TECHSAR™

Advanced Guide to MATLAB

Practical Examples in Science and Engineering

S. N. Alam
S. Islam
S. K. Patel

DT8273954

dreamtech PRESS

Distributed by:
WILEY

ADVANCED GUIDE TO MATLAB

ADVANCED GUIDE TO

MATLAB

Practical Examples in Science and Engineering

Syed Nasimul Alam

Assistant Professor

Metallurgical and Materials Engineering
National Institute of Technology
Rourkela, Odisha, India

Sanjib Islam

Senior Architect

Upland Software
25 First Street
Cambridge, MA02141, USA

Saroj Kumar Patel

Associate Professor

Mechanical Engineering
National Institute of Technology
Rourkela, Odisha, India

©Copyright 2019 I.K. International Pvt. Ltd., New Delhi-110002.

This book may not be duplicated in any way without the express written consent of the publisher, except in the form of brief excerpts or quotations for the purposes of review. The information contained herein is for the personal use of the reader and may not be incorporated in any commercial programs, other books, databases, or any kind of software without written consent of the publisher. Making copies of this book or any portion for any purpose other than your own is a violation of copyright laws.

Limits of Liability/disclaimer of Warranty: The author and publisher have used their best efforts in preparing this book. The author make no representation or warranties with respect to the accuracy or completeness of the contents of this book, and specifically disclaim any implied warranties of merchantability or fitness of any particular purpose. There are no warranties which extend beyond the descriptions contained in this paragraph. No warranty may be created or extended by sales representatives or written sales materials. The accuracy and completeness of the information provided herein and the opinions stated herein are not guaranteed or warranted to produce any particulars results, and the advice and strategies contained herein may not be suitable for every individual. Neither Dreamtech Press nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Trademarks: All brand names and product names used in this book are trademarks, registered trademarks, or trade names of their respective holders. Dreamtech Press is not associated with any product or vendor mentioned in this book.

ISBN: 978-93-89520-29-3

Edition: 2019

Printed at: Rekha Printers

Dedicated

To
Erina, Anjum and Aadil
and
Sanmay

Preface

This book is on MATLAB, a programming language that has become very popular over the years. As the title suggests this book is an advanced level guide to MATLAB with a number of examples from science and engineering. There are a number of solved problems from all areas of science and engineering to give enough practice to the reader. We decided to write this book when we found that MATLAB is a language that is extremely useful today. MATLAB integrates computation, visualization and programming in a very user-friendly and easy-to-use environment. It is a high performance computer language used for technical computing. This book is mainly aimed at those who already know the language and would like to use MATLAB to solve advanced engineering problems. Any professional involved in using MATLAB for problem solving can refer this book. This book covers a wide range of topics. MATLAB is today a part of the curriculum in most institutions and universities. It has become very important and useful software and knowing how to use it can be extremely beneficial.

The past decade has seen a phenomenal rise in the use of MATLAB to solve and understand science and engineering problems. MATLAB by MathWorks provides a computational software environment for use in engineering, mathematics, statistics, science and finance. MATLAB frequently releases updates and includes new features. Here in this book we have tried to illustrate the advanced features of MATLAB. In an earlier book on MATLAB for beginners, author S.N. Alam has already discussed the basic features of MATLAB. This book covers a wide range of topics.

This book intends to present all the advanced features of MATLAB allowing the reader to understand the true potential of MATLAB. We would request the reader to solve all the examples in order to understand the true potential of MATLAB and this would give good practice in writing MATLAB programs. The reader should also practice the several functions available in MATLAB and try to write programs using these functions. This would acquaint the reader with a number of inbuilt functions and features of MATLAB. The text incorporates a large number of examples and illustrations. The twenty-five solved problems included in chapter 10 are expected to give enough practice to the reader.

Any feedback and suggestion for improving the book is welcome. We hope this book will help you learn and master MATLAB in a very short time!

Syed Nasimul Alam
Sanjib Islam
Saroj Kumar Patel

Acknowledgements

We gratefully acknowledge the help and support of our colleagues and students. We are indebted to the Metallurgical and Materials Engineering and Mechanical Engineering departments, NIT-Rourkela for providing all the support and inspiration for taking up this challenge.

