Introduction to Matlab:
Application to Electrical Engineering

Houssem Rafik El Hana Bouchekara

Umm El Qura University

(version 1, February 2011)
Contents

1 CHAPTER 1 ............................................................................................................. 7
  1.1 TUTORIAL LESSONS 1 ..................................................................................... 7
    1.1.1 Introduction ................................................................................................. 7
  1.2 STARTING AND QUITTING MATLAB ............................................................. 8
    1.2.1 Starting MATLAB ...................................................................................... 8
    1.2.2 Quitting MATLAB ..................................................................................... 8
  1.3 MATLAB DESKTOP ......................................................................................... 8
  1.4 DESKTOP TOOLS ............................................................................................ 10
    1.4.1 Command Window .................................................................................... 10
    1.4.2 Command History .................................................................................... 11
    1.4.3 Launch Pad .................................................................................................. 12
    1.4.4 Help Browser .............................................................................................. 12
    1.4.5 Current Directory Browser ......................................................................... 13
    1.4.6 Workspace Browser ................................................................................... 14
    1.4.7 Editor/Debugger .......................................................................................... 16
  1.5 GETTING STARTED ............................................................................................ 17
    1.5.1 Using MATLAB as a calculator ................................................................. 17
    1.5.2 Creating MATLAB variables ................................................................. 18
    1.5.3 Overwriting variable .................................................................................. 19
    1.5.4 Error messages ......................................................................................... 19
    1.5.5 Making corrections .................................................................................... 19
    1.5.6 Controlling the hierarchy of operations or precedence ............................. 19
    1.5.7 Controlling the appearance of floating point number .............................. 21
    1.5.8 Managing the workspace ........................................................................... 21
    1.5.9 Keeping track of your work session ......................................................... 22
    1.5.10 Entering multiple statements per line .................................................... 22
    1.5.11 Miscellaneous commands .................................................................... 22
    1.5.12 Getting help ............................................................................................ 23
  1.6 EXERCISES ........................................................................................................ 23

2 CHAPTER 2 ............................................................................................................. 24
  2.1 MATHEMATICAL FUNCTIONS ........................................................................ 24
2.1.1 Examples ........................................................................................................................................... 25
2.2 Basic Plotting ......................................................................................................................................... 26
2.2.1 Overview ........................................................................................................................................... 26
2.2.2 Creating simple plots ...................................................................................................................... 26
2.2.3 Adding titles, axis labels, and annotations .................................................................................... 27
2.2.4 Multiple data sets in one plot ...................................................................................................... 28
2.2.5 Specifying line styles and colors ................................................................................................. 29
2.2.6 Copy/Paste Figures ....................................................................................................................... 31
2.2.7 Saving Figures ............................................................................................................................... 32
2.3 Exercises ................................................................................................................................................ 32
2.4 Animations ............................................................................................................................................. 33
2.4.1 Erase Mode Method ..................................................................................................................... 33
2.4.2 Creating Movies ............................................................................................................................. 34
2.5 Working with Matrices ........................................................................................................................ 35
2.5.1 Introduction ................................................................................................................................... 35
2.5.2 Matrix generation .......................................................................................................................... 35
2.6 Exercises ................................................................................................................................................ 45

3 Chapter 3: Array Operations and Linear Equations ............................................................................... 46
3.1 Array Operations .................................................................................................................................... 46
3.1.1 Matrix arithmetic operations ....................................................................................................... 46
3.1.2 Array arithmetic operations ......................................................................................................... 46
3.2 Solving Linear Equations ..................................................................................................................... 47
3.2.1 Matrix inverse ............................................................................................................................... 49
3.2.2 Matrix functions ............................................................................................................................ 50
3.3 Exercises ................................................................................................................................................ 50

4 Chapter 4: Introduction to Programming in MATLAB ........................................................................... 51
4.1 Introduction ............................................................................................................................................. 51
4.2 M-File Scripts ...................................................................................................................................... 51
4.2.1 Examples ....................................................................................................................................... 51
4.2.2 Script side-effects .......................................................................................................................... 53
4.3 M-File Functions .................................................................................................................................. 53
4.3.1 Anatomy of a M-File function ..................................................................................................... 53
4.3.2 Input and output arguments .......................................................................................................... 54
Matlab is an interactive system for doing numerical computations. The aim of this book is to help the student to be familiar with Matlab. The emphasis here is "learning by doing".
About the author

Houssem REH Bouchekara is an assistant professor in the electrical engineering department of Umm Al-Qura University. He has received his BS in electrical engineering from University Mentouri Constantine, Algeria, in 2004. He received his Master’s in Electronic Systems and Electrical Engineering from Polytechnic School of the University of Nantes, France, 2005. He received his Ph.D. in Electrical Engineering from Grenoble Electrical Engineering Laboratory, France, in 2008. His research interest includes Electric machines, Magnetic refrigeration, and Power system.
1 Chapter 1

1.1 Tutorial lessons 1

1.1.1 Introduction

The primarily objective is to help you learn quickly the first steps. The emphasis here is “learning by doing”. Therefore, the best way to learn is by trying it yourself. Working through the examples will give you a feel for the way that MATLAB operates. In this introduction we will describe how MATLAB handles simple numerical expressions and mathematical formulas.

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects.

The basic building block in MATLAB is the matrix. The fundamental data type is the array. Vectors, scalars, real and complex matrices are all automatically handled as special cases of basic arrays. The built-in functions are optimized for vector operations. Thus, vectorized commands or codes run much faster in MATLAB (vectorization is a way of computing in which an operation is performed simultaneously on a list of numbers rather than sequentially on each member of the list).

A nice thing to realize is that MATLAB is primarily a numerical computation package, although with the 'Symbolic' Toolbox it can do also symbolic algebra. Mathematica, Maple, and Macsyma are primarily symbolic algebra packages. MATLAB’s ease of use is its best feature since you can have more learning with less effort, while the computer algebra systems have a steeper learning curve.

In mathematical computations, especially those that utilize vectors and matrices, MATLAB is better in terms of ease of use, availability of built-in functions, ease of programming, and speed. MATLAB’s popularity today has forced such packages as Macsyma and Mathematica to provide extensions for files in MATLAB’s format.

There are numerous prepared commands for 2D and 3D graphics as well as for animation. The user is not limited to the built-in functions; he can write his own functions in MATLAB language. Once written, these functions work just like the internal functions. MATLAB’s language is designed to be easy to learn and use.

The many built-in functions provide excellent tools for linear algebra, signal processing, data analysis, optimization, solution of ordinary differential equations (ODEs), and many other types of scientific operations.

There are also several optional 'toolboxes' available which are collections of functions written for special applications such as 'Image Processing', 'Statistics', 'Neural Networks', etc.

The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide.
1.2 Starting and Quitting MATLAB

1.2.1 Starting MATLAB

On a Microsoft Windows platform, to start MATLAB, double-click the MATLAB shortcut icon on your Windows desktop.

On Linux, to start MATLAB, type `matlab` at the operating system prompt.

After starting MATLAB, the MATLAB desktop opens – see “MATLAB Desktop”.

You can change the directory in which MATLAB starts, define startup options including running a script upon startup, and reduce startup time in some situations.

1.2.2 Quitting MATLAB

To end your MATLAB session, select Exit MATLAB from the File menu in the desktop, or type `quit` in the Command Window. To execute specified functions each time MATLAB quits, such as saving the workspace, you can create and run a `finish.m` script.

1.3 MATLAB Desktop

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

The first time MATLAB starts, the desktop appears as shown in the following illustration, although your Launch Pad may contain different entries.
You can change the way your desktop looks by opening, closing, moving, and resizing the tools in it. You can also move tools outside of the desktop or return them back inside the desktop (docking). All the desktop tools provide common features such as context menus and keyboard shortcuts.

You can specify certain characteristics for the desktop tools by selecting Preferences from the File menu. For example, you can specify the font characteristics for Command Window text. For more information, click the Help button in the Preferences dialog box as shown in Figure 2.
This section provides an introduction to MATLAB’s desktop tools. You can also use MATLAB functions to perform most of the features found in the desktop tools. The tools are:

- Command Window.
- Command History.
- Launch Pad
- Help Browser.
- Current Directory Browser.
- Workspace Browser.
- Array Editor.
- Editor/Debugger.

### 1.4.1 Command Window

Use the Command Window to enter variables and run functions and M-files.
1.4.2 Command History

Lines you enter in the Command Window are logged in the Command History window. In the Command History, you can view previously used functions, and copy and execute selected lines.

To save the input and output from a MATLAB session to a file, use the \texttt{diary} function.

Figure 3: Command Window.

Figure 4: Command History.
Running External Programs

You can run external programs from the MATLAB Command Window. The exclamation point character `!` is a shell escape and indicates that the rest of the input line is a command to the operating system. This is useful for invoking external utilities or running other programs without quitting MATLAB. On Linux, for example,

```
!emacs magik.m
```

invokes an editor called emacs for a file named magik.m. When you quit the external program, the operating system returns control to MATLAB.

### 1.4.3 Launch Pad

MATLAB’s Launch Pad provides easy access to tools, demos, and documentation.

![Sample of listings in Launch Pad – you’ll see listings for all products installed on your system.](image)

**Figure 5: Launch Pad.**

### 1.4.4 Help Browser

Use the Help browser to search and view documentation for all MathWorks products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents.

To open the Help browser, click the help button in the toolbar, or type `helpbrowser` in the Command Window.
1.4.5 Current Directory Browser

MATLAB file operations use the current directory and the search path as reference points. Any file you want to run must either be in the current directory or on the search path.

A quick way to view or change the current directory is by using the Current Directory field in the desktop toolbar as shown below.

![Current Directory Field](image)

To search for, view, open, and make changes to MATLAB-related directories and files, use the MATLAB Current Directory browser. Alternatively, you can use the functions dir, cd, and delete.
Search Path

To determine how to execute functions you call, MATLAB uses a *search path* to find M-files and other MATLAB-related files, which are organized in directories on your file system. Any file you want to run in MATLAB must reside in the current directory or in a directory that is on the search path. By default, the files supplied with MATLAB and MathWorks toolboxes are included in the search path.

To see which directories are on the search path or to change the search path, select Set Path from the File menu in the desktop, and use the Set Path dialog box. Alternatively, you can use the path function to view the search path, addpath to add directories to the path, and rmpath to remove directories from the path.

1.4.6 Workspace Browser

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces.

To view the workspace and information about each variable, use the Workspace browser, or use the functions who and whos.

To delete variables from the workspace, select the variable and select **Delete** from the *Edit* menu. Alternatively, use the clear function.
The workspace is not maintained after you end the MATLAB session. To save the workspace to a file that can be read during a later MATLAB session, select **Save Workspace As** from the **File** menu, or use the `save` function. This saves the workspace to a binary file called a MAT-file, which has a `.mat` extension.

There are options for saving to different formats. To read in a MAT-file, select **Import Data** from the **File** menu, or use the `load` function.

**Array Editor**

Double-click on a variable in the Workspace browser to see it in the Array Editor. Use the Array Editor to view and edit a visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace.
1.4.7 Editor/Debugger

Use the Editor/Debugger to create and debug M-files, which are programs you write to run MATLAB functions. The Editor/Debugger provides a graphical user interface for basic text editing, as well as for M-file debugging.

![Figure 11: Editor/Debugger.](image)

You can use any text editor to create M-files, such as Emacs, and can use preferences (accessible from the desktop File menu) to specify that editor as the default. If you use another editor, you can still use the MATLAB Editor/Debugger for debugging, or you can use debugging functions, such as dbstop, which sets a breakpoint.

If you just need to view the contents of an M-file, you can display it in the Command Window by using the type function.
1.5 Getting started

Now, we are interested in doing some simple calculations. We will assume that you have sufficient understanding of your computer under which MATLAB is being run.

You are now faced with the MATLAB desktop on your computer, which contains the prompt (>>) in the Command Window. Usually, there are 2 types of prompt:

- >> for full version
- EDU> for educational version

**Note:** To simplify the notation, we will use this prompt, >>, as a standard prompt sign, though our MATLAB version is for educational purpose.

1.5.1 Using MATLAB as a calculator

As an example of a simple interactive calculation, just type the expression you want to evaluate. Let’s start at the very beginning. For example, let’s suppose you want to calculate the expression, \(1 + 2 \times 3\). You type it at the prompt command (>>) as follows,

\[
\text{>> } 1+2*3
\]

\[
\text{ans} = 7
\]

You will have noticed that if you do not specify an output variable, MATLAB uses a default variable ‘ans’, short for answer, to store the results of the current calculation. Note that the variable ‘ans’ is created (or overwritten, if it is already existed). To avoid this, you may assign a value to a variable or output argument name. For example,

\[
\text{>> } x = 1+2*3
\]

\[
x = 7
\]

will result in \(x\) being given the value \(1 + 2 \times 3 = 7\). This variable name can always be used to refer to the results of the previous computations. Therefore, computing \(4x\) will result in

\[
\text{>> } 4*x
\]

\[
\text{ans} = 28.0000
\]

Before we conclude this minimum session, Table 1.1 gives the partial list of commonly used MATLAB operators and special characters used to solve many engineering and science problems.
After learning the minimum MATLAB session, we will now learn to use some additional operations.

