PRACTICAL COMPUTING USING MATLAB

The Basics

JESSE GABRIEL

Contents

Chapter 1

Introduction

1.1 Overview

Computing is such an interesting field that many of us are undeniably curious and interested in. Computing comes in many forms, types, systems, tools and platforms, but often, we generally describe it as digital technologies. The ones that we are most familiar with and use almost daily include the internet, the apps or applications and software in our phones and computers, the social media applications including Facebook and WhatsApp, even our mobile phones, our desktop or laptop computers and many others. These are all computing devices that are generally grouped as software applications and hardware devices. A computer hardware is any element of a computer that is physical. This includes things like monitors, keyboards, and also the insides of devices, like microchips and hard drives, as well as our phones and physical calculators. A software is anything that tells the hardware what to do and how to do it, including computer programs and apps on our phones including Facebook and WhatsApp.

In the context of this book, we are interested in the computer software. Think of computers as a human body. In the context of this book, we are simply interested in the computer's brain and nervious system, where the brain is like the central processing unit (CPU) and the software applications are like the nervious system and together they instruct the different parts of the body (hardware) what to do and how to do things. This brings us to the big question: how do you write such a computer software? The purpose of this book is simply to answer that question: not in words, but in practical software writing. As all things start small, this book provides basic software writing techniques, methods and terminologies especially through solved practice exercises. The MATLAB software is used for the purpose of this.

MATLAB is one of the powerful tools out there to write or create computer software or program, or simply computer code. You can use MATLAB to write different types of code, but it specializes in numerical (numbers) computations mainly applied in engineering, science and economics. We won't go into those big calculations, but start with the basics.

You can also use the `hold on` function in MATLAB to create many plots on the same figure. Try the following code in the editor.

```
x = linspace(0, 2*pi, 100);
y_sin = sin(x); y_cos = cos(x); y_2sin = 0.5*sin(2*x);
plot(x, y_sin, 'b', 'LineWidth', 1.2); hold on;
plot(x, y_cos, 'r', 'LineWidth', 1.2);
plot(x, y_2sin, 'g', 'LineWidth', 1.2); hold off;
xlabel('x'); ylabel('f(x)'); title('Trigonometric Functions');
xticks([0,pi/2,pi,3*pi/2,2*pi]);
xticklabels({'0','\pi/2','\pi','3\pi/2','2\pi'}); yticks
    ([-1,-0.5,0,0.5,1]);
legend('\sin(x)', '\cos(x)', '\frac{1}{2}\sin(2x)');
```

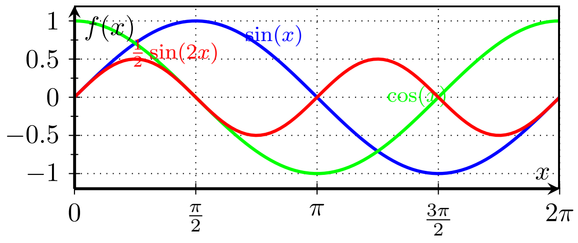


Figure 2.9: Trigonometric functions.

Scatter Plots

Scatter plots are used to visualize the relationship between variables by plotting individual data points on a plane. Each data point is represented by a marker at its corresponding $x - y$ coordinates. To create a 2D scatter plot, you need two vectors of equal length: one for the x-values and one for the corresponding y-values. Here's an example:

```
>> x = [1 2 3 4 5];
>> y = [5 2 7 4 8];
>> scatter(x, y);
>> xlabel('x'); ylabel('y');
>> title('Scatter Plot');
```

In this example, we have two vectors x and y representing the x and y coordinates of the data points. The scatter function is used to generate the scatter plot by plotting the data points on the graph. This generates a plot similar to Figure 2.10: Scatter plots are particularly useful for visualizing patterns, trends, or relationships between variables. They can also be customized by changing the marker shape, size, color, and adding legends to distinguish different groups of data points.

Bar Plots

Bar plots are commonly used to represent categorical data or to compare different quantities across categories. Each category is represented by a bar whose height

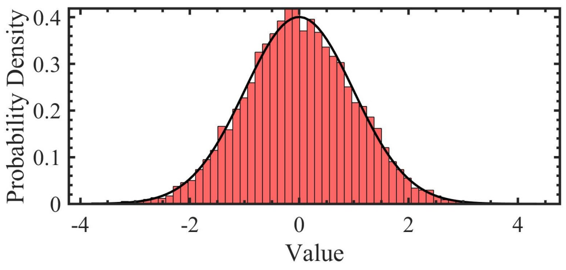
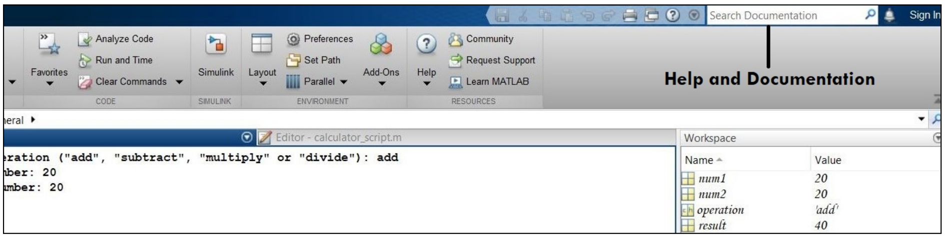


Figure 2.16: Histogram plot of Gaussian (normal) data distribution.