We would like to especially thank all our family members for the support they showed and the inspiration and hope they gave without which completion of this work would not have been possible.

We are indebted our teachers for introducing computer languages which are still etched in our minds and always attract us to the programming languages.

This book covers a wide range of topics and it is a pleasure to acknowledge the help of colleagues, friends and family who provided numerous help in several areas. The authors welcome any suggestions from readers that could be incorporated in the future edition of this book.

Syed Nasimul Alam
Sanjib Islam
Saroj Kumar Patel

Contents

Preface *vii*
Acknowledgements *ix*

1. History of Computers, Computer Languages and MATLAB **1**

1.1 The Beginning of Computing 1
1.2 Random Access Memory (RAM) 7
1.3 The Microprocessor 7
1.4 Motherboard 9
1.5 Server 9
1.6 Algorithm 10
1.7 Kernel 10
1.8 Programming Languages 11
1.9 High-Level Language and Assembly Language 12
1.10 Simple Terms Related to Computers 13
1.11 Steps to Execute a Program 16
1.12 History of MATLAB 17
1.13 The Script Files 21
1.14 Help 22
1.15 Storage Duration of a Variable 23
1.16 Fun with MATLAB and Built-in Demos 23
1.17 Few Useful Commands Features 29

2. The Compiler **35**

2.1 What is a Compiler? 35
2.2 How Does a Compiler Work? 37
2.3 Interpreter and Compiler 38
2.4 The MATLAB Compiler 38

3. Numbers, Review of Matrices and Matrix Operations **40**

3.1 Arrays and Matrices 40
3.2 Array Operations in MATLAB 41
3.3 Integer Representation in Various Formats 44
3.4 Defining a Variable in MATLAB 49
3.5 Real Number 50

3.6 Absolute Value 50
3.7 ASCII, EBCDIC 51
3.8 Infinity (∞) 53
3.9 Imaginary Number 53
3.10 Mantissa 54
3.11 eps 55
3.12 Hadamard Matrix 55
3.13 Upper and Lower Triangular Matrix 57
3.14 pi 60
3.15 meshgrid 61
3.16 Convert Decimal to Binary and Binary to Decimal, dec2bin, de2bi and bin2dec 64
3.17 Ones' Compliment 66
3.18 Logical Operators &, |, ~ and xor 67
3.19 Relational Operators 68
3.20 The ':' Operator 69
3.21 Use of Temporary Variable for Sorting an Array 70
3.22 Product of Two Matrices 71
3.23 Determinant, Finding Determinant of a Square Matrix 74
3.24 Transpose of a Matrix 77
3.25 Eigenvalue and Eigenvector 79
3.26 Palindromic Number, Reversing a Number, Palindrome 81
3.27 Identity Matrix 82
3.28 Sparse Matrix 83
3.29 Rotation Matrix 85
3.30 Translation Matrix 87
3.31 trace 87
3.32 fliplr 88

4. Plotting Using MATLAB 90

4.1 Plotting a Line 90
4.2 Colors in MATLAB 95
4.3 ginput 96
4.4 plot and plot3 98
4.5 figure 99
4.6 Scatter Plot 100
4.7 colordef 101
4.8 Mesh Plot 102
4.9 surf and surfc Plots 104
4.10 The Barnsley's Fern Problem 106
4.11 A Cone 109
4..12 A Paraboloid 110
4.13 Contour Plots 111

4.14 A Plane 113
4.15 Cube 117

5. Image Processing Using MATLAB 119

5.1 Digital Image 119
5.2 image (c) 129
5.3 Create a Shadow to a Plot 130
5.4 Children 130
5.5 Shading 132
5.6 FaceAlpha 132
5.7 Giving Color and Transparency to Edges of a Figure 135
5.8 Types of Axis 137
5.9 Lighting and Material 139
5.10 View 150
5.11 Creating Plots Using Markers Filled with Colors and Having Varying Sizes 154
5.12 Rotation of an Image 156
5.13 pcolor 158
5.14 Reading an Image From an URL 160
5.15 Changing the Background Color Using Figure('color',c) 161
5.16 Convert a 8-bit RGB Image to a Grayscale Image 162
5.17 camup 164
5.18 rotate 167
5.19 Colormaps 169
5.20 RGB Image 172
5.21 camlight 174
5.22 Insert Ellipse, Annotation or Double Arrow in a Figure 176
5.23 Saving an Image with a New Filename 179
5.24 Obtaining Information About an Image File 181
5.25 Variation of Colors with the Variation in the Values in the Array 182
5.26 Add Line and Text to an Image 187
5.27 Buckyball 192

6. Animation, Sound and Movies Using MATLAB 197

6.1 Animation 197
6.2 drawnow 201
6.3 handle 203
6.4 Making Movies with MATLAB 204
6.5 getframe 205
6.6 Sound 206
6.7 Audio Data 207

7. Numerical Methods Using MATLAB 210

7.1 Numerical Integration 210
7.2 System of Linear Equations 216
7.3 System of Non-Linear Equations 223
7.4 Solving System of Equations Using MATLAB Function fsolve 225
7.5 Interpolation 227
7.6 Straight Line Fit Using Least Square Method 230
7.7 Curve Fitting Using Built-In Functions polyval and polyfit 235
7.8 Cubic Fit Using Least Square Method 237
7.9 Finding Roots of a Polynomial 239
7.10 Solution to Ordinary Differential Equations (ODEs) 242

8. Unconstrained Single Variable Optimization Methods 249

8.1 Bracketing Methods 249
8.2 Region Elimination Methods 259
8.3 Gradient-based Methods 272

9. Statistical Functions 285

9.1 Statistics 285
9.2 Mean, Median, Mode and Range 285
9.3 Standard Deviation, Variance 291
9.4 erf (x), Error Function 293
9.5 erfc(x), Complimentary Error Function 293
9.6 Normal Distribution 293
9.7 Normalization 298
9.8 Representation of Union, Intersection, Exclusive or and Subtraction 301
9.9 Pie Chart 305
9.10 Bar Chart 307
9.11 Histogram 311
9.12 boxplot 312
9.13 Random Numbers 313

10. Solved Problems 317

11. Dictionary of Additional Terms 358

Index 403

1

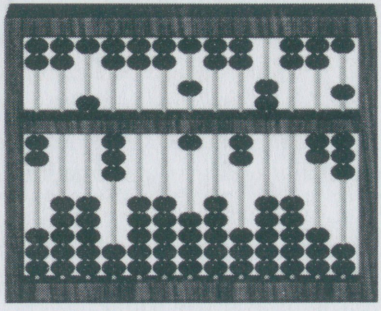
CHAPTER

History of Computers, Computer Languages and MATLAB

"Made weak by time and fate, but strong in will
To strive, to seek, to find and not to yield."
— Lord Alfred Tennyson
(Ulysses)

1.1 THE BEGINNING OF COMPUTING

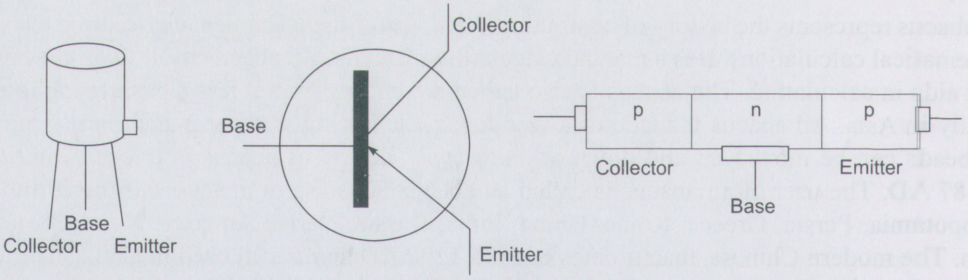
The abacus represents the history of computing. An abacus is a mechanical device used to assist in mathematical calculations. It is a manual calculating machine. An abacus itself does not calculate but it aids in calculating. The abacus is also called a counting frame. It is a calculating tool used initially in Asia. An abacus is made of a wooden frame having wires in which beads can slide. The beads can be moved up and down or sideways. The use of abacus is traced centuries ago in 1387 AD. The user of an abacus is called an abacist. The use of abacus was traced in Egypt, Mesopotamia, Persia, Greece, Rome, China, India, Russia, Native America, Russia, Korea and Japan. The modern Chinese abacus dates back to 1200 AD and is still used in China. The abacus used by the Aztec people of central Mexico had maize beads threaded through strings that were held by a wooden frame. The Aztecs are known to have used the abacus since 300 BC.



Abacus

The journey from abacus to computers has been very long and definitely very winding. One of the persons who provided the start and showed the direction is Charles Babbage. Charles Babbage (26th December 1791–18th October 1871) was a mathematician, philosopher and also a mechanical engineer. He is the creator of programmable computer. He is considered the father of computing. He invented the first mechanical computer. The original machine created by Babbage is now in the London Science Museum. Babbage on June 14th 1822 presented his Difference Enginein a paper entitled "Note on the application of machinery to the computation of astronomical and mathematical tables" to the Royal Astronomical Society. His machine called the difference engine or the difference machine was the prototype of modern automatic computing methods and also the prototype of government computing projects. The machine was able to calculate polynomials by using a numerical method known as the Differences Method. In 1823 the Society approved the idea presented by Charles Babbage and the government granted him £1500 to build the machined he designed.

A device that reduced the size of the early computer is the transistor. A transistor is a very important component in all modern electronic devices including computers. It is one on the greatest inventions of the 20th century. It is a low cost but is a very important component of all modern electronic devices. It was first invented in the Bell Labs and was named an IEEE milestone in 2009. The first transistor was invented by American physicists William Shockley, John Bardeen and Walter Brattain on 16th December 1947. The transistor made possible the creation of integrated circuits and microprocessors which are the most important components of today's electronics devices. The microprocessor is known to contain millions of microscopic transistors. The symbol of a transistor is shown below.