### 1.5.2 Creating MATLAB variables

MATLAB variables are created with an assignment statement. The syntax of variable assignment is variable name = a value (or an expression)

For example,

```matlab
>> x = expression
```

where expression is a combination of numerical values, mathematical operators, variables, and function calls. On other words, expression can involve:

- manual entry
- built-in functions
- user-defined functions
1.5.3 Overwriting variable

Once a variable has been created, it can be reassigned. In addition, if you do not wish to see the intermediate results, you can suppress the numerical output by putting a semicolon (;) at the end of the line. Then the sequence of commands looks like this:

```matlab
>> t = 5;
>> t = t+1
  t =
  6
```

1.5.4 Error messages

If we enter an expression incorrectly, MATLAB will return an error message. For example, in the following, we left out the multiplication sign, *, in the following expression

```matlab
>> x = 10;
>> 5x
??? 5x
| Error: Unexpected MATLAB expression.
```

1.5.5 Making corrections

To make corrections, we can, of course retype the expressions. But if the expression is lengthy, we make more mistakes by typing a second time. A previously typed command can be recalled with the up-arrow key ↑. When the command is displayed at the command prompt, it can be modified if needed and executed.

1.5.6 Controlling the hierarchy of operations or precedence

Let’s consider the previous arithmetic operation, but now we will include parentheses. For example, \(1 + 2 \times 3\) will become \((1 + 2) \times 3\)

```matlab
>> (1+2)*3
  ans =
   9
```

and, from previous example
By adding parentheses, these two expressions give different results: 9 and 7.

The order in which MATLAB performs arithmetic operations is exactly that taught in high school algebra courses. *Exponentiations* are done first, followed by *multiplications and divisions*, and finally by *additions and subtractions*. However, the standard order of precedence of arithmetic operations can be changed by inserting *parentheses*. For example, the result of \(1 + 2 \times 3\) is quite different than the similar expression with parentheses \((1 + 2) \times 3\). The results are 7 and 9 respectively. Parentheses can always be used to overrule *priority*, and their use is recommended in some complex expressions to avoid ambiguity.

Therefore, to make the evaluation of expressions unambiguous, MATLAB has established a series of rules. The order in which the arithmetic operations are evaluated is given in Table 2. MATLAB arithmetic operators obey the same *precedence* rules as those in most computer programs. For operators of equal precedence, evaluation is from *left* to *right*. Now, consider another example:

\[
\frac{1}{2 + 3^2} + \frac{4}{5} \times \frac{6}{7}
\]

In MATLAB, it becomes

```
>> 1/(2+3^2)+4/5*6/7
ans =
0.7766
```

or, if parentheses are missing,

```
>> 1/2+3^2+4/5*6/7
ans =
10.1857
```
So here what we get: two different results. Therefore, we want to emphasize the importance of precedence rule in order to avoid ambiguity.

1.5.7 Controlling the appearance of floating point number

MATLAB by default displays only 4 decimals in the result of the calculations, for example $-163.6667$, as shown in above examples. However, MATLAB does numerical calculations in double precision, which is 15 digits. The command format controls how the results of computations are displayed. Here are some examples of the different formats together with the resulting outputs.

```matlab
>> format short
>> x=-163.6667

If we want to see all 15 digits, we use the command format long

```matlab
>> format long
>> x=-1.636666666666667e+002

To return to the standard format, enter format short, or simply format.

There are several other formats. For more details, see the MATLAB documentation, or type help format.

Note - Up to now, we have let MATLAB repeat everything that we enter at the prompt (>>). Sometimes this is not quite useful, in particular when the output is pages en length. To prevent MATLAB from echoing what we type, simply enter a semicolon (;) at the end of the command. For example,

```matlab
>> x=-163.6667;

and then ask about the value of x by typing,

```matlab
>> x
x =
-163.6667

1.5.8 Managing the workspace

The contents of the workspace persist between the executions of separate commands. Therefore, it is possible for the results of one problem to have an effect on the next one. To avoid this possibility, it is a good idea to issue a clear command at the start of each new independent calculation.

```matlab
>> clear

The command clear or clear all removes all variables from the workspace. This frees up system memory. In order to display a list of the variables currently in the memory, type
>> who

while, who will give more details which include size, space allocation, and class of the variables.

1.5.9 Keeping track of your work session

It is possible to keep track of everything done during a MATLAB session with the diary command.

>> diary

or give a name to a created file,

>> diary FileName

where FileName could be any arbitrary name you choose.

The function diary is useful if you want to save a complete MATLAB session. They save all input and output as they appear in the MATLAB window. When you want to stop the recording, enter diary off. If you want to start recording again, enter diary on. The file that is created is a simple text file. It can be opened by an editor or a word processing program and edited to remove extraneous material, or to add your comments. You can use the function type to view the diary file or you can edit in a text editor or print. This command is useful, for example in the process of preparing a homework or lab submission.

1.5.10 Entering multiple statements per line

It is possible to enter multiple statements per line. Use commas (,) or semicolons (;) to enter more than one statement at once. Commas (,) allow multiple statements per line without suppressing output.

>> a=7; b=cos(a), c=cosh(a)

b =
0.6570

c =
548.3170

1.5.11 Miscellaneous commands

Here are few additional useful commands:

- To clear the Command Window, type clc
- To abort a MATLAB computation, type ctrl-c
- To continue a line, type . . .
1.5.12 Getting help

To view the online documentation, select MATLAB Help from Help menu or MATLAB Help directly in the Command Window. The preferred method is to use the Help Browser. The Help Browser can be started by selecting the ? icon from the desktop toolbar. On the other hand, information about any command is available by typing

```
>> help Command
```

Another way to get help is to use the lookfor command. The lookfor command differs from the help command. The help command searches for an exact function name match, while the lookfor command searches the quick summary information in each function for a match. For example, suppose that we were looking for a function to take the inverse of a matrix. Since MATLAB does not have a function named inverse, the command help inverse will produce nothing. On the other hand, the command lookfor inverse will produce detailed information, which includes the function of interest, inv.

```
>> lookfor inverse
```

Note - At this particular time of our study, it is important to emphasize one main point. Because MATLAB is a huge program; it is impossible to cover all the details of each function one by one. However, we will give you information how to get help. Here are some examples:

- Use on-line help to request info on a specific function

```
>> help sqrt
```

- In the current version (MATLAB version 7), the doc function opens the on-line version of the help manual. This is very helpful for more complex commands.

```
>> doc plot
```

- Use lookfor to find functions by keywords. The general form is

```
>> lookfor FunctionName
```

1.6 Exercises
Chapter 2

2.1 Mathematical functions

MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.

Typing `help elfun` and `help specfun` calls up full lists of elementary and special functions respectively.

There is a long list of mathematical functions that are built into MATLAB. These functions are called built-ins. Many standard mathematical functions, such as \( \sin(x) \), \( \cos(x) \), \( \tan(x) \), \( e^x \), \( \ln(x) \), are evaluated by the functions `sin`, `cos`, `tan`, `exp`, and `log` respectively in MATLAB.

Table 3 lists some commonly used functions, where variables \( x \) and \( y \) can be numbers, vectors, or matrices.

Table 3: Elementary functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs</code></td>
<td>Absolute value and complex magnitude</td>
</tr>
<tr>
<td><code>acos</code>, <code>acosh</code></td>
<td>Inverse cosine and inverse hyperbolic cosine</td>
</tr>
<tr>
<td><code>acot</code>, <code>acoth</code></td>
<td>Inverse cotangent and inverse hyperbolic cotangent</td>
</tr>
<tr>
<td><code>acsc</code>, <code>acsch</code></td>
<td>Inverse cosecant and inverse hyperbolic cosecant</td>
</tr>
<tr>
<td><code>angle</code></td>
<td>Phase angle</td>
</tr>
<tr>
<td><code>asec</code>, <code>asech</code></td>
<td>Inverse secant and inverse hyperbolic secant</td>
</tr>
<tr>
<td><code>asin</code>, <code>asinh</code></td>
<td>Inverse sine and inverse hyperbolic sine</td>
</tr>
<tr>
<td><code>atan</code>, <code>atanh</code></td>
<td>Inverse tangent and inverse hyperbolic tangent</td>
</tr>
<tr>
<td><code>atan2</code></td>
<td>Four-quadrant inverse tangent</td>
</tr>
<tr>
<td><code>ceil</code></td>
<td>Round toward infinity</td>
</tr>
<tr>
<td><code>complex</code></td>
<td>Construct complex data from real and imaginary components</td>
</tr>
<tr>
<td><code>conj</code></td>
<td>Complex conjugate</td>
</tr>
<tr>
<td><code>cos</code>, <code>cosh</code></td>
<td>Cosine and hyperbolic cosine</td>
</tr>
<tr>
<td><code>cot</code>, <code>coth</code></td>
<td>Cotangent and hyperbolic cotangent</td>
</tr>
<tr>
<td><code>csc</code>, <code>csch</code></td>
<td>Cosecant and hyperbolic cosecant</td>
</tr>
<tr>
<td><code>exp</code></td>
<td>Exponential function</td>
</tr>
<tr>
<td><code>fix</code></td>
<td>Round toward zero</td>
</tr>
<tr>
<td><code>floor</code></td>
<td>Round toward minus infinity</td>
</tr>
<tr>
<td><code>gcd</code></td>
<td>Greatest common divisor</td>
</tr>
<tr>
<td><code>imag</code></td>
<td>Imaginary part of a complex number</td>
</tr>
<tr>
<td><code>lcm</code></td>
<td>Least common multiple</td>
</tr>
<tr>
<td><code>log</code></td>
<td>Natural logarithm</td>
</tr>
<tr>
<td><code>log2</code></td>
<td>Base 2 logarithm and dissect floating-point numbers into exponent and mantissa</td>
</tr>
<tr>
<td><code>log10</code></td>
<td>Common (base 10) logarithm</td>
</tr>
<tr>
<td><code>mod</code></td>
<td>Modulus (signed remainder after division)</td>
</tr>
<tr>
<td><code>nchoosek</code></td>
<td>Binomial coefficient or all combinations</td>
</tr>
<tr>
<td><code>real</code></td>
<td>Real part of complex number</td>
</tr>
<tr>
<td><code>rem</code></td>
<td>Remainder after division</td>
</tr>
<tr>
<td><code>round</code></td>
<td>Round to nearest integer</td>
</tr>
<tr>
<td><code>sec</code>, <code>sech</code></td>
<td>Secant and hyperbolic secant</td>
</tr>
<tr>
<td><code>sign</code></td>
<td>Signum function</td>
</tr>
<tr>
<td><code>sin</code>, <code>sinh</code></td>
<td>Sine and hyperbolic sine</td>
</tr>
<tr>
<td><code>sqrt</code></td>
<td>Square root</td>
</tr>
<tr>
<td><code>tan</code>, <code>tanh</code></td>
<td>Tangent and hyperbolic tangent</td>
</tr>
</tbody>
</table>
In addition to the elementary functions, MATLAB includes a number of predefined constant values. A list of the most common values is given in Table 4.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>3.14159265…</td>
</tr>
<tr>
<td>i</td>
<td>Imaginary unit, $\sqrt{-1}$</td>
</tr>
<tr>
<td>j</td>
<td>Same as i</td>
</tr>
<tr>
<td>eps</td>
<td>Floating-point relative precision, $2^{-52}$</td>
</tr>
<tr>
<td>realmin</td>
<td>Smallest floating-point number, $2^{-1022}$</td>
</tr>
<tr>
<td>realmax</td>
<td>Largest floating-point number, $(2 - \varepsilon)2^{1023}$</td>
</tr>
<tr>
<td>Inf</td>
<td>Infinity</td>
</tr>
<tr>
<td>NaN</td>
<td>Not-a-number</td>
</tr>
</tbody>
</table>

2.1.1 Examples

We illustrate here some typical examples which relate to the elementary functions previously defined.

As a first example, the value of the expression $y = e^{-a} \sin(x) + 10\sqrt{y}$, for $a = 5$, $x = 2$, and $y = 8$ is computed by

```matlab
>> a = 5; x = 2; y = 8;
>> y = exp(-a)*sin(x)+10*sqrt(y)
y =
28.2904
```

The subsequent examples are

```matlab
>> log(142)
ans =
4.9558
>> log10(142)
ans =
2.1523
```

Note the difference between the natural logarithm $\log(x)$ and the decimal logarithm (base 10) $\log_{10}(x)$. 

25
To calculate $\sin(\pi/4)$ and $e^{10}$, we enter the following commands in MATLAB,

```matlab
>> sin(pi/4)
ans =
0.7071
>> exp(10)
ans =
2.2026e+004
```

Notes:

- Only use built-in functions on the right hand side of an expression. Reassigning the value to a built-in function can create problems.
- There are some exceptions. For example, $i$ and $j$ are pre-assigned to $\sqrt{-1}$. However one or both of $i$ or $j$ are often used as loop indices.
- To avoid any possible confusion, it is suggested to use instead $ii$ or $jj$ as loop indices.

### 2.2 Basic plotting

#### 2.2.1 Overview

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible. Trying to understand mathematical equations with graphics is an enjoyable and very efficient way of learning mathematics. Being able to plot mathematical functions and data freely is the most important step, and this section is written to assist you to do just that.

#### 2.2.2 Creating simple plots

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of $x$-coordinates, $x = (x_1, \ldots, x_N)$, and a vector of $y$-coordinates, $y = (y_1, \ldots, y_N)$, locate the points $(x_i, y_i)$, with $i = 1, 2, \ldots, n$ and then join them by straight lines. You need to prepare $x$ and $y$ in an identical array form; namely, $x$ and $y$ are both row arrays or column arrays of the same length.

The MATLAB command to plot a graph is `plot(x,y).` The vectors $x = (1, 2, 3, 4, 5, 6)$ and $y = (3, -1, 2, 4, 5, 1)$ produce the picture shown in Figure 2.1.

```matlab
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```
Note: The plot functions has different forms depending on the input arguments. If \( y \) is a vector \( \text{plot}(y) \) produces a piecewise linear graph of the elements of \( y \) versus the index of the elements of \( y \). If we specify two vectors, as mentioned above, \( \text{plot}(x,y) \) produces a graph of \( y \) versus \( x \).

For example, to plot the function \( \sin(x) \) on the interval \([0, 2\pi]\), we first create a vector of \( x \) values ranging from 0 to \( 2\pi \), then compute the sine of these values, and finally plot the result:

\[
\begin{align*}
\text{Figure 12: Plot for the vectors } x \text{ and } y \\
\end{align*}
\]

\[
\begin{align*}
 >> x = 0:pi/100:2*pi; \\
 >> y = \sin(x); \\
 >> \text{plot}(x,y)
\end{align*}
\]

Notes:

- \( 0:pi/100:2*pi \) yields a vector that
  - starts at 0,
  - takes steps (or increments) of \( \pi/100 \),
  - stops when \( 2\pi \) is reached.
- If you omit the increment, MATLAB automatically increments by 1.

2.2.3 Adding titles, axis labels, and annotations

MATLAB enables you to add axis labels and titles. For example, using the graph from the previous example, add an \( x \)- and \( y \)-axis labels.

Now label the axes and add a title. The character \( \backslash \pi \) creates the symbol \( \pi \). An example of 2D plot is shown in Figure 2.2.
Figure 2.2: Plot of the Sine function

```matlab
>> xlabel('x = 0:2\pi')
>> ylabel('Sine of x')
>> title('Plot of the Sine function')
```

The color of a single curve is, by default, blue, but other colors are possible. The desired color is indicated by a third argument. For example, red is selected by `plot(x,y,'r')`. Note the single quotes, ’’, around r.

### 2.2.4 Multiple data sets in one plot

Multiple \((x, y)\) pairs arguments create *multiple* graphs with a single call to `plot`. For example, these statements plot three related functions of \(x\): \(y_1 = 2\cos(x)\), \(y_2 = \cos(x)\), and \(y_3 = 0.5\times\cos(x)\), in the interval \(0 \leq x \leq 2\pi\).

```matlab
>> x = 0:pi/100:2*pi;
>> y1 = 2*cos(x);
>> y2 = cos(x);
>> y3 = 0.5*cos(x);
>> plot(x,y1,'--',x,y2,'-',x,y3,':')
>> xlabel('0 \leq x \leq 2\pi')
>> ylabel('Cosine functions')
```
The result of multiple data sets in one graph plot is shown in Figure 13.

![Typical example of multiple plots](image)

**Figure 13: Typical example of multiple plots.**

By default, MATLAB uses line style and color to distinguish the data sets plotted in the graph. However, you can change the appearance of these graphic components or add annotations to the graph to help explain your data for presentation.

### 2.2.5 Specifying line styles and colors

It is possible to specify line styles, colors, and markers (e.g., circles, plus signs, ... ) using the plot command: `plot(x,y,'style_color_marker')`, where style_color_marker is a triplet of values from Table 5.

To find additional information, type help plot or doc plot.

<table>
<thead>
<tr>
<th>Symbol Color</th>
<th>Symbol Line Style</th>
<th>Symbol Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Specifying the Color and Size of Markers

You can also specify other line characteristics using graphics properties (see line for a description of these properties):

- **LineWidth** — Specifies the width (in points) of the line.
- **MarkerEdgeColor** — Specifies the color of the marker or the edge color for filled markers (circle, square, diamond, pentagram, hexagram, and the four triangles).
- **MarkerFaceColor** — Specifies the color of the face of filled markers.
- **MarkerSize** — Specifies the size of the marker in units of points.

For example, these statements produce the graph of

```matlab
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,...
     'MarkerEdgeColor','k',...
     'MarkerFaceColor','g',...
     'MarkerSize',10)
```
2.2.6 Copy/Paste Figures

Figures can be pasted into other apps (word, ppt, etc)

*Edit*→*copy options*→*figure copy template*→Change font sizes, line properties; presets for word and ppt.

*Edit*→*copy figure* to copy figure.

Paste into document of interest.
2.2.7 Saving Figures

Figures can be saved in many formats. The common ones are given the following figure.

![Image of Save As dialog box](image.png)

- **.fig** preserves all information
- **.bmp** uncompressed image
- **.eps** high-quality scalable format
- **.pdf** compressed image

Courtesy of The MathWorks, Inc. Used with permission.

Figure 16: Saving figure.

2.3 Exercises
2.4 Animations

MATLAB provides two ways of generating moving, animated graphics:

- Continually erase and then redraw the objects on the screen, making incremental changes with each redraw.
- Save a number of different pictures and then play them back as a movie.

2.4.1 Erase Mode Method

Using the EraseMode property is appropriate for long sequences of simple plots where the change from frame to frame is minimal. Here is an example showing simulated Brownian motion. Specify a number of points, such as

\[ n = 20 \]

and a temperature or velocity, such as

\[ s = .02 \]

The best values for these two parameters depend upon the speed of your particular computer. Generate \( n \) random points with \((x,y)\) coordinates between \(-1/2\) and \(1/2\).

\[
\begin{align*}
x &= \text{rand}(n,1) - 0.5; \\
y &= \text{rand}(n,1) - 0.5;
\end{align*}
\]

Plot the points in a square with sides at -1 and +1. Save the handle for the vector of points and set its EraseMode to xor. This tells the MATLAB graphics system not to redraw the entire plot when the coordinates of one point are changed, but to restore the background color in the vicinity of the point using an “exclusive or” operation.

\[
\begin{align*}
h &= \text{plot}(x,y,'.'); \\
\text{axis([-1 1 -1 1])} \\
\text{axis square} \\
\text{grid off} \\
\text{set(h,'EraseMode','xor','MarkerSize',18)}
\end{align*}
\]

Now begin the animation. Here is an infinite while loop, which you can eventually exit by typing `Ctrl+c`. Each time through the loop, add a small amount of normally distributed random noise to the coordinates of the points.

Then, instead of creating an entirely new plot, simply change the XData and YData properties of the original plot.

\[
\begin{align*}
\text{while 1}
\end{align*}
\]
How long does it take for one of the points to get outside of the square? How long before all of the points are outside the square?

2.4.2 Creating Movies

If you increase the number of points in the Brownian motion example to something like \( n = 300 \) and \( s = .02 \), the motion is no longer very fluid; it takes too much time to draw each time step. It becomes more effective to save a predetermined number of frames as bitmaps and to play them back as a movie.

First, decide on the number of frames, say \( n_{\text{frames}} = 50 \);

Next, set up the first plot as before, except using the default EraseMode (normal).

```matlab
drawnow
x = x + s*randn(n,1);
y = y + s*randn(n,1);
set(h,'XData',x,'YData',y)
end
```

Figure 17: Animation.
```
axis([-1 1 -1 1])
axis square
grid off

Generate the movie and use getframe to capture each frame.

for k = 1:nframes
    x = x + s*randn(n,1);
    y = y + s*randn(n,1);
    set(h,'XData',x,'YData',y)
    M(k) = getframe;
end

Finally, play the movie 30 times.
movie(M,30)
```

### 2.5 Working with Matrices

#### 2.5.1 Introduction

Matrices are the basic elements of the MATLAB environment. A matrix is a two-dimensional array consisting of \( m \) rows and \( n \) columns. Special cases are column vectors \( (n = 1) \) and row vectors \( (m = 1) \).

In this section we will illustrate how to apply different operations on matrices. The following topics are discussed: vectors and matrices in MATLAB, the inverse of a matrix, determinants, and matrix manipulation.

MATLAB supports two types of operations, known as matrix operations and array operations. Matrix operations will be discussed first.

#### 2.5.2 Matrix generation

Matrices are fundamental to MATLAB. Therefore, we need to become familiar with matrix generation and manipulation. Matrices can be generated in several ways.

##### 2.5.2.1 Entering a vector

A vector is a special case of a matrix. The purpose of this section is to show how to create vectors and matrices in MATLAB. As discussed earlier, an array of dimension \( 1 \times n \) is called a row vector, whereas an array of dimension \( m \times 1 \) is called a column vector. The elements of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by commas. For example, to enter a row vector, \( v \), type
Column vectors are created in a similar way, however, semicolon (;) must separate the components of a column vector,

$$
>> v = [1 \ 4 \ 7 \ 10 \ 13] \\
v = \\
1 \ 4 \ 7 \ 10 \ 13
$$

On the other hand, a row vector is converted to a column vector using the transpose operator. The transpose operation is denoted by an apostrophe or a single quote (’).

$$
>> w = [1;4;7;10;13] \\
w = \\
1 \\
4 \\
7 \\
10 \\
13
$$

Thus, $v(1)$ is the first element of vector $v$, $v(2)$ its second element, and so forth. Furthermore, to access blocks of elements, we use MATLAB's colon notation (:) . For example, to access the first three elements of $v$, we write,

$$
>> v(1:3) \\
ans = \\
1 \ 4 \ 7
$$

Or, all elements from the third through the last elements,
>> v(3,end)
ans =
  7   10  13

where end signifies the last element in the vector. If v is a vector, writing

>> v(:)

produces a column vector, whereas writing

>> v(1:end)

produces a row vector.

2.5.2.2 Entering a matrix

A matrix is an array of numbers. To type a matrix into MATLAB you must

• begin with a square bracket, [
• separate elements in a row with spaces or commas (,)
• use a semicolon (;) to separate rows
• end the matrix with another square bracket, ].

Here is a typical example. To enter a matrix A, such as,

\[ A = \begin{bmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
  7 & 8 & 9 \\
\end{bmatrix} \]

type,

>> A = [1 2 3; 4 5 6; 7 8 9]

MATLAB then displays the 3 × 3 matrix as follows,

\[
A = 
\begin{bmatrix}
  1 & 2 & 3 \\
  4 & 5 & 6 \\
  7 & 8 & 9 \\
\end{bmatrix}
\]

Note that the use of semicolons (;) here is different from their use mentioned earlier to suppress output or to write multiple commands in a single line.
Once we have entered the matrix, it is automatically stored and remembered in the *Workspace*. We can refer to it simply as matrix A. We can then view a particular element in a matrix by specifying its location. We write,

```latex
>> A(2,1)
ans = 4
```

A(2,1) is an element located in the second row and first column. Its value is 4.

### 2.5.2.3 Matrix indexing

We select elements in a matrix just as we did for vectors, but now we need two indices. The element of row $i$ and column $j$ of the matrix A is denoted by $A(i,j)$. Thus, $A(i,j)$ in MATLAB refers to the element $A_{ij}$ of matrix A. The *first* index is the *row* number and the *second* index is the *column* number. For example, $A(1,3)$ is an element of first row and third column. Here, $A(1,3)=3$.

Correcting any entry is easy through indexing. Here we substitute $A(3,3)=9$ by $A(3,3)=0$. The result is

```latex
>> A(3,3) = 0
```

Single elements of a matrix are accessed as $A(i,j)$, where $i \geq 1$ and $j \geq 1$. Zero or negative subscripts are not supported in MATLAB.

### 2.5.2.4 Colon operator

The colon operator will prove very useful and understanding how it works is the key to efficient and convenient usage of MATLAB. It occurs in several different forms.

Often we must deal with matrices or vectors that are too large to enter one element at a time. For example, suppose we want to enter a vector $x$ consisting of points $(0, 0.1, 0.2, 0.3, \cdot \cdot \cdot, 5)$. We can use the command

```latex
>> x = 0:0.1:5;
```

The row vector has 51 elements.

### 2.5.2.5 Linear spacing

On the other hand, there is a command to generate linearly spaced vectors: `linspace`. It is similar to the colon operator `:`, but gives direct control over the number of points. For example,

```latex
y = linspace(a,b)
```

generates a row vector $y$ of 100 points linearly spaced between and including $a$ and $b$. 

38
\[
y = \text{linspace}(a,b,n)
\]

generates a row vector \( y \) of \( n \) points linearly spaced between and including \( a \) and \( b \). This is useful when we want to divide an interval into a number of subintervals of the same length. For example,

\[
\text{>> theta = linspace(0,2*pi,101)}
\]

divides the interval \([0, 2\pi]\) into 100 equal subintervals, then creating a vector of 101 elements.

### 2.5.2.6 Colon operator in a matrix

The colon operator can also be used to pick out a certain row or column. For example, the statement \( A(m:n,k:l) \) specifies rows \( m \) to \( n \) and column \( k \) to \( l \). Subscript expressions refer to portions of a matrix. For example,

\[
\text{>> A(2,:)}
\]

\[
\begin{array}{ccc}
4 & 5 & 6 \\
\end{array}
\]

is the second row elements of \( A \).

The colon operator can also be used to extract a sub-matrix from a matrix \( A \).

\[
\text{>> A(:,2:3)}
\]

\[
\begin{array}{cc}
2 & 3 \\
5 & 6 \\
8 & 0 \\
\end{array}
\]

\( A(:,2:3) \) is a sub-matrix with the last two columns of \( A \).

A row or a column of a matrix can be deleted by setting it to a null vector, \([\ ]\).

\[
\text{>> A(:,2)=''}
\]

\[
\begin{array}{c}
1 & 3 \\
4 & 6 \\
7 & 0 \\
\end{array}
\]
2.5.2.7 Creating a sub-matrix

To extract a submatrix $B$ consisting of rows 2 and 3 and columns 1 and 2 of the matrix $A$, do the following

```matlab
>> B = A([2 3],[1 2])
B =
    4   5
    7   8
```

To interchange rows 1 and 2 of $A$, use the vector of row indices together with the colon operator.

```matlab
>> C = A([2 1 3],:)
C =
    4   5   6
    1   2   3
    7   8   0
```

It is important to note that the colon operator (:) stands for all columns or all rows. To create a vector version of matrix $A$, do the following

```matlab
>> A(:)
ans =
    1
    2
    3
    4
    5
    6
    7
    8
    0
```

The submatrix comprising the intersection of rows $p$ to $q$ and columns $r$ to $s$ is denoted by $A(p:q,r:s)$. 
As a special case, a colon (:) as the row or column specifier covers all entries in that row or column; thus

- \( A(:,j) \) is the \( j \)th column of \( A \), while
- \( A(i,:) \) is the \( i \)th row, and
- \( A(\text{end},:) \) picks out the last row of \( A \).

The keyword \texttt{end}, used in \( A(\text{end},:) \), denotes the last index in the specified dimension. Here are some examples.

```matlab
>> A
A =
    1    2    3
    4    5    6
    7    8    9
```

```matlab
>> A(2:3,2:3)
ans =
    5    6
    8    9
```

```matlab
>> A(end:-1:1,end)
ans =
    9
    6
    3
```

```matlab
>> A([1 3],[2 3])
ans =
    2    3
    8    9
```

2.5.2.8 Deleting row or column

To delete a row or column of a matrix, use the \texttt{empty vector} operator, [ ].

```matlab
>> A(3,:) = []
```
Third row of matrix A is now deleted. To restore the third row, we use a technique for creating a matrix

```
>> A = [A(1,:);A(2,:);[7 8 0]]
A =
     1     2     3
     4     5     6
     7     8     9
```
Matrix A is now restored to its original form.

### 2.5.2.9 Dimension
To determine the *dimensions* of a matrix or vector, use the command `size`. For example,

```
>> size(A)
ans =
     3     3
```
means 3 rows and 3 columns. Or more explicitly with,

```
>> [m,n]=size(A)
```

### 2.5.2.10 Continuation
If it is not possible to type the entire input on the same line, use consecutive periods, called an **ellipsis** . . . , to signal continuation, then continue the input on the next line.

```
B = [4/5 7.23*tan(x) sqrt(6); ...
     1/x^2 0 3/(x*log(x)); ...
     x-7 sqrt(3) x*sin(x)];
```

Note that *blank* spaces around +, −, = signs are optional, but they improve readability.

### 2.5.2.11 Transposing a matrix
The *transpose* operation is denoted by an apostrophe or a single quote (‘). It flips a matrix about its main diagonal and it turns a row vector into a column vector. Thus,
By using linear algebra notation, the transpose of \( m \times n \) real matrix \( A \) is the \( n \times m \) matrix that results from interchanging the rows and columns of \( A \). The transpose matrix is denoted \( A^T \).

### 2.5.2.12 Concatenating matrices

Matrices can be made up of sub-matrices. Here is an example. First, let’s recall our previous matrix \( A \).

\[
A =
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

The new matrix \( B \) will be,

\[
>> B = [A 10*A; -A [1 0 0; 0 1 0; 0 0 1]]
\]

\[
B =
\begin{bmatrix}
1 & 2 & 3 & 10 & 20 & 30 \\
4 & 5 & 6 & 40 & 50 & 60 \\
7 & 8 & 9 & 70 & 80 & 90 \\
-1 & -2 & -3 & 1 & 0 & 0 \\
-4 & -5 & -6 & 0 & 1 & 0 \\
-7 & -8 & -9 & 0 & 0 & 1
\end{bmatrix}
\]

### 2.5.2.13 Matrix generators

MATLAB provides functions that generate elementary matrices. The matrix of zeros, the matrix of ones, and the identity matrix are returned by the functions zeros, ones, and eye, respectively.

**Table 6: Elementary matrices.**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eye(m,n)</code></td>
<td>Returns an m-by-n matrix with 1 on the main diagonal</td>
</tr>
<tr>
<td><code>eye(n)</code></td>
<td>Returns an n-by-n square identity matrix</td>
</tr>
</tbody>
</table>
zeros(m,n)  Returns an m-by-n matrix of zeros
ones(m,n)  Returns an m-by-n matrix of ones
diag(A)  Extracts the diagonal of matrix A
rand(m,n)  Returns an m-by-n matrix of random numbers

For a complete list of elementary matrices and matrix manipulations, type help elmat or doc elmat. Here are some examples:

1.  >> b=ones(3,1)
   b =
       1
       1
       1
   Equivalently, we can define b as >> b=[1;1;1]

2.  >> eye(3)
   ans =
       1 0 0
       0 1 0
       0 0 1

3.  >> c=zeros(2,3)
   c =
       0 0 0
       0 0 0
   In addition, it is important to remember that the three elementary operations of addition (+), subtraction (−), and multiplication (*) apply also to matrices whenever the dimensions are compatible.

Two other important matrix generation functions are rand and randn, which generate matrices of (pseudo-)random numbers using the same syntax as eye.

In addition, matrices can be constructed in a block form. With C defined by C = [1 2; 3 4], we may create a matrix D as follows

   >> D = [C zeros(2); ones(2) eye(2)]
   D =
       1 2 0 0
2.5.2.14 Special matrices

MATLAB provides a number of special matrices (see Table 2.5). These matrices have interesting properties that make them useful for constructing examples and for testing algorithms. For more information, see MATLAB documentation.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hilb</td>
<td>Hilbert matrix</td>
</tr>
<tr>
<td>invhilb</td>
<td>Inverse Hilbert matrix</td>
</tr>
<tr>
<td>magic</td>
<td>Magic square</td>
</tr>
<tr>
<td>pascal</td>
<td>Pascal matrix</td>
</tr>
<tr>
<td>toeplitz</td>
<td>Toeplitz matrix</td>
</tr>
<tr>
<td>vander</td>
<td>Vandermonde matrix</td>
</tr>
<tr>
<td>wilkinson</td>
<td>Wilkinson’s eigenvalue test matrix</td>
</tr>
</tbody>
</table>

2.6 Exercises
Chapter 3: Array operations and Linear equations

3.1 Array operations

MATLAB has two different types of arithmetic operations: matrix arithmetic operations and array arithmetic operations. We have seen matrix arithmetic operations in the previous lab. Now, we are interested in array operations.

3.1.1 Matrix arithmetic operations

As we mentioned earlier, MATLAB allows arithmetic operations: +, −, *, and ^ to be carried out on matrices. Thus,

- $A+B$ or $B+A$ is valid if $A$ and $B$ are of the same size
- $A*B$ is valid if $A$'s number of column equals $B$'s number of rows
- $A^2$ is valid if $A$ is square and equals $A*A$
- $\alpha*A$ or $A*\alpha$ multiplies each element of $A$ by $\alpha$

3.1.2 Array arithmetic operations

On the other hand, array arithmetic operations or array operations for short, are done element-by-element. The period character, ., distinguishes the array operations from the matrix operations. However, since the matrix and array operations are the same for addition (+) and subtraction (−), the character pairs (.+) and (.−) are not used. The list of array operators is shown below in Table 8.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element-by-element multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element-by-element division</td>
</tr>
<tr>
<td>.^</td>
<td>Element-by-element exponentiation</td>
</tr>
</tbody>
</table>

If $A$ and $B$ are two matrices of the same size with elements $A = [a_{ij}]$ and $B = [b_{ij}]$, then the command

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

produces another matrix $C$ of the same size with elements $c_{ij} = a_{ij} b_{ij}$. For example, using the same $3 \times 3$ matrices,

$$
A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}
$$

we have,

```matlab
>> C = A.*B
```
To raise a scalar to a power, we use for example the command $10^2$. If we want the operation to be applied to each element of a matrix, we use `.^2`. For example, if we want to produce a new matrix whose elements are the square of the elements of the matrix $A$, we enter

```
>> A.^2
ans =
    1     4     9
   16    25    36
   49    64    81
```

The relations below summarize the above operations. To simplify, let’s consider two vectors $U$ and $V$ with elements $U = [u_i]$ and $V = [v_j]$.

$U. * V$ produces $[u_1 v_1 \ u_2 v_2 \ldots u_n v_n]$

$U./V$ produces $[\frac{u_1}{v_1} \ \frac{u_2}{v_2} \ldots \frac{u_n}{v_n}]$

$U.\hat{\vphantom{\ast}}^V$ produces $[u_1^{v_1} \ u_2^{v_2} \ldots u_n^{v_n}]$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Matrix</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>.*</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>./</td>
</tr>
<tr>
<td>Left division</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>^</td>
<td>.^</td>
</tr>
</tbody>
</table>

**Table 9: Summary of matrix and array operations.**

### 3.2 Solving linear equations

One of the problems encountered most frequently in scientific computation is the solution of systems of simultaneous linear equations. With matrix notation, a system of simultaneous linear equations is written

$$Ax = b$$

where there are as many equations as unknown. $A$ is a given square matrix of order $n$, $b$ is a given column vector of $n$ components, and $x$ is an unknown column vector of $n$ components.
In linear algebra we learn that the solution to $Ax = b$ can be written as $x = A^{-1}b$, where $A^{-1}$ is the inverse of $A$.

For example, consider the following system of linear equations

$$\begin{align*}
x + 2y + 3z &= 1 \\
4x + 5y + 6z &= 1 \\
7x + 8y &= 1
\end{align*}$$

The coefficient matrix $A$ is

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \text{ and the vector } b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

With matrix notation, a system of simultaneous linear equations is written $Ax = b$

This equation can be solved for $x$ using linear algebra. The result is $x = A^{-1}b$.

There are typically two ways to solve for $x$ in MATLAB:

1. The first one is to use the matrix inverse, inv.

   ```matlab
   >> A = [1 2 3; 4 5 6; 7 8 0];
   >> b = [1; 1; 1];
   >> x = inv(A)*b
   x =
   -1.0000
   1.0000
   -0.0000
   ```

2. The second one is to use the backslash (\) operator. The numerical algorithm behind this operator is computationally efficient. This is a numerically reliable way of solving system of linear equations by using a well-known process of Gaussian elimination.

   ```matlab
   >> A = [1 2 3; 4 5 6; 7 8 0];
   >> b = [1; 1; 1];
   >> x = A\b
   x =
   -1.0000
   ```
This problem is at the heart of many problems in scientific computation. Hence it is important that we know how to solve this type of problem efficiently.

Now, we know how to solve a system of linear equations. In addition to this, we will see some additional details which relate to this particular topic.

### 3.2.1 Matrix inverse

Let’s consider the same matrix $A$.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Calculating the inverse of $A$ manually is probably not a pleasant work. Here the hand-calculation of $A^{-1}$ gives as a final result:

$$A^{-1} = \frac{1}{9} \begin{bmatrix} -16 & 8 & -1 \\ 14 & -7 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

In MATLAB, however, it becomes as simple as the following commands:

```matlab
>> A = [1 2 3; 4 5 6; 7 8 0];
>> inv(A)
ans =
-1.7778 0.8889 -0.1111
1.5556 -0.7778 0.2222
-0.1111 0.2222 -0.1111
```

which is similar to:

$$A^{-1} = \frac{1}{9} \begin{bmatrix} -16 & 8 & -1 \\ 14 & -7 & 2 \\ -1 & 2 & -1 \end{bmatrix}$$

and the determinant of $A$ is

```matlab
>> det(A)
ans =
27
```
For further details on applied numerical linear algebra, see [10] and [11].

3.2.2 Matrix functions

MATLAB provides many matrix functions for various matrix/vector manipulations; see Table 10 for some of these functions. Use the online help of MATLAB to find how to use these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>det</td>
<td>Determinant</td>
</tr>
<tr>
<td>diag</td>
<td>Diagonal matrices and diagonals of a matrix</td>
</tr>
<tr>
<td>eig</td>
<td>Eigenvalues and eigenvectors</td>
</tr>
<tr>
<td>inv</td>
<td>Matrix inverse</td>
</tr>
<tr>
<td>norm</td>
<td>Matrix and vector norms</td>
</tr>
<tr>
<td>rank</td>
<td>Number of linearly independent rows or columns</td>
</tr>
</tbody>
</table>

3.3 Exercises
4 Chapter 4: Introduction to programming in MATLAB

4.1 Introduction

So far in these lab sessions, all the commands were executed in the Command Window. The problem is that the commands entered in the Command Window cannot be saved and executed again for several times. Therefore, a different way of executing repeatedly commands with MATLAB is:

1. to create a file with a list of commands,
2. save the file, and
3. run the file.

If needed, corrections or changes can be made to the commands in the file. The files that are used for this purpose are called script files or scripts for short.

This section covers the following topics:

• M-File Scripts
• M-File Functions

4.2 M-File Scripts

A script file is an external file that contains a sequence of MATLAB statements. Script files have a filename extension .m and are often called M-files. M-files can be scripts that simply execute a series of MATLAB statements, or they can be functions that can accept arguments and can produce one or more outputs.

4.2.1 Examples

Here are two simple scripts.

Example 1

Consider the system of equations:

\[
\begin{align*}
    x + 2y + 3z &= 1 \\
    3x + 2y + 4z &= 1 \\
    2x + 3y + 3z &= 2
\end{align*}
\]

Find the solution x to the system of equations.

Solution:

• Use the MATLAB editor to create a file: File → New → M-file.
• Enter the following statements in the file:
A = [1 2 3; 3 3 4; 2 3 3];
b = [1; 1; 2];
x = A\b

- Save the file, for example, example1.m.
- Run the file, in the command line, by typing:

  >> example1

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5000</td>
</tr>
<tr>
<td>1.5000</td>
</tr>
<tr>
<td>-0.5000</td>
</tr>
</tbody>
</table>

When execution completes, the variables (A, b, and x) remain in the workspace. To see a listing of them, enter who at the command prompt.

Note: The MATLAB editor is both a text editor specialized for creating M-files and a graphical MATLAB debugger. The MATLAB editor has numerous menus for tasks such as saving, viewing, and debugging. Because it performs some simple checks and also uses color to differentiate between various elements of codes, this text editor is recommended as the tool of choice for writing and editing M-files.

There is another way to open the editor:

  >> edit

or

  >> edit filename.m

to open filename.m.

Example 2

Plot the following cosine functions, \(y_1 = 2 \cos(x)\), \(y_2 = \cos(x)\), and \(y_3 = 0.5 \times \cos(x)\), in the interval \(0 \leq x \leq 2\pi\). This example has been presented in previous Chapter. Here we put the commands in a file.

Create a file, say example2.m, which contains the following commands:

\[ x = 0:pi/100:2*pi; \]
\[ y1 = 2*cos(x); \]
\[ y2 = cos(x); \]
\[ y3 = 0.5*cos(x); \]
plot(x,y1,'--',x,y2,'-',x,y3,':')
xlabel('0 \leq x \leq 2\pi')
ylabel('Cosine functions')
legend('2*cos(x)','cos(x)','0.5*cos(x)')
title('Typical example of multiple plots')
axis([0 2*pi -3 3])

Run the file by typing example2 in the Command Window.

4.2.2 Script side-effects

All variables created in a script file are added to the workspace. This may have undesirable effects, because:

- Variables already existing in the workspace may be overwritten.
- The execution of the script can be affected by the state variables in the workspace.

As a result, because scripts have some undesirable side-effects, it is better to code any complicated applications using rather function M-file.

4.3 M-File functions

As mentioned earlier, functions are programs (or routines) that accept input arguments and return output arguments. Each M-file function (or function or M-file for short) has its own area of workspace, separated from the MATLAB base workspace.

4.3.1 Anatomy of a M-File function

This simple function shows the basic parts of an M-file.

```matlab
function f = factorial(n)  % (1)

% FACTORIAL(N) returns the factorial of N.  % (2)
% Compute a factorial value.  % (3)
f = prod(1:n);  % (4)
```

The first line of a function M-file starts with the keyword function. It gives the function name and order of arguments. In the case of function factorial, there are up to one output argument and one input argument. Table 11 summarizes the M-file function.

As an example, for \( n = 5 \), the result is,

```matlab
>> f = factorial(5)
```
Table 11: Anatomy of a M-File function

<table>
<thead>
<tr>
<th>Part no.</th>
<th>M-file element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Function definition line</td>
<td>Define the function name, and the number and order of input and output arguments</td>
</tr>
<tr>
<td>(2)</td>
<td>H1 line</td>
<td>A one line summary description of the program, displayed when you request Help</td>
</tr>
<tr>
<td>(3)</td>
<td>Help text</td>
<td>A more detailed description of the program</td>
</tr>
<tr>
<td>(4)</td>
<td>Function body</td>
<td>Program code that performs the actual computations</td>
</tr>
</tbody>
</table>

Both *functions* and *scripts* can have all of these parts, except for the *function definition line* which applies to *function* only.

In addition, it is important to note that *function name* must begin with a letter, and must be no longer than the maximum of 63 characters. Furthermore, the name of the text file that you save will consist of the function name with the extension .m. Thus, the above example file would be factorial.m.

Table 12 summarizes the differences between *scripts* and *functions*.

Table 12: Difference between scripts and functions.

<table>
<thead>
<tr>
<th>Scripts</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Do not accept input arguments or return output arguments.</td>
<td>- Can accept input arguments and return output arguments.</td>
</tr>
<tr>
<td>- Store variables in a workspace that is shared with other scripts.</td>
<td>- Store variables in a workspace internal to the function.</td>
</tr>
<tr>
<td>- Are useful for automating a series of commands.</td>
<td>- Are useful for extending the MATLAB language for your application.</td>
</tr>
</tbody>
</table>

4.3.2 Input and output arguments

As mentioned above, the input arguments are listed inside parentheses following the function name. The output arguments are listed inside the brackets on the left side. They are used to transfer the output from the function file. The general form looks like this

\[
\text{function } \text{[outputs]} = \text{function_name(inputs)}
\]

Function file can have none, one, or several output arguments. Table 13 illustrates some possible combinations of input and output arguments.

Table 13: Example of input and output arguments.

<table>
<thead>
<tr>
<th>function C=FtoC(F)</th>
<th>One input argument and one output argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>function area=TrapArea(a,b,h)</td>
<td>Three inputs and one output</td>
</tr>
<tr>
<td>function [h,d]=motion(v,angle)</td>
<td>Two inputs and two outputs</td>
</tr>
</tbody>
</table>
4.4 Input/Output Commands

MATLAB has commands for inputting information in the command window and outputting data. Examples of input/output commands are echo, input, pause, keyboard, break, error, display, format, and fprintf. Brief descriptions of these commands are shown in Table 14.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>exits while or for loops</td>
</tr>
<tr>
<td>disp</td>
<td>displays text or matrix</td>
</tr>
<tr>
<td>echo</td>
<td>displays m-files during execution</td>
</tr>
<tr>
<td>error</td>
<td>displays error messages</td>
</tr>
<tr>
<td>format</td>
<td>switches output display to a particular format</td>
</tr>
<tr>
<td>fprintf</td>
<td>displays text and matrices and specifies format for printing values</td>
</tr>
<tr>
<td>input</td>
<td>allows user input</td>
</tr>
<tr>
<td>keyboard</td>
<td>invokes the keyboard as an m-file</td>
</tr>
<tr>
<td>pause</td>
<td>causes an m-file to stop executing. Pressing any key causes resumption of program execution.</td>
</tr>
</tbody>
</table>

**Break**

The `break` command may be used to terminate the execution of `for` and `while` loops. If the break command exits in an innermost part of a nested loop, the break command will exit from that loop only. The break command is useful in exiting a loop when an error condition is detected.

**Disp**

The `disp` command displays a matrix without printing its name. It can also be used to display a text string. The general form of the `disp` command is

`disp(x)`

`disp('text string')`

`disp(x)` will display the matrix x. Another way of displaying matrix x is to type its name. This is not always desirable since the display will start with a leading “x = “. `disp('text string')` will display the text string in quotes. For example, the MATLAB statement

`disp('3-by-3 identity matrix')`

will result in

3-by-3 identity matrix

and

`disp(eye(3,3))`

will result in

1 0 0
**Echo**

The echo command can be used for debugging purposes. The `echo` command allows commands to be viewed as they execute. The echo can be enabled or disabled.

- `echo on` - enables the echoing of commands
- `echo off` - disables the echoing of commands
- `echo -` by itself toggles the echo state

**Error**

The `error` command causes an error return from the m-files to the keyboard and displays a user written message. The general form of the command is

```
error('message for display')
```

Consider the following MATLAB statements:

```matlab
x = input('Enter age of student');
if x < 0
    error('wrong age was entered, try again')
end
x = input('Enter age of student')
```

For the above MATLAB statements, if the age is less than zero, the error message ‘wrong age was entered, try again’ will be displayed and the user will again be prompted for the correct age.

**Format**

The `format` controls the format of an output. Table 15 shows some formats available in MATLAB.

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>format short</td>
<td>5 significant decimal digits</td>
</tr>
<tr>
<td>format long</td>
<td>15 significant digits</td>
</tr>
<tr>
<td>format short e</td>
<td>scientific notation with 5 significant digits</td>
</tr>
<tr>
<td>format long e</td>
<td>scientific notation with 15 significant digits</td>
</tr>
<tr>
<td>format hex</td>
<td>hexadecimal</td>
</tr>
<tr>
<td>format +</td>
<td>+ printed if value is positive, - if negative; space is skipped if value is zero</td>
</tr>
</tbody>
</table>
By default, MATLAB displays numbers in “short” format (5 significant digits). **Format compact** suppresses line-feeds that appear between matrix displays, thus allowing more lines of information to be seen on the screen. **Format loose** reverts to the less compact display. Format compact and format loose do not affect the numeric format.

**fprintf**

The `fprintf` can be used to print both text and matrix values. The format for printing the matrix can be specified, and line feed can also be specified. The general form of this command is:

```plaintext
fprintf('text with format specification', matrices)
```

For example, the following statements:

```plaintext
cap = 1.0e-06;
fprintf('The value of capacitance is %7.3e Farads\n', cap)
```

when executed will yield the output:

```
The value of capacitance is 1.000e-006 Farads
```

The format specifier `%7.3e` is used to show where the matrix value should be printed in the text. `7.3e` indicates that the capacitance value should be printed with an exponential notation of 7 digits, three of which should be decimal digits. Other **format specifiers** are:

- `%f` - floating point
- `%g` - signed decimal number in either `%e` or `%f` format, whichever is shorter

The text with format specification should end with `\n` to indicate the end of line. However, we can also use `\n` to get line feeds as represented by the following example:

```plaintext
r1 = 1500;
fprintf('resistance is \n%f Ohms \n', r1)
```

the output is:

```
resistance is
1500.000000 Ohms
```

**Input**

The `input` command displays a user-written text string on the screen, waits for an input from the keyboard, and assigns the number entered on the keyboard as the value of a variable. For example, if one types the command:
r = input('Please enter the four resistor values');

when the above command is executed, the text string ‘Please, enter the four resistor values’ will be displayed on the terminal screen. The user can then type an expression such as

```
[10 15 30 25];
```

The variable r will be assigned a vector [10 15 30 25]. If the user strikes the return key, without entering an input, an empty matrix will be assigned to r.

To return a string typed by a user as a text variable, the input command may take the form

```
x = input('Enter string for prompt', 's')
```

For example, the command

```
x = input('What is the title of your graph', 's')
```

when executed, will echo on the screen, ‘What is the title of your graph.’ The user can enter a string such as ‘Voltage (mV) versus Current (mA).’

**Keyboard**

The **keyboard** command invokes the keyboard as an m-file. When the word **keyboard** is placed in an m-file, execution of the m-file stops when the word keyboard is encountered. MATLAB commands can then be entered. The keyboard mode is terminated by typing the word, “**return**” and pressing the return key. The keyboard command may be used to examine or change a variable or may be used as a tool for debugging m-files.

**Pause**

The **pause** command stops the execution of m-files. The execution of the m-file resumes upon pressing any key. The general forms of the pause command are

```
pause pause(n)
```

**Pause** stops the execution of m-files until a key is pressed. **Pause(n)** stops the execution of m-files for n seconds before continuing. The pause command can be used to stop m-files temporarily when plotting commands are encountered during program execution. If pause is not used, the graphics are momentarily visible.

4.5 **Exercises**
5 Chapter 5: Control flow and operators

5.1 Introduction

MATLAB is also a programming language. Like other computer programming languages, MATLAB has some decision making structures for control of command execution. These decision making or control flow structures include for loops, while loops, and if-else-end constructions. Control flow structures are often used in script M-files and function M-files.

By creating a file with the extension .m, we can easily write and run programs. We do not need to compile the program since MATLAB is an interpretative (not compiled) language. MATLAB has thousand of functions, and you can add your own using m-files.

MATLAB provides several tools that can be used to control the flow of a program (script or function). In a simple program as shown in the previous Chapter, the commands are executed one after the other. Here we introduce the flow control structure that make possible to skip commands or to execute specific group of commands.

5.2 Control flow

MATLAB has four control flow structures: the if statement, the for loop, the while loop, and the switch statement.

5.2.1 The ‘ ‘if...end’ ’ structure

MATLAB supports the variants of “if” construct.

- if ... end
- if ... else ... end
- if ... elseif ... else ... end

The simplest form of the if statement is

```matlab
if expression
    statements
end
```

Here are some examples based on the familiar quadratic formula.

Example 1

```matlab
discr = b*b - 4*a*c;
if discr < 0
    disp('Warning: discriminant is negative, roots are imaginary');
```
Example 2

discr = b*b - 4*a*c;
if discr < 0
    disp('Warning: discriminant is negative, roots are imaginary');
else
    disp('Roots are real, but may be repeated')
end

Example 3

discr = b*b - 4*a*c;
if discr < 0
    disp('Warning: discriminant is negative, roots are imaginary');
elseif discr == 0
    disp('Discriminant is zero, roots are repeated')
else
    disp('Roots are real')
end

It should be noted that:

- elseif has no space between else and if (one word)
- no semicolon (;) is needed at the end of lines containing if, else, end
- indentation of if block is not required, but facilitate the reading.
- the end statement is required

### 5.2.2 Relational and logical operators

A relational operator compares two numbers by determining whether a comparison is true or false. Relational operators are shown in Table 5.1.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>Relation</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Equal to</td>
</tr>
<tr>
<td><code>~=</code></td>
<td>Not equal to</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>AND operator</td>
</tr>
<tr>
<td><code>/</code></td>
<td>OR operator</td>
</tr>
<tr>
<td><code>~</code></td>
<td>NOT operator</td>
</tr>
</tbody>
</table>

Note that the “equal to” relational operator consists of two equal signs (==) (with no space between them), since = is reserved for the assignment operator.

### 5.2.3 The ‘‘for...end’’ loop

In the for ... end loop, the execution of a command is repeated at a fixed and predetermined number of times. The syntax is

```matlab
for variable = expression
    statements
end
```

Usually, expression is a vector of the form i:s:j. A simple example of for loop is

```matlab
for ii=1:5
    x=ii*ii
end
```

It is a good idea to indent the loops for readability, especially when they are nested. Note that MATLAB editor does it automatically.

Multiple for loops can be nested, in which case indentation helps to improve the readability. The following statements form the 5-by-5 symmetric matrix A with (i, j) element \( i/j \) for \( j \geq i \):

```matlab
n = 5; A = eye(n);
for j=2:n
    for i=1:j-1
        A(i,j)=i/j; A(j,i)=i/j;
    end
end
```
5.2.4 The “while...end” loop

This loop is used when the number of passes is not specified. The looping continues until a stated condition is satisfied. The while loop has the form:

```
while expression
  statements
end
```

The statements are executed as long as expression is true.

```
x = 1
while x <= 10
  x = 3*x
end
```

It is important to note that if the condition inside the looping is not well defined, the looping will continue indefinitely. If this happens, we can stop the execution by pressing Ctrl-C.

5.2.5 Other flow structures

- The break statement. A while loop can be terminated with the break statement, which passes control to the first statement after the corresponding end. The break statement can also be used to exit a for loop.
- The continue statement can also be used to exit a for loop to pass immediately to the next iteration of the loop, skipping the remaining statements in the loop.
- Other control statements include return, continue, switch, etc. For more detail about these commands, consult MATLAB documentation.

5.2.6 Operator precedence

We can build expressions that use any combination of arithmetic, relational, and logical operators. Precedence rules determine the order in which MATLAB evaluates an expression. We have already seen this in the “Tutorial Lessons”.

Here we add other operators in the list. The precedence rules for MATLAB are shown in this list (Table 5.2), ordered from highest (1) to lowest (9) precedence level. Operators are evaluated from left to right.

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parentheses ()</td>
</tr>
<tr>
<td>2</td>
<td>Transpose (. ‘), power (.^), matrix power (ˆ)</td>
</tr>
</tbody>
</table>
In addition to displaying output on the screen, the command fprintf can be used for writing the output to a file. The saved data can subsequently be used by MATLAB or other softwares.

To save the results of some computation to a file in a text format requires the following steps:

1. Open a file using fopen
2. Write the output using fprintf
3. Close the file using fclose

Here is an example (script) of its use.

```matlab
% write some variable length strings to a file
op = fopen('weekdays.txt','wt');
fprintf(op,'Sunday
Monday
Tuesday
Wednesday
');
fprintf(op,'Thursday
Friday
Saturday
');
fclose(op);
```

This file (weekdays.txt) can be opened with any program that can read .txt file.

5.4 Exercises
6 Chapter 6: Debugging M-files

6.1 Introduction

This section introduces general techniques for finding errors in M-files. Debugging is the process by which you isolate and fix errors in your program or code.

Debugging helps to correct two kinds of errors:

- Syntax errors - For example omitting a parenthesis or misspelling a function name.
- Run-time errors - Run-time errors are usually apparent and difficult to track down. They produce unexpected results.

6.2 Debugging process

We can debug the M-files using the Editor/Debugger as well as using debugging functions from the Command Window. The debugging process consists of

- Preparing for debugging
- Setting breakpoints
- Running an M-file with breakpoints
- Stepping through an M-file
- Examining values
- Correcting problems
- Ending debugging

6.2.1 Preparing for debugging

Here we use the Editor/Debugger for debugging. Do the following to prepare for debugging:

- Open the file
- Save changes
- Be sure the file you run and any files it calls are in the directories that are on the search path.

6.2.2 Setting breakpoints

Set breakpoints to pause execution of the function, so we can examine where the problem might be. There are three basic types of breakpoints:

- A standard breakpoint, which stops at a specified line.
- A conditional breakpoint, which stops at a specified line and under specified conditions.
- An error breakpoint that stops when it produces the specified type of warning, error, NaN, or infinite value.

You cannot set breakpoints while MATLAB is busy, for example, running an M-file.
6.2.3 Running with breakpoints

After setting breakpoints, run the M-file from the Editor/Debugger or from the Command Window. Running the M-file results in the following:

- The prompt in the Command Window changes to K>>
- indicating that MATLAB is in debug mode.
- The program pauses at the first breakpoint. This means that line will be executed when you continue. The pause is indicated by the green arrow.
- In breakpoint, we can examine variable, step through programs, and run other calling functions.

6.2.4 Examining values

While the program is paused, we can view the value of any variable currently in the workspace. Examine values when we want to see whether a line of code has produced the expected result or not. If the result is as expected, step to the next line, and continue running. If the result is not as expected, then that line, or the previous line, contains an error. When we run a program, the current workspace is shown in the Stack field. Use who or whos to list the variables in the current workspace.

**Viewing values as datatips**

First, we position the cursor to the left of a variable on that line. Its current value appears. This is called a datatip, which is like a tooltip for data. If you have trouble getting the datatip to appear, click in the line and then move the cursor next to the variable.

6.2.5 Correcting and ending debugging

While debugging, we can change the value of a variable to see if the new value produces expected results. While the program is paused, assign a new value to the variable in the Command Window, Workspace browser, or Array Editor. Then continue running and stepping through the program.

6.2.6 Ending debugging

After identifying a problem, end the debugging session. It is best to quit debug mode before editing an M-file. Otherwise, you can get unexpected results when you run the file. To end debugging, select Exit Debug Mode from the Debug menu.

6.2.7 Correcting an M-file

To correct errors in an M-file,

- Quit debugging
- Do not make changes to an M-file while MATLAB is in debug mode
- Make changes to the M-file
- Save the M-file
• Clear breakpoints
• Run the M-file again to be sure it produces the expected results.

For details on debugging process, see MATLAB documentation.
## 7 Appendix A: Summary of commands

### Table 17: Arithmetic operators and special characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication (scalar and array)</td>
</tr>
<tr>
<td>/</td>
<td>Division (right)</td>
</tr>
<tr>
<td>^</td>
<td>Power or exponentiation</td>
</tr>
<tr>
<td>:</td>
<td>Colon; creates vectors with equally spaced elements</td>
</tr>
<tr>
<td>;</td>
<td>Semi-colon; suppresses display; ends row in array</td>
</tr>
<tr>
<td>,</td>
<td>Comma; separates array subscripts</td>
</tr>
<tr>
<td>. . .</td>
<td>Continuation of lines</td>
</tr>
<tr>
<td>%</td>
<td>Percent; denotes a comment; specifies output format</td>
</tr>
<tr>
<td>'</td>
<td>Single quote; creates string; specifies matrix transpose</td>
</tr>
<tr>
<td>=</td>
<td>Assignment operator</td>
</tr>
<tr>
<td>( )</td>
<td>Parentheses; encloses elements of arrays and input arguments</td>
</tr>
<tr>
<td>[ ]</td>
<td>Brackets; encloses matrix elements and output arguments</td>
</tr>
</tbody>
</table>

### Table 18: Array operators

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Array multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Array (right) division</td>
</tr>
<tr>
<td>.^</td>
<td>Array power</td>
</tr>
<tr>
<td>.\</td>
<td>Array (left) division</td>
</tr>
<tr>
<td>;'</td>
<td>Array (nonconjugated) transpose</td>
</tr>
</tbody>
</table>

### Table 19: Relational and logical operators

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>≤</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>≥</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>~=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical or element-wise AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;&amp;</td>
<td>Short-circuit AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 20: Managing workspace and file commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>Change current directory</td>
</tr>
<tr>
<td>clc</td>
<td>Clear the Command Window</td>
</tr>
<tr>
<td>clear (all)</td>
<td>Removes all variables from the workspace</td>
</tr>
<tr>
<td>clear x</td>
<td>Remove x from the workspace</td>
</tr>
<tr>
<td>copyfile</td>
<td>Copy file or directory</td>
</tr>
<tr>
<td>delete</td>
<td>Delete files</td>
</tr>
<tr>
<td>Dir</td>
<td>Display directory listing</td>
</tr>
<tr>
<td>exist</td>
<td>Check if variables or functions are defined</td>
</tr>
<tr>
<td>Help</td>
<td>Display help for MATLAB functions</td>
</tr>
<tr>
<td>lookfor</td>
<td>Search for specified word in all help entries</td>
</tr>
<tr>
<td>mkdir</td>
<td>Make new directory</td>
</tr>
<tr>
<td>movefile</td>
<td>Move file or directory</td>
</tr>
<tr>
<td>pwd</td>
<td>Identify current directory</td>
</tr>
<tr>
<td>rmdir</td>
<td>Remove directory</td>
</tr>
<tr>
<td>type</td>
<td>Display contents of file</td>
</tr>
<tr>
<td>what</td>
<td>List MATLAB files in current directory</td>
</tr>
<tr>
<td>which</td>
<td>Locate functions and files</td>
</tr>
<tr>
<td>who</td>
<td>Display variables currently in the workspace</td>
</tr>
<tr>
<td>whos</td>
<td>Display information on variables in the workspace</td>
</tr>
</tbody>
</table>

Table 21: Predefined variables and math constants

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ans</td>
<td>Value of last variable (answer)</td>
</tr>
<tr>
<td>eps</td>
<td>Floating-point relative accuracy</td>
</tr>
<tr>
<td>i</td>
<td>Imaginary unit of a complex number</td>
</tr>
<tr>
<td>Inf</td>
<td>Infinity ((\infty))</td>
</tr>
<tr>
<td>eps</td>
<td>Floating-point relative accuracy</td>
</tr>
<tr>
<td>j</td>
<td>Imaginary unit of a complex number</td>
</tr>
<tr>
<td>NaN</td>
<td>Not a number</td>
</tr>
<tr>
<td>pi</td>
<td>The number (\pi) (3.14159 ...)</td>
</tr>
</tbody>
</table>

Table 22: Elementary matrices and arrays

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eye</td>
<td>Identity matrix</td>
</tr>
<tr>
<td>linspace</td>
<td>Generate linearly space vectors</td>
</tr>
<tr>
<td>ones</td>
<td>Create array of all ones</td>
</tr>
<tr>
<td>rand</td>
<td>Uniformly distributed random numbers and arrays</td>
</tr>
<tr>
<td>zeros</td>
<td>Create array of all zeros</td>
</tr>
</tbody>
</table>
Table 23: Arrays and Matrices: Basic information

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disp</td>
<td>Display text or array</td>
</tr>
<tr>
<td>isempty</td>
<td>Determine if input is empty matrix</td>
</tr>
<tr>
<td>isequal</td>
<td>Test arrays for equality</td>
</tr>
<tr>
<td>length</td>
<td>Length of vector</td>
</tr>
<tr>
<td>ndims</td>
<td>Number of dimensions</td>
</tr>
<tr>
<td>numel</td>
<td>Number of elements</td>
</tr>
<tr>
<td>size</td>
<td>Size of matrix</td>
</tr>
</tbody>
</table>

Table A.8: Arrays and Matrices: operations and manipulation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross</td>
<td>Vector cross product</td>
</tr>
<tr>
<td>diag</td>
<td>Diagonal matrices and diagonals of matrix</td>
</tr>
<tr>
<td>dot</td>
<td>Vector dot product</td>
</tr>
<tr>
<td>end</td>
<td>Indicate last index of array</td>
</tr>
<tr>
<td>find</td>
<td>Find indices of nonzero elements</td>
</tr>
<tr>
<td>kron</td>
<td>Kronecker tensor product</td>
</tr>
<tr>
<td>max</td>
<td>Maximum value of array</td>
</tr>
<tr>
<td>min</td>
<td>Minimum value of array</td>
</tr>
<tr>
<td>prod</td>
<td>Product of array elements</td>
</tr>
<tr>
<td>reshape</td>
<td>Reshape array</td>
</tr>
<tr>
<td>sort</td>
<td>Sort array elements</td>
</tr>
<tr>
<td>sum</td>
<td>Sum of array elements</td>
</tr>
<tr>
<td>size</td>
<td>Size of matrix</td>
</tr>
</tbody>
</table>

Table 24: Arrays and Matrices: matrix analysis and linear equations.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Condition number with respect to inversion</td>
</tr>
<tr>
<td>det</td>
<td>Determinant</td>
</tr>
<tr>
<td>inv</td>
<td>Matrix inverse</td>
</tr>
<tr>
<td>linsolve</td>
<td>Solve linear system of equations</td>
</tr>
<tr>
<td>lu</td>
<td>LU factorization</td>
</tr>
<tr>
<td>norm</td>
<td>Matrix or vector norm</td>
</tr>
<tr>
<td>null</td>
<td>Null space</td>
</tr>
<tr>
<td>orth</td>
<td>Orthogonalization</td>
</tr>
<tr>
<td>rank</td>
<td>Matrix rank</td>
</tr>
<tr>
<td>rref</td>
<td>Reduced row echelon form</td>
</tr>
<tr>
<td>trace</td>
<td>Sum of diagonal elements</td>
</tr>
</tbody>
</table>
8 Appendix C: Main characteristics of MATLAB

8.1 History

- Developed primarily by Cleve Moler in the 1970’s
- Derived from FORTRAN subroutines LINPACK and EISPACK, linear and eigenvalue systems.
- Developed primarily as an interactive system to access LINPACK and EISPACK.
- Gained its popularity through word of mouth, because it was not officially distributed.
- Rewritten in C in the 1980’s with more functionality, which include plotting routines.

The MathWorks Inc. was created (1984) to market and continue development of MATLAB.

According to Cleve Moler, three other men played important roles in the origins of MATLAB: J. H. Wilkinson, George Forsythe, and John Todd. It is also interesting to mention the authors of LINPACK: Jack Dongara, Pete Steward, Jim Bunch, and Cleve Moler. Since then another package emerged: LAPACK. LAPACK stands for Linear Algebra Package. It has been designed to supersede LINPACK and EISPACK.

8.2 Strengths

- MATLAB may behave as a calculator or as a programming language
- MATLAB combine nicely calculation and graphic plotting.
- MATLAB is relatively easy to learn
- MATLAB is interpreted (not compiled), errors are easy to fix
- MATLAB is optimized to be relatively fast when performing matrix operations
- MATLAB does have some object-oriented elements

8.3 Weaknesses

- MATLAB is not a general purpose programming language such as C, C++, or FORTRAN
- MATLAB is designed for scientific computing, and is not well suitable for other applications
- MATLAB is an interpreted language, slower than a compiled language such as C++
- MATLAB commands are specific for MATLAB usage. Most of them do not have a direct equivalent with other programming language commands

8.4 Competition

- One of MATLAB’s competitors is Mathematica, the symbolic computation program.
- MATLAB is more convenient for numerical analysis and linear algebra. It is frequently used in engineering community.
- Mathematica has superior symbolic manipulation, making it popular among physicists.
- There are other competitors:
  - Scilab
  - GNU Octave
  - Rlab

70
9 Bibliography


Problem 1

The voltage, $v$, across a resistance is given as (Ohm’s Law), $v = Ri$, where $i$ is the current and $R$ the resistance. The power dissipated in resistor $R$ is given by the expression

$$ P = Ri^2 $$

If $R = 10$ Ohms and the current is increased from 0 to 10 A with increments of 2A, write a MATLAB program to generate a table of current, voltage and power dissipation.

Solution:

MATLAB Script

```matlab
diary ex1_1.dat
% diary causes output to be written into file ex1_1.dat
% Voltage and power calculation R=10; % Resistance value
i=(0:2:10); % Generate current values
v=i.*R; % array multiplication to obtain voltage
p=(i.^2)*R;  % power calculation
sol=[i v p] % current, voltage and power values are printed
% the last diary command turns off the diary state
```

MATLAB produces the following result:

```
sol =

Columns 1 through 6
0   2   4   6   8   10

Columns 7 through 12
0  20  40  60  80  100

Columns 13 through 18
0 40 160 360 640 1000
```

Columns 1 through 6 constitute the current values, columns 7 through 12 are the voltages, and columns 13 through 18 are the power dissipation values.

Problem 2

Simplify the complex number $z$ and express it both in rectangular and polar form.

$$ Z = \frac{(3 + j4)(5 + j2)(2\angle60^\circ)}{(3 + j6)(1 + j2)} $$
Solution:

The following program shows the script file that was used to evaluate the complex number, z, and express the result in polar notation and rectangular form.

MATLAB Script

```matlab
diary ex1_2.dat
% Evaluation of Z
% the complex numbers are entered
Z1 = 3+4*j; Z2 = 5+2*j;
theta = (60/180)*pi; % angle in radians
Z3 = 2*exp(j*theta); Z4 = 3+6*j;
Z5 = 1+2*j;
% Z_rect is complex number Z in rectangular form
disp('Z in rectangular form is'); % displays text inside brackets
Z_rect = Z1*Z2*Z3/(Z4+Z5); Z_rect
Z_mag = abs(Z_rect); % magnitude of Z
Z_angle = angle(Z_rect)*(180/pi); % Angle in degrees
disp('complex number Z in polar form, mag, phase'); % displays text inside brackets
% inside brackets
Z_polar = [Z_mag, Z_angle]
diary
```

The program is named ex1_2.m. It is included in the disk that accompanies this book. Execute it by typing ex1_2 in the MATLAB command window. Observe the result, which should be

Z in rectangular form is

```
Z_rect =
1.9108 + 5.7095i
```

complex number Z in polar form (magnitude and phase) is

```
Z_polar =
```
Problem 3

Write a function file to solve the equivalent resistance of series connected resistors, R1, R2, R3, ..., Rn.

Solution:

MATLAB Script

```matlab
function req = equiv_sr(r)
% equiv_sr is a function program for obtaining
% the equivalent resistance of series
% connected resistors
% usage: req = equiv_sr(r)
% r is an input vector of length n
% req is an output, the equivalent resistance(scalar)

n = length(r);
% number of resistors req = sum (r);
% sum up all resistors
end
```

The above MATLAB script can be found in the function file equiv_sr.m, which is available on the disk that accompanies this book.

Suppose we want to find the equivalent resistance of the series connected resistors 10, 20, 15, 16 and 5 ohms. The following statements can be typed in the MATLAB command window to reference the function equiv_sr

```matlab
a = [10 20 15 16 5];
Rseries = equiv_sr(a)
diary
```

The result obtained from MATLAB is

```
Rseries =
66
```
Problem 4

Write a MATLAB function to obtain the roots of the quadratic equation

\[ ax^2 + bx + c = 0 \]

Solution:

MATLAB Script

```matlab
function rt = rt_quad(coef)

% rt_quad is a function for obtaining the roots of
% of a quadratic equation
% usage: rt = rt_quad(coef)
% coef is the coefficients a,b,c of the quadratic
% equation ax^2 + bx + c =0
% rt are the roots, vector of length 2
% coefficient a, b, c are obtained from vector coef a = coef(1); b = coef(2); c = coef(3);
int = b^2 - 4*a*c;
if int > 0
srint = sqrt(int);
x1 = (-b + srint)/(2*a);
x2 = (-b - srint)/(2*a);
elseif int == 0
x1 = -b/(2*a); x2 = x1;
elseif int < 0
srint = sqrt(-int);
p1 = -b/(2*a);
p2 = srint/(2*a); x1 = p1+p2*j; x2 = p1-p2*j;
end
rt = [x1;
```
The above MATLAB script can be found in the function file rt_quad.m, which is available on the disk that accompanies this book.

We can use m-file function, rt_quad, to find the roots of the following quadratic equations:

(a) \[ x^2 + 3x + 2 = 0 \]
(b) \[ x^2 + 2x + 1 = 0 \]
(c) \[ x^2 - 2x + 3 = 0 \]

The following statements, that can be found in the m-file ex1_4.m, can be used to obtain the roots:

```matlab
diary ex1_4.dat
ca = [1 3 2];
ra = rt_quad(ca)
cb = [1 2 1];
rb = rt_quad(cb)
cc = [1 -2 3];
rc = rt_quad(cc)
diary
```

Type into the MATLAB command window the statement `ex1_4` and observe the results. The following results will be obtained:

```matlab
ra =
  -1
  -2
rb =
  -1
  -1
rc=
```
Problem 5
The voltage across a discharging capacitor is

\[ v(t) = 10(1 - e^{-0.2t}) \]

Generate a table of voltage, \( v(t) \), versus time, \( t \), for \( t = 0 \) to 50 seconds with increment of 5 s.

Problem 6
Use MATLAB to evaluate the complex number

\[ Z = \frac{(3 + j6)(6 + j4)}{(2 + j1)j2} + 7 + j10 \]

Problem 7
Write a function-file to obtain the dot product and the vector product of two vectors \( \mathbf{a} \) and \( \mathbf{b} \). Use the function to evaluate the dot and vector products of vectors \( \mathbf{x} \) and \( \mathbf{y} \), where \( \mathbf{x} = (1 \ 5 \ 6) \) and \( \mathbf{y} = (2 \ 3 \ 8) \).

Problem 8
Write a function-file that can be used to calculate the equivalent resistance of \( n \) parallel connected resistors. In general, the equivalent resistance of resistors \( R_1, R_2, R_3, \ldots, R_n \) is given by

\[ \frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_n} \]

Problem 9
The voltage \( V \) is given as \( V = RI \), where \( R \) and \( I \) are resistance matrix and current vector. Evaluate \( V \) given that

\[ R = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 3 & 6 \\ 3 & 6 & 7 \end{bmatrix} \quad \text{and} \quad I = \begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix} \]

Problem 10
Use MATLAB to simplify the expression

\[ y = 0.5 + j6 + 3.5e^{j0.6} + (3 + j6)e^{j0.3\pi} \]

Problem 11
Write a function file to evaluate n factorial (i.e. n!); where

\[ n! = n(n-1)(n-2) \cdots (2)(1) \]

Use the function to compute

\[ x = \frac{7!}{3!4!} \]

**Problem 12**

For a triangle with sides of length a, b, and c, the area A is given as

\[ A = \sqrt{s(s-a)(s-b)(s-c)} \]

where

\[ s = \frac{(a + b + c)}{2} \]

Write a function to compute the area of triangles given the sides of a triangle. Use the function to compute the area of triangles with the lengths:

(a) 56, 27 and 43
(b) 5, 12 and 13.

**Problem 13**

For an R-L circuit, the voltage \( v(t) \) and current \( i(t) \) are given as

\[ v(t) = 10 \cos(377t) \]

\[ i(t) = 5 \cos(377t + 60^\circ) \]

Sketch \( v(t) \) and \( i(t) \) for \( t = 0 \) to 20 milliseconds.

**Solution**

MATLAB Script

```matlab
% RL circuit
% current i(t) and voltage v(t) are generated; t is time t = 0:1E-3:20E-3; v = 10*cos(377*t);
% a_rad = (60*pi/180); % angle in radians i = 5*cos(377*t + a_rad); plot(t,v,*',t,i,'o')
title('Voltage and Current of an RL circuit')
xlabel('Sec')
ylabel('Voltage(V) and Current(mA)')
```
Figure 18 shows the resulting graph. The file ex2_1.m is a script file for the solution of the problem.

**Problem 14**

The gain versus frequency of a capacitively coupled amplifier is shown below. Draw a graph of gain versus frequency using a logarithmic scale for the frequency and a linear scale for the gain.

<table>
<thead>
<tr>
<th>Frequency (Hz)</th>
<th>Gain (dB)</th>
<th>Frequency (Hz)</th>
<th>Gain (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>5</td>
<td>2000</td>
<td>34</td>
</tr>
<tr>
<td>40</td>
<td>10</td>
<td>5000</td>
<td>34</td>
</tr>
<tr>
<td>80</td>
<td>30</td>
<td>8000</td>
<td>34</td>
</tr>
<tr>
<td>100</td>
<td>32</td>
<td>10000</td>
<td>32</td>
</tr>
<tr>
<td>120</td>
<td>34</td>
<td>12000</td>
<td>30</td>
</tr>
</tbody>
</table>

**Solution**

MATLAB Script

```matlab
% Bode plot for capacitively coupled amplifier
f = [20 40 80 100 120 2000 5000 8000 10000 ... 12000 15000 20000];
```
The plot is shown in Figure 19. The MATLAB script file is ex2_2.m.

Problem 15

A complex number $z$ can be represented as $z = re^{j\theta}$. The $n^{th}$ power of the complex number is given as $z^n = r^n e^{j n\theta}$. If $r = 1.2$ and $\theta = 10^\circ$, use the polar plot to plot $|z^n|$ versus $n\theta$ for $n = 1$ to $n = 36$.

Solution

MATLAB Script

```matlab
% polar plot of z
r = 1.2; theta = 10*pi/180;
angle = 0:theta:36*theta; mag = r.^((angle/theta));
polar(angle,mag)
grid
title('Polar Plot')
```
The polar plot is shown in Figure 20.

Figure 20: Polar Plot of \( z = 1.2^n e^{10n} \).

**Problem 16**

The repulsive Coulomb force that exists between two protons in the nucleus of a conductor is given as

\[
F = \frac{q_1 q_2}{4\pi \varepsilon_0 r^2}
\]

If

\[
q_1 = q_2 = 1.6 \times 10^{-19} \text{ C}, \quad \text{and} \quad \frac{1}{4\pi \varepsilon_0} = 8.99 \times 10^9 \text{ Nm}^2/\text{C}^2
\]

sketch a graph of force versus radius \( r \). Assume a radius from \( 1.0 \times 10^{-15} \) to \( 1.0 \times 10^{-14} \) m with increments of \( 2.0 \times 10^{-15} \) m.

**Problem 17**

The current flowing through a drain of a field effect transistor during saturation is given as

\[
i_{DS} = k(V_{GS} - V_t)^2
\]

If \( V_t = 1.0 \) volt and \( k = 2.5mA/V^2 \) plot the current \( i_{DS} \) for the following values of \( V_{GS} \): 1.5, 2.0, 2.5, ..., 5V.

**Problem 18**

Plot the voltage across a parallel RLC circuit given as

\[
v(t) = 5e^{2t} \sin(1000\pi t)
\]
Problem 19

Obtain the polar plot of \( z = r^{-n}e^{jn\theta} \) for \( \theta = 15^\circ \) and \( n = 1 \) to 20.

Problem 20

The table below shows the grades of three examinations of ten students in a class.

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>EXAM #1</th>
<th>EXAM #2</th>
<th>EXAM #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>81</td>
<td>78</td>
<td>83</td>
</tr>
<tr>
<td>2</td>
<td>75</td>
<td>77</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>95</td>
<td>90</td>
<td>93</td>
</tr>
<tr>
<td>4</td>
<td>65</td>
<td>69</td>
<td>72</td>
</tr>
<tr>
<td>5</td>
<td>72</td>
<td>73</td>
<td>71</td>
</tr>
<tr>
<td>6</td>
<td>79</td>
<td>84</td>
<td>86</td>
</tr>
<tr>
<td>7</td>
<td>93</td>
<td>97</td>
<td>94</td>
</tr>
<tr>
<td>8</td>
<td>69</td>
<td>72</td>
<td>67</td>
</tr>
<tr>
<td>9</td>
<td>83</td>
<td>80</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>87</td>
<td>81</td>
<td>77</td>
</tr>
</tbody>
</table>

(a) Plot the results of each examination.

(b) Use MATLAB to calculate the mean and standard deviation of each examination.

Problem 21

A function \( f(x) \) is given as

\[
f(x) = x^4 + 3x^3 + 4x^2 + 2x + 6
\]

(a) Plot \( f(x) \) and

(b) Find the roots of \( f(x) \)

Problem 22

A message signal \( m(t) \) and the carrier signal \( c(t) \) of a communication system are, respectively:

\[
m(t) = 4 \cos(120\pi t) + 2 \cos(240\pi t)
\]
\[
c(t) = 10 \cos(10,000\pi t)
\]

A double-sideband suppressed carrier \( s(t) \) is given as \( s(t) = m(t)c(t) \)

Plot \( m(t) \), \( c(t) \) and \( s(t) \) using the subplot command.

Problem 23

The voltage \( v \) and current \( I \) of a certain diode are related by the expression

\[
i = I_S \exp\left[\frac{v}{(nV_T)}\right]
\]
If \( I_S = 1.0 \times 10^{-14} \text{A} \), \( n = 2.0 \) and \( V_T = 26 \text{ mV} \),

plot the current versus voltage curve of the diode for diode voltage between 0 and 0.6 volts.

**Problem 24**

The horizontal displacement \( x(t) \) and vertical displacement \( y(t) \) are given with respect to time, \( t \), as

\[
\begin{align*}
\quad x(t) &= 2t \\
\quad y(t) &= \sin(t)
\end{align*}
\]

For \( t = 0 \) to \( 10 \text{ ms} \), determine the values of \( x(t) \) and \( y(t) \). Use the values to plot \( x(t) \) versus \( y(t) \).

**Solution:**

MATLAB Script

```matlab
% for i= 0:10
x(i+1) = 2*i;
y(i+1) = 2*sin(i);
end
plot(x,y)
```

Figure 21 shows the plots of \( x(t) \) and \( y(t) \).

![Figure 21: Plot of x versus y.](image-url)
Problem 25

A 3-bit A/D converter, with an analog input $x$ and digital output $y$, is represented by the equation:

$$
\begin{array}{c|c}
 y & x \leq -2.5 \\
 0 & -2.5 \leq x < -1.5 \\
 1 & -1.5 \leq x < -0.5 \\
 2 & -0.5 \leq x < 0.5 \\
 3 & 0.5 \leq x < 1.5 \\
 4 & 1.5 \leq x < 2.5 \\
 5 & 2.5 \leq x < 3.5 \\
 6 & x \geq 3.5 \\
\end{array}
$$

Write a MATLAB program to convert analog signal $x$ to digital signal $y$. Test the program by using an analog signal with the following amplitudes: -1.25, 2.57 and 6.0.

Solution

MATLAB Script

diary ex3_2.dat

% $y_1 = \text{bitatd}_3(-1.25)$
% $y_2 = \text{bitatd}_3(2.57)$
% $y_3 = \text{bitatd}_3(6.0)$

diary

function $Y_{\text{dig}} = \text{bitatd}_3(X_{\text{analog}})$

%% bitatd_3 is a function program for obtaining
%% the digital value given an input analog
%% signal

%%

%% usage: $Y_{\text{dig}} = \text{bitatd}_3(X_{\text{analog}})$
%% $Y_{\text{dig}}$ is the digital number (in integer form)
%% $X_{\text{analog}}$ is the analog input (in decimal form)

%
if X_analog < -2.5
    Y_dig = 0;
elseif X_analog >= -2.5 & X_analog < -1.5
    Y_dig = 1;
elseif X_analog >= -1.5 & X_analog < -0.5
    Y_dig = 2;
elseif X_analog >= -0.5 & X_analog < 0.5
    Y_dig = 3;
elseif X_analog >= 0.5 & X_analog < 1.5
    Y_dig = 4;
elseif X_analog >= 1.5 & X_analog < 2.5
    Y_dig = 5;
elseif X_analog >= 2.5 & X_analog < 3.5
    Y_dig = 6;
else
    Y_dig = 7;
end
Y_dig;
end

The function file, bitatd_3.m, is an m-file available in the disk that accompanies this book. In addition, the script file, ex3_2.m on the disk, can be used to perform this example. The results obtained, when the latter program is executed, are

\[ y_1 = 2 \]
\[ y_2 = 6 \]
\[ y_3 = \]
Problem 26

Determine the number of consecutive integer numbers which when added together will give a value equal to or just less than 210.

Solution

MATLAB Script

diary ex3_3.dat

% integer summation
int = 1; int_sum = 0;
max_val = 210;
while int_sum < max_val
    int_sum = int_sum + int;
    int = int + 1;
end
last_int = int
if int_sum == max_val
    num_int = int - 1
    tt_int_ct = int_sum
elseif int_sum > max_val
    num_int = int - 1
    tt_int_ct = int_sum - last_int
end
end
diary

The solution obtained will be

last_int =
21
Thus, the number of integers starting from 1 that would add up to 210 is 20.

That is, \(1 + 2 + 3 + 4 + ... + 20 = 210\)

\textbf{Problem 27}

Write a MATLAB program to add all the even numbers from 0 to 100.

\textbf{Problem 28}

Add all the terms in the series

\[1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\]

until the sum exceeds 1.995. Print out the sum and the number of terms needed to just exceed the sum of 1.995.

\textbf{Problem 29}

The Fibonacci sequence is given as 1 1 2 3 5 8 13 21 34 ...

Write a MATLAB program to generate the Fibonacci sequence up to the twelfth term. Print out the results.

\textbf{Problem 30}

The table below shows the final course grade and its corresponding relevant letter grade.

<table>
<thead>
<tr>
<th>Letter Grade</th>
<th>Final Course Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>90 (&lt; \text{grade} \leq 100</td>
</tr>
<tr>
<td>B</td>
<td>80 (&lt; \text{grade} \leq 90</td>
</tr>
<tr>
<td>C</td>
<td>70 (&lt; \text{grade} \leq 80</td>
</tr>
<tr>
<td>D</td>
<td>60 (&lt; \text{grade} \leq 70</td>
</tr>
<tr>
<td>F</td>
<td>\text{grade} \leq 60</td>
</tr>
</tbody>
</table>

For the course grades: 70, 85, 90, 97, 50, 60, 71, 83, 91, 86, 77, 45, 67, 88, 64, 79, 75, 92, and 69

(a) Determine the number of students who attained the grade of A and F.

(b) What are the mean grade and the standard deviation?

\textbf{Problem 31}

$$y[n] = 2y[n-1] - y[n-2] + x[n]$$

for $n \geq 0$. Assume that $x[n] = 1$ for $n \geq 0$, $y[-2] = 2$ and $y[-1] = 1$.

**Problem 32**

The equivalent impedance of a circuit is given as

$$Z_{eq}(jw) = 100 + jwL + \frac{1}{jwC}$$

If $L = 4$ H and $C = 1 \mu F$,

(a) Plot $|Z_{eq}(jw)|$ versus $w$.

(b) What is the minimum impedance?

(c) With what frequency does the minimum impedance occur?

**Problem 33**

For the circuit shown below, find the nodal voltages $V_1$, $V_2$ and $V_3$.

![Circuit with Nodal Voltages](image)

**Figure 22: Circuit with Nodal Voltages.**

**Solution:**
Using KCL and assuming that the currents leaving a node are positive, we have

For node 1,
\[
\frac{V_1 - V_2}{10} + \frac{V_1 - V_3}{20} - 5 = 0
\]
i.e.,
\[
0.15V_1 - 0.1V_2 - 0.05V_3 = 5
\]

At node 2,
\[
\frac{V_2 - V_1}{10} + \frac{V_2}{50} + \frac{V_2 - V_3}{40} = 0
\]
i.e.,
\[
-0.1V_1 + 0.145V_2 - 0.025V_3 = 0
\]

At node 3,
\[
\frac{V_3 - V_1}{20} + \frac{V_3 - V_2}{40} - 2 = 0
\]
i.e.,
\[
-0.05V_1 - 0.025V_2 + 0.075V_3 = 2
\]

The MATLAB program for solving the nodal voltages is

MATLAB Script

diary ex4_1.dat

% program computes the nodal voltages
% given the admittance matrix Y and current vector I
% Y is the admittance matrix and I is the current vector
% initialize matrix y and vector I using YV=I form
Y = [ 0.15 -0.1 -0.05; -0.1 0.145 -0.025; -0.05 -0.025 0.075];
I = [5; 0; 2];
% solve for the voltage
fprintf('Nodal voltages V1, V2 and V3 are \n')
The results obtained from MATLAB are Nodal voltages $V_1$, $V_2$ and $V_3$,

\[ \begin{align*}
V &= \\
&= 404.2857 \\
&= 350.0000 \\
&= 412.8571
\end{align*} \]

**Problem 34**

For the circuit shown below, find the nodal voltages $V_1$, $V_2$ and $V_3$.

![Circuit with Dependent and Independent Sources](image)

**Solution:**

Using KCL and the convention that currents leaving a node is positive, we have

At node 1

\[ \frac{V_1}{20} + \frac{V_1 - V_2}{5} + \frac{V_1 - V_4}{2} - 5 = 0 \]

Simplifying, we get

\[ 0.75V_1 - 0.2V_2 - 0.5V_4 = 5 \]
The MATLAB program for solving the nodal voltages is

MATLAB Script

diary ex4_2.dat

% this program computes the nodal voltages
% given the admittance matrix Y and current vector I
% Y is the admittance matrix
% I is the current vector
% initialize the matrix Y and vector I using YV=I

\[ Y = \begin{bmatrix} 0.75 & -0.2 & 0 & -0.5 \\
-5 & 1 & -1 & 5 \\
-0.2 & 0.45 & 0.166666667 & -0.066666667 \\
0 & 0 & 0 & 1 \end{bmatrix}; \]

% current vector is entered as a transpose of row vector

\[ I = \begin{bmatrix} 5 \\
0 \\
0 \\
10 \end{bmatrix}'; \]

% solve for nodal voltage

\[ V = \text{inv}(Y) * I \]

We obtain the following results.

Nodal voltages V1,V2,V3,V4 are

\[ V = \begin{bmatrix} 18.1107 \\
17.9153 \\
-22.6384 \\
10.0000 \end{bmatrix} \]

**Problem 35**

Use the mesh analysis to find the current flowing through the resistor \( R_B \). In addition, find the power supplied by the 10-volt voltage source.

![Figure 24: Bridge circuit.](image)
Solution

Using loop analysis and designating the loop currents as $I_1, I_2, I_3$, we obtain the following figure.

![Bridge Circuit with Loop Currents](image)

Note that $I = I_3 - I_2$ and power supplied by the source is $P = 10I_1$

The loop equations are

Loop 1.

$10(I_1 - I_2) + 30(I_1 - I_3) - 10 = 0$

$40I_1 - 10I_2 - 30I_3 = 10$

Loop 2.

$10(I_2 - I_1) + 15I_2 + 5(I_2 - I_3) = 0$

$-10I_1 + 30I_2 - 5I_3 = 0$

Loop 3.

$30(I_3 - I_1) + 5(I_3 - I_2) + 30I_3 = 0$

$-30I_1 - 5I_2 + 65I_3 = 0$

In matrix form,

$$
\begin{bmatrix}
40 & -10 & -30 \\
-10 & 30 & -5 \\
-30 & -5 & 65
\end{bmatrix}
\begin{bmatrix}
I_1 \\
I_2 \\
I_3
\end{bmatrix}
= 
\begin{bmatrix}
10 \\
0 \\
0
\end{bmatrix}
$$

The MATLAB program for solving the loop currents $I_1, I_2, I_3$, the current $I$ and the power supplied by the 10-volt source is
diary ex4_3.dat

% this program determines the current
% flowing in a resistor RB and power supplied by source
% it computes the loop currents given the impedance
% matrix Z and voltage vector V
% Z is the impedance matrix
% V is the voltage matrix
% initialize the matrix Z and vector V
Z = [40 -10 -30;
    -10  30 -5;
    -30 -5  65];
V = [10 0 0]';
% solve for the loop currents
I = inv(Z)*V;
% current through RB is calculated
IRB = I(3) - I(2);
fprintf('the current through R is %8.3f Amps \n',IRB)
% the power supplied by source is calculated
PS = I(1)*10;
fprintf('the power supplied by 10V source is %8.4f watts \n',PS)

MATLAB answers are

the current through R is 0.037 Amps
the power supplied by 10V source is 4.7531 watts
Problem 36

Find the nodal voltages in the circuit, i.e., $V_1, V_2, \ldots, V_5$

![Circuit Diagram](image)

**Figure 26**: circuit for this program.

Solution

The MATLAB program for obtaining the nodal voltages is shown below.

MATLAB Script

```matlab
% Program determines the nodal voltages
diary ex4_5.dat
% given an admittance matrix Y and current vector I
% Initialize matrix Y and the current vector I of
% matrix equation Y V = I
Y = [-4.4 0.125 -0.125 4.9 0;
    -0.1 0.325 -0.2 0.55 -0.25;
    0 -0.125 0.325 -0.2 0;
    1 0 -0.6 -0.4 0;
    0 0 0 0 1];
I = [0 0 5 -40 24];
V = inv(Y)*I;
diary
```

The nodal voltages $V(1), V(2), \ldots, V(5)$ are:

$V_1 = V(1)$, $V_2 = V(2)$, $V_3 = V(3)$, $V_4 = V(4)$, $V_5 = V(5)$.
The results obtained from MATLAB are

Nodal voltages V(1), V(2), ... V(5) are

\[ V = \begin{bmatrix} 117.4792 \\ 299.7708 \\ 193.9375 \\ 102.7917 \\ 24.0000 \end{bmatrix} \]

**Problem 37**

In Figure 27, as RL \( \text{varies from 0 to 50K}\Omega \), plot the power dissipated by the load. Verify that the maximum power dissipation by the load occurs when RL is 10 KΩ.

![Resistive Circuit](image)

**Solution**

MATLAB Script

```matlab
% maximum power transfer
% vs is the supply voltage
% rs is the supply resistance
% rl is the load resistance
% vl is the voltage across the load
% pl is the power dissipated by the load vs = 10; rs = 10e3;
rl = 0:1e3:50e3;
```
k = length(rl); % components in vector rl

% Power dissipation calculation for i=1:k
pl(i) = ((vs/(rs+rl(i)))^2)*rl(i);
end

% Derivative of power is calculated using backward difference
dp = diff(pl)./diff(rl);

rld = rl(2:length(rl)); % length of rld is 1 less than that of rl

% Determination of critical points of derivative of power
prod = dp(1:length(dp) - 1).*dp(2:length(dp));
crit_pt = rld(find(prod < 0));

max_power = max(pl); % maximum power is calculated

% print out results
fprintf('Maximum power occurs at %8.2f Ohms
',crit_pt)
fprintf('Maximum power dissipation is %8.4f Watts
', max_power)

% Plot power versus load
plot(rl,pl,'+')
title('Power delivered to load') xlabel('load resistance in Ohms') ylabel('power in watts')

The results obtained from MATLAB are

Maximum power occurs at 10000.00 Ohms

Maximum power dissipation is  0.0025 Watts

The plot of the power dissipation obtained from MATLAB is shown in Figure 28.
Figure 28: Power delivered to load.