2.8 Further Reading

There are many books and online resources which aim at teaching MATLAB, including the official documentation of MATLAB from MathWorks. The topics covered here are basics and the keen reader is encouraged use these as pointers to explore more detailed online resources. References [20, 21] consist of a complete introduction to programming in MATLAB. This list is not restricted to those references though and among the online books there is a huge variety of books written for teaching MATLAB. Books [22, 24] also provide concrete introduction to MATLAB with specialization in numerical computing in science and engineering. The book [23] teaches data science in MATLAB. In terms of arrays including matrices that we briefly covered, a complete reference on matrices and linear algebra can be found from introductory books to linear algebra written by Gilbert Strang [27, 28].

The MATLAB software package also offers comprehensive help and documentation resources to assist you in learning and using the software effectively. The Help Browser provides access to the MATLAB documentation, which includes detailed explanations, examples, and references for each function and toolbox. You can search for specific topics or browse through the available documentation that can be accessed as shown in Figure 2.17.



```
eration ("add", "subtract", "multiply" or "divide"): add
nber: 20
umber: 20
```

Name	Value
num1	20
num2	20
operation	'add'
result	40

Figure 2.17: Help can be searched in the indicated space. This brings out comprehensive documentation of the search results.

Comment Your Code

Comments are crucial for explaining complex logic, documenting your code's purpose, and providing usage instructions for functions. Use comments to describe the purpose of variables, functions, and major code sections.

```
% Calculate the average temperature
average_temp = mean(temperature_data);
% Define a function to calculate the square of a number
function result = square_number(x)
    % Square the input
    result = x^2;
end
```

Organize Code into Functions

Breaking your code into functions enhances modularity and reusability. Each function should have a single, well-defined purpose. Aim for functions that are relatively short and focus on one task.

```
% Function to load and preprocess temperature data
function temperature_data = load_and_preprocess_data(filename)
    temperature_data = load(filename);
    % Additional preprocessing steps can go here
end
```

```
% Main script
data = load_and_preprocess_data('temperature_data.txt');
plot_data(data);
```

Use Script Files for Sequential Code

Scripts are useful for organizing sequential code that runs from top to bottom. However, avoid placing complex logic directly in script files. Instead, call functions defined in separate function files.

Create Meaningful Subdirectories

Finally, as your MATLAB project grows, organizing your files into meaningful subdirectories becomes essential. This practice helps maintain a clear project structure and makes it easier to locate specific files. Consider the following subdirectory organization:

i) **Data**: Store raw data files or datasets in this directory.
ii) **Functions**: Place custom functions you've written in this directory. Organize them based on functionality or purpose.
iii) **Scripts**: Keep your main scripts or entry point scripts in this directory. These scripts usually call functions defined in the "Functions" directory.
iv) **Tests**: If you're following best practices and writing unit tests for your functions, keep your test scripts and data in this directory.
v) **Documentation**: Store any project-related documentation, such as README files or notes, here.

Note that in the above code, we have used the **structure** data type that we briefly described in Table 2.3, to store the details of the excnange rates. The structure data type in MATLAB allows grouping different data elements together under a single variable. In the above code, the structure named `exchangeRates` is created to store the exchange rates for various currencies. Each currency is associated with a specific exchange rate.

Structure data type

A structure data type in MATLAB is a like a flexible container that allows the grouping of related data using named fields, providing a way to organize and access different types of information within a single variable.

Let's use the Currency Converter with some examples, then see how data is stored in the structure data type. Run the script and provide inputs in the command window as follows.

```
Enter the amount: 100
Enter source currency code (e.g., USD): USD
Enter target currency code (e.g., AUD): AUD
Converted Amount: 136 AUD
```

```
Enter the amount: 1000
Enter source currency code (e.g., USD): PGK
Enter target currency code (e.g., AUD): USD
Converted Amount: 281.6901 USD
```

Our currency converter is working perfectly! Now let's see how the data elements are stored in the structure data type`exchangeRates`. Type `exchangeRates` in the command window and press "ENTER".

```
>> exchangeRates
      exchangeRates =
        struct with fields:
        PGK: 3.5500
        AUD: 1.3600
        USD: 1
        NZD: 1.4800
        CNY: 6.4400
```

`exchangeRates` has five fields with data corresponding to the currency conversion factors. We can access each conversion factor as follows:

```
>> exchangeRates.PGK
        ans =
                3.5500
```

```
>> exchangeRates.CNY
        ans =
                6.4400
```

You can also create structure data type in `exchangeRates` is as follows: