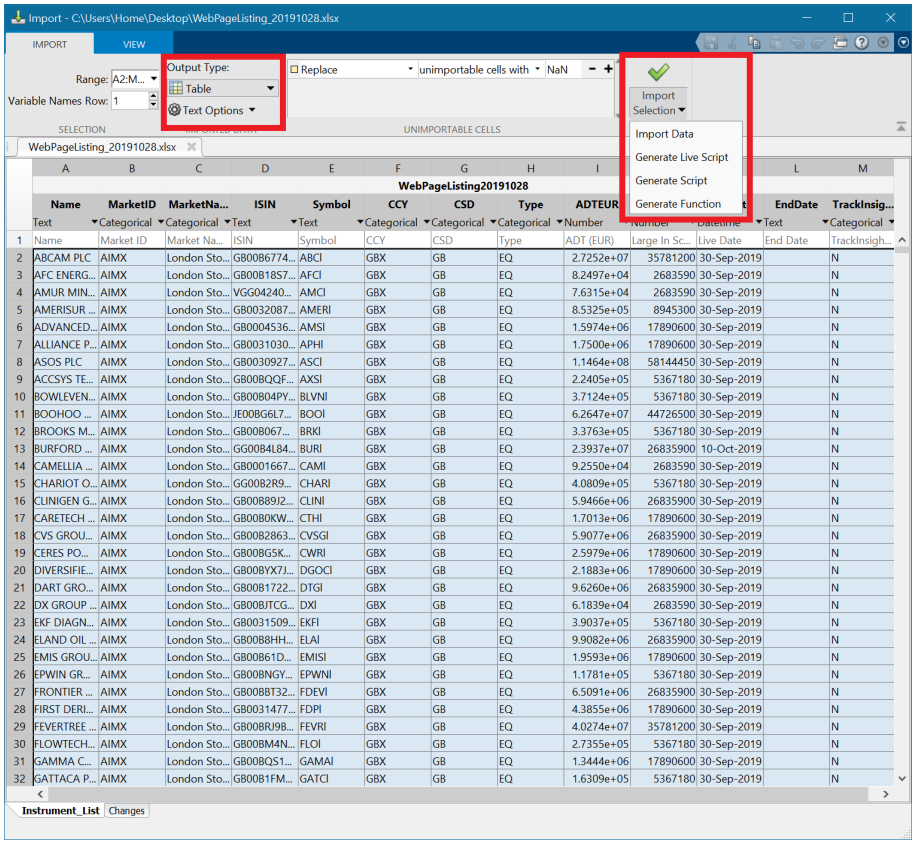


Figure 2: Matlab Data Import Window

Provider	Data	Source
CRSP	US Stocks	www.crsp.com
Commodity Systems Inc.	Futures	www.csidata.com
Datastream	Stocks, bonds, currencies, etc.	www.datastream.com/product/has/
IFM	Futures, US Stocks	www.theifm.org
Olsen & Associates	Currencies, etc.	www.olsen.ch
Trades and Quotes	US Stocks	www.nyse.com/marketinfo
US Federal Reserve	Currencies, etc.	www.federalreserve.gov/releases/

Table 1: Common online data repositories in economics and finance

```
close all

% Define connection to request data from FRED data server:
c  = fred('https://research.stlouisfed.org/fred2/');
sd = '01/01/1948';  % Start Date
ed = '01/12/2018';  % End Date

% FRED Series
s1 = 'UNRATE';      % Civilian Unemployment Rate
s2 = 'INDPRO';      % Industrial Production Index
s3 = 'USRECM';      % NBER-based Recession Indicators for the United States
s4 = 'NASDAQCOM';   % NASDAQ Composite Index
s5 = 'DJIA';        % Dow Jones Industrial Average
s6 = 'RU3000TR';    % Russell 3000 Total Market Index

x1 = 'GDPDEF';
```

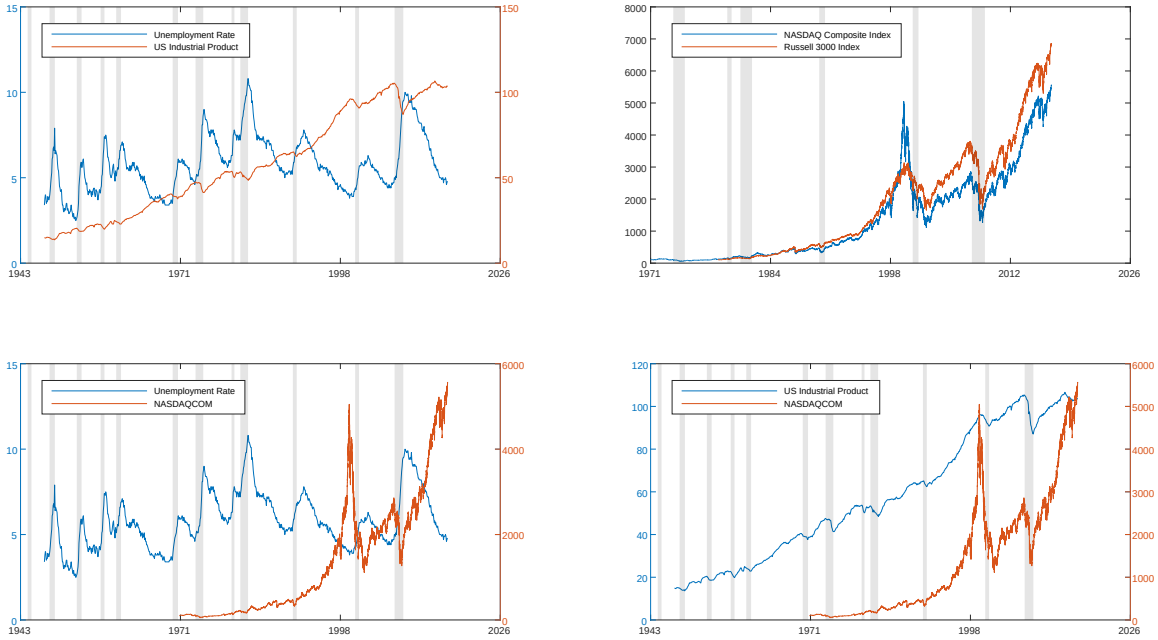


Figure 3: Depicted data based on data acquisition routine described in subsection (3.2).

```
x2 = 'CPIAUCSL';

% Receives data from FRED connection c and the specified FRED series:
d1 = fetch(c,s1,sd,ed);
d2 = fetch(c,s2,sd,ed);
d3 = fetch(c,s3,sd,ed);
d4 = fetch(c,s4,sd,ed);
d5 = fetch(c,s5,sd,ed);
d6 = fetch(c,s6,sd,ed);

X1 = fetch(c,x1,sd,ed);
X2 = fetch(c,x2,sd,ed);

close(c)
% For further comments see:
% https://uk.mathworks.com/help/datafeed/functionlist.html
```

3.3 Exporting Data

Exporting data from Matlab's workspace requires the user to indicate which variable from the workspace is exported into a comma-delimited `.csv`, Excel `.xls` or `.xlsx` by using the following command where *data* in `writematrix(data,'example.xls')` indicates the selected variable and *example.xls* is the name of exported file saved within the current directory.[7] Alternatively, the entire workspace contents can be exported via `save('example')` where the resulting exported file is stored as *example.mat*.[8]

[7] Recall `pwd` which prints out the current working directory assumed by the software.

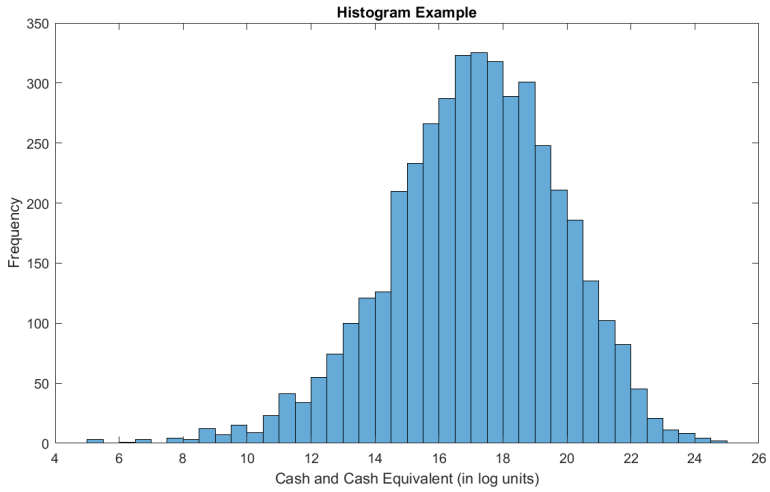
[8] This can later be re-loaded via `load('example')` into the workspace, noting that the working directory must be correctly set up.

4 Visualization

Graphical representations of datasets and conceptual frameworks provide informative perspective to assist data scientific method. Matlab enables users to depict variables in various ways, using histograms, scatterplot, geographical maps, etc. The following part describes two of the most commonly used Matlab diagrams. A comprehensive description of all Matlab diagrams are provided in the references.[9]

Histogram A histogram or a frequency plot provides information about the distribution of a variable across possible observed outcomes versus their counts over that specific outcome. For example the diagram below illustrates how much cash & cash equivalent assets are held by firms (in logarithmic units). The horizontal axis shows these possible outcomes and the vertical axis shows how many firms in the data are holding such amounts as described by their counts between 0-350 per each bar.

```
histogram(log(x.CCE))
xlabel('Cash and Cash Equivalent (in log units)')
ylabel('Frequency')
title('Histogram Example')
```

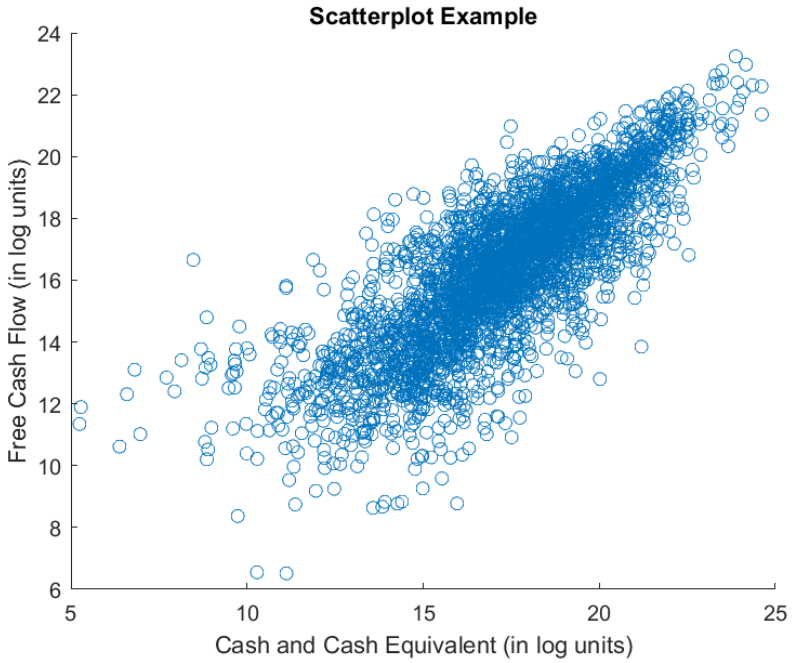


The variable used in this example `x.CCE` refers to a column value under a table variable, imported based on a dataset imported earlier.

Scatterplot A scatterplot is a simple graphical illustration to examine the relationship between two variables by visually inspecting their values versus each other. The following example illustrates firms cash holding versus free cash flows (both in logarithmic units) across all observations in the dataset. In this case, the diagonally upward appearance statistically suggests a positive relationship between the two variables.

```
scatter(log(x.CCE),log(x.FCF))
xlabel('Cash and Cash Equivalent (in log units)')
ylabel('Free Cash Flow (in log units)')
title('Scatterplot Example')
```

[9] https://uk.mathworks.com/help/matlab/creating_plots/types-of-matlab-plots.html.



References

Attaway, S. (2013), *MATLAB: a practical introduction to programming and problem solving*, 3rd edn, Butterworth-Heinemann Ltd, Waltham, MA.

5 Appendix

The following summary by MathWorks provides a brief overview of most commonly used commands and functions by Matlab developers.

MathWorks®

MATLAB® Basic Functions Reference

MATLAB Environment

`clc`	Clear command window
`help fun`	Display in-line help for `fun`
`doc fun`	Open documentation for `fun`
`load("filename","vars")`	Load variables from `.mat` file
`uiimport("filename")`	Open interactive import tool
`save("filename","vars")`	Save variables to file
`clear item`	Remove items from workspace
`examplescript`	Run the script file named `examplescript`
`format style`	Set output display format
`ver`	Get list of installed toolboxes
`tic, toc`	Start and stop timer
`Ctrl+C`	Abort the current calculation

Operators and Special Characters

`+, -, *, /`	Matrix math operations
`.*, ./`	Array multiplication and division (element-wise operations)
`^, .^`	Matrix and array power
`\`	Left division or linear optimization
`.', '`	Normal and complex conjugate transpose
`==, ~=, <, >, <=, >=`	Relational operators
`&&, ||, ~, xor`	Logical operations (AND, NOT, OR, XOR)
`;`	Suppress output display
`...`	Connect lines (with break)
`% Description`	Comment
`'Hello'`	Definition of a character vector
`"This is a string"`	Definition of a string
`str1 + str2`	Append strings

Special Variables and Constants

`ans`	Most recent answer
`pi`	π=3.141592654...
`i, j, 1i, 1j`	Imaginary unit
`NaN, nan`	Not a number (i.e., division by zero)
`Inf, inf`	Infinity
`eps`	Floating-point relative accuracy

Defining and Changing Array Variables

`a = 5`	Define variable a with value 5

`A = [1 2 3; 4 5 6]`
`A = [1 2 3`
`4 5 6]`
	Define A as a 2x3 matrix
	"space" separates columns
	";" or new line separates rows

`[A,B]`	Concatenate arrays horizontally
`[A;B]`	Concatenate arrays vertically
`x(4) = 7`	Change 4th element of x to 7
`A(1,3) = 5`	Change A(1,3) to 5
`x(5:10)`	Get 5th to 10th elements of x
`x(1:2:end)`	Get every 2nd element of x (1st to last)
`x(x>6)`	List elements greater than 6
`x(x==10)=1`	Change elements using condition
`A(4,:)`	Get 4th row of A
`A(:,3)`	Get 3rd column of A
`A(6, 2:5)`	Get 2nd to 5th element in 6th row of A
`A(:,[1 7])=A(:,[7 1])`	Swap the 1st and 7th column
`a:b`	[a, a+1, a+2, ..., a+n] with a+n≤b
`a:ds:b`	Create regularly spaced vector with spacing ds
`linspace(a,b,n)`	Create vector of n equally spaced values
`logspace(a,b,n)`	Create vector of n logarithmically spaced values
`zeros(m,n)`	Create m x n matrix of zeros
`ones(m,n)`	Create m x n matrix of ones
`eye(n)`	Create a n x n identity matrix
`A=diag(x)`	Create diagonal matrix from vector
`x=diag(A)`	Get diagonal elements of matrix
`meshgrid(x,y)`	Create 2D and 3D grids
`rand(m,n), randi`	Create uniformly distributed random numbers or integers
`randn(m,n)`	Create normally distributed random numbers

Complex Numbers

`i, j, 1i, 1j`	Imaginary unit
`real(z)`	Real part of complex number
`imag(z)`	Imaginary part of complex number
`angle(z)`	Phase angle in radians
`conj(z)`	Element-wise complex conjugate
`isreal(z)`	Determine whether array is real

MathWorks®

Elementary Functions

`sin(x), asin`	Sine and inverse (argument in radians)
`sind(x), asind`	Sine and inverse (argument in degrees)
`sinh(x), asinh`	Hyperbolic sine and inverse (arg. in radians)
Analogous for the other trigonometric functions: `cos`, `tan`, `csc`, `sec`, and `cot`	
`abs(x)`	Absolute value of x, complex magnitude
`exp(x)`	Exponential of x
`sqrt(x), nthroot(x,n)`	Square root, real nth root of real numbers
`log(x)`	Natural logarithm of x
`log2(x), log10`	Logarithm with base 2 and 10, respectively
`factorial(n)`	Factorial of n
`sign(x)`	Sign of x
`mod(x,d)`	Remainder after division (modulo)
`ceil(x), fix, floor`	Round toward +inf, 0, -inf
`round(x)`	Round to nearest decimal or integer

Tables

`table(var1,...,varN)`	Create table from data in variables var1, ..., varN
`readtable("file")`	Create table from file
`array2table(A)`	Convert numeric array to table
`T.var`	Extract data from variable var
`T(rows,columns)`, `T(rows,["col1","coln"])`	Create a new table with specified rows and columns from T
`T.varname=data`	Assign data to (new) column in T
`T.Properties`	Access properties of T
`categorical(A)`	Create a categorical array
`summary(T), groupsummary`	Print summary of table
`join(T1, T2)`	Join tables with common variables

Tasks (Live Editor)

Live Editor tasks are apps that can be added to a live script to interactively perform a specific set of operations. Tasks represent a series of MATLAB commands. To see the commands that the task runs, show the generated code.

Common tasks available from the Live Editor tab on the desktop toolstrip:

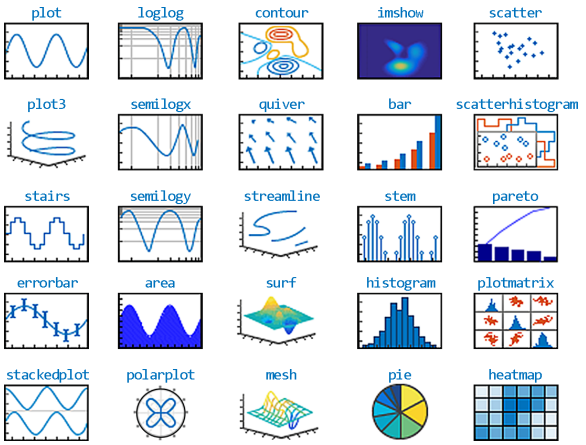
- Clean Missing Data
- Clean Outlier
- Find Change Points
- Find Local Extrema
- Remove Trends
- Smooth Data

Plotting

`plot(x,y,LineSpec)`
Line styles:
`-, --, :, -.`
Markers:
`+, o, *, ., x, s, d`
Colors:
`r, g, b, c, m, y, k, w`
	Plot y vs. x
	(`LineSpec` is optional)
	`LineSpec` is a combination of `linestyle`, `marker`, and `color` as a string.
	Example: `"-r"`
	= red solid line without markers

`title("Title")`	Add plot title
`legend("1st", "2nd")`	Add legend to axes
`x/y/zlabel("label")`	Add x/y/z axis label
`x/y/zticks(ticksvec)`	Get or set x/y/z axis ticks
`x/y/zticklabels(labels)`	Get or set x/y/z axis tick labels
`x/y/ztickangle(angle)`	Rotate x/y/z axis tick labels
`x/y/zlim`	Get or set x/y/z axis range
`axis(lim), axis style`	Set axis limits and style
`text(x,y,"txt")`	Add text
`grid on/off`	Show axis grid
`hold on/off`	Retain the current plot when adding new plots
`subplot(m,n,p)`, `tiledlayout(m,n)`	Create axes in tiled positions
`yyaxis left/right`	Create second y-axis
`figure`	Create figure window
`gcf, gca`	Get current figure, get current axis
`clf`	Clear current figure
`close all`	Close open figures

Common Plot Types



Plot Gallery: *mathworks.com/products/matlab/plot-gallery*

MathWorks®

Programming Methods

Functions

```
% Save your function in a function file or at the end
% of a script file. Function files must have the
% same name as the 1st function
function cavg = cumavg(x) %multiple args. possible
    cavg=cumsum(vec)./(1:length(vec));
end
```

Anonymous Functions

```
% defined via function handles
fun = @(x) cos(x.^2)./abs(3*x);
```

Control Structures

if, elseif Conditions

```
if n<10
    disp("n smaller 10")
elseif n<=20
    disp("n between 10 and 20")
else
    disp("n larger than 20")
```

Switch Case

```
n = input("Enter an integer: ");
switch n
    case -1
        disp("negative one")
    case {0,1,2,3} % check four cases together
        disp("integer between 0 and 3")
    otherwise
        disp("integer value outside interval [-1,3]")
end % control structures terminate with end
```

For-Loop

```
% loop a specific number of times, and keep
% track of each iteration with an incrementing
% index variable
for i = 1:3
    disp("cool");
end % control structures terminate with end
```

While-Loop

```
% loops as long as a condition remains true
n = 1;
nFactorial = 1;
while nFactorial < 1e100
    n = n + 1;
    nFactorial = nFactorial * n;
end % control structures terminate with end
```

Further programming/control commands

`break`	Terminate execution of for- or while-loop
`continue`	Pass control to the next iteration of a loop
`try, catch`	Execute statements and catch errors

Numerical Methods

`fzero(fun,x0)`	Root of nonlinear function
`fminsearch(fun,x0)`	Find minimum of function
`fminbnd(fun,x1,x2)`	Find minimum of fun in [x1, x2]
`fft(x), ifft(x)`	Fast Fourier transform and its inverse

Integration and Differentiation

`integral(f,a,b)`	Numerical integration (analogous functions for 2D and 3D)
`trapz(x,y)`	Trapezoidal numerical integration
`diff(X)`	Differences and approximate derivatives
`gradient(X)`	Numerical gradient
`curl(X,Y,Z,U,V,W)`	Curl and angular velocity
`divergence(X,..,W)`	Compute divergence of vector field
`ode45(ode,tspan,y0)`	Solve system of nonstiff ODEs
`ode15s(ode,tspan,y0)`	Solve system of stiff ODEs
`deval(sol,x)`	Evaluate solution of differential equation
`pdepe(m,pde,ic,... bc,xm,ts)`	Solve 1D partial differential equation
`pdeval(m,xmesh,... usol,xq)`	Interpolate numeric PDE solution

Interpolation and Polynomials

`interp1(x,v,xq)`	1D interpolation (analogous for 2D and 3D)
`pchip(x,v,xq)`	Piecewise cubic Hermite polynomial interpolation
`spline(x,v,xq)`	Cubic spline data interpolation
`ppval(pp,xq)`	Evaluate piecewise polynomial
`mkpp(breaks,coeffs)`	Make piecewise polynomial
`unmkpp(pp)`	Extract piecewise polynomial details
`poly(x)`	Polynomial with specified roots x
`polyeig(A0,A1,...,Ap)`	Eigenvalues for polynomial eigenvalue problem
`polyfit(x,y,d)`	Polynomial curve fitting
`residue(b,a)`	Partial fraction expansion/decomposition
`roots(p)`	Polynomial roots
`polyval(p,x)`	Evaluate poly p at points x
`polyint(p,k)`	Polynomial integration
`polyder(p)`	Polynomial differentiation

MathWorks®

Accelerating the pace of engineering and science

Matrices and Arrays

`length(A)`	Length of largest array dimension
`size(A)`	Array dimensions
`numel(A)`	Number of elements in array
`sort(A)`	Sort array elements
`sortrows(A)`	Sort rows of array or table
`flip(A)`	Flip order of elements in array
`squeeze(A)`	Remove dimensions of length 1
`reshape(A,sz)`	Reshape array
`repmat(A,n)`	Repeat copies of array
`any(A), all`	Check if any/all elements are nonzero
`nnz(A)`	Number of nonzero array elements
`find(A)`	Indices and values of nonzero elements

Linear Algebra

`rank(A)`	Rank of matrix
`trace(A)`	Sum of diagonal elements of matrix
`det(A)`	Determinant of matrix
`poly(A)`	Characteristic polynomial of matrix
`eig(A), eigs`	Eigenvalues and vectors of matrix (subset)
`inv(A), pinv`	Inverse and pseudo inverse of matrix
`norm(x)`	Norm of vector or matrix
`expm(A), logm`	Matrix exponential and logarithm
`cross(A,B)`	Cross product
`dot(A,B)`	Dot product
`kron(A,B)`	Kronecker tensor product
`null(A)`	Null space of matrix
`orth(A)`	Orthonormal basis for matrix range
`tril(A), triu`	Lower and upper triangular part of matrix
`linsolve(A,B)`	Solve linear system of the form AX=B
`lsqminnorm(A,B)`	Least-squares solution to linear equation
`qr(A), lu, chol`	Matrix decompositions
`svd(A)`	Singular value decomposition
`gsvd(A,B)`	Generalized SVD
`rref(A)`	Reduced row echelon form of matrix

Descriptive Statistics

`sum(A), prod`	Sum or product (along columns)
`max(A), min, bounds`	Largest and smallest element
`mean(A), median, mode`	Statistical operations
`std(A), var`	Standard deviation and variance
`movsum(A,n), movprod, movmax, movmin, movmean, movmedian, movstd, movvar`	Moving statistical functions n = length of moving window
`cumsum(A), cumprod, cummax, cummin`	Cumulative statistical functions
`smoothdata(A)`	Smooth noisy data
`histcounts(X)`	Calculate histogram bin counts
`corrcoef(A), cov`	Correlation coefficients, covariance
`xcorr(x,y), xcov`	Cross-correlation, cross-covariance
`normalize(A)`	Normalize data
`detrend(x)`	Remove polynomial trend
`isoutlier(A)`	Find outliers in data

Symbolic Math*

`sym x, syms x y z`	Declare symbolic variable
`eqn = y == 2*a + b`	Define a symbolic equation
`solve(eqns,vars)`	Solve symbolic expression for variable
`subs(expr,var,val)`	Substitute variable in expression
`expand(expr)`	Expand symbolic expression
`factor(expr)`	Factorize symbolic expression
`simplify(expr)`	Simplify symbolic expression
`assume(var,assumption)`	Make assumption for variable
`assumptions(z)`	Show assumptions for symbolic object
`fplot(expr), fcontour, fsurf, fmesh, fimplicit`	Plotting functions for symbolic expressions
`diff(expr,var,n)`	Differentiate symbolic expression
`dsolve(deqn,cond)`	Solve differential equation symbolically
`int(expr,var,[a, b])`	Integrate symbolic expression
`taylor(fun,var,z0)`	Taylor expansion of function

*requires Symbolic Math Toolbox

mathworks.com/help/matlab