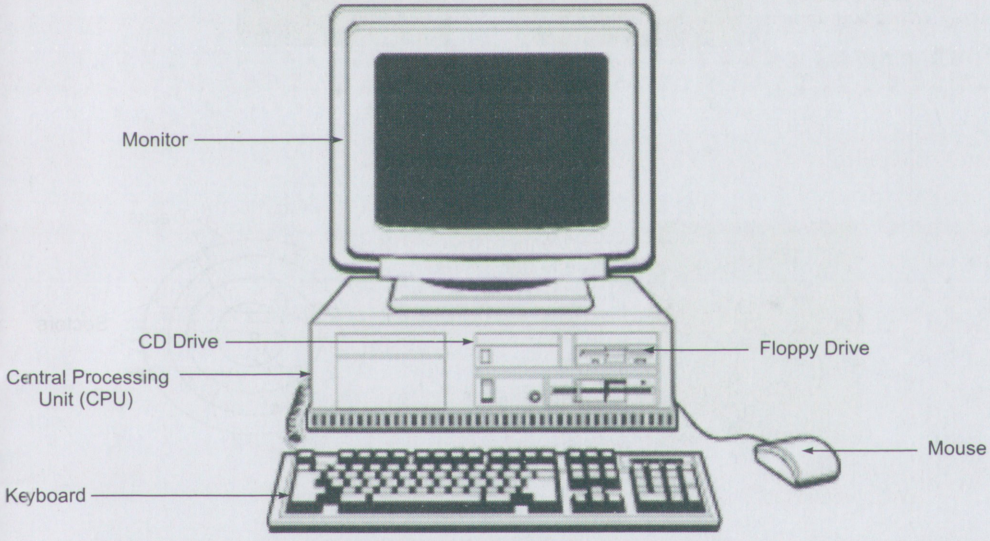


A transistor

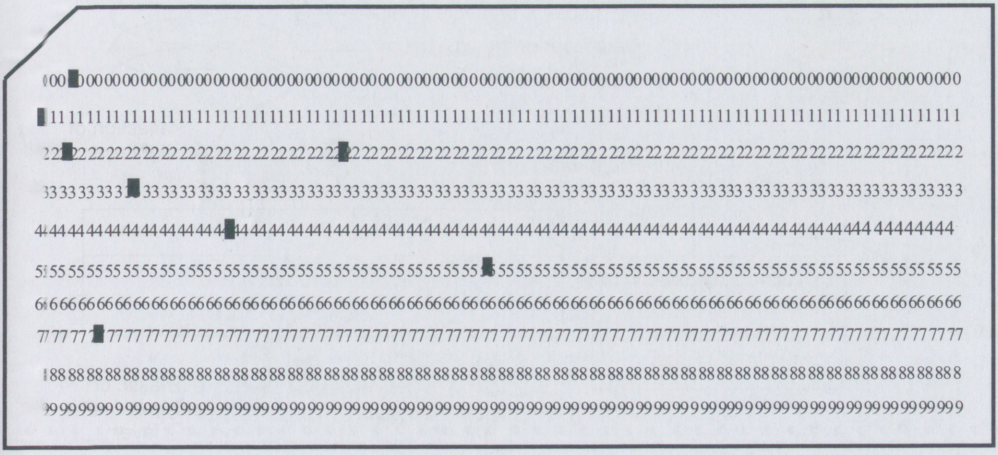
Before the invention of the transistor electronic devices consisted of vacuum tubes, which were large in size, required more energy to run and also dissipated a lot of heat. These devices were cumbersome and often failed to work. In fact without the invention of this device it would not be possible to have such efficient and small sized computers. It is the transistors which transformed the computer.

Data in the early days was stored in punched cards. A punched card also known as IBM card or Hollerith card is a card containing digital information. It is an early method by which data was stored. They are made of stiff paper having thickness same as that of greeting cards. They have holes punched in them in definite positions depending on the data. These cards have been used since 1725 in textile handlooms. In 1832 the punch cards were later used to store information by Semen Korsakov. Herman Hollerith (February 29th 1860 to November

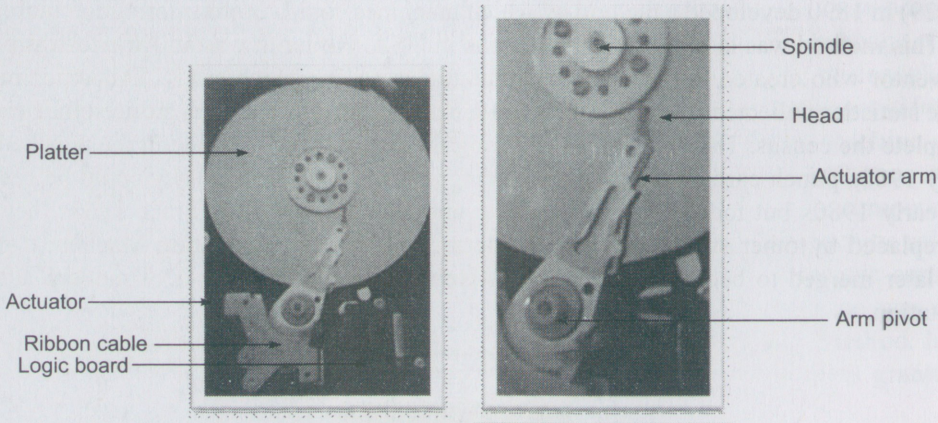
17th 1929) in 1890 developed a method by which machines could record information in the punch cards. This method was later used for the census in USA. Hollerith was an American statistician and inventor who created the mechanical tabulator using the punch cards. This machine could tabulate statistics collected from millions of data like the census data. This reduced the time taken to complete the census. The data collected during the census was collected in the punched cards. In early 1900s punch cards were used for storing and retrieving data. They could be seen even in the early 1980s but today they are obsolete and very hard to find. Since 1960s they began to be replaced by other storage devices. Hollerith founded the Tabulating Machine Company, which later merged to become IBM in 1924. He is known as the father of modern automatic computation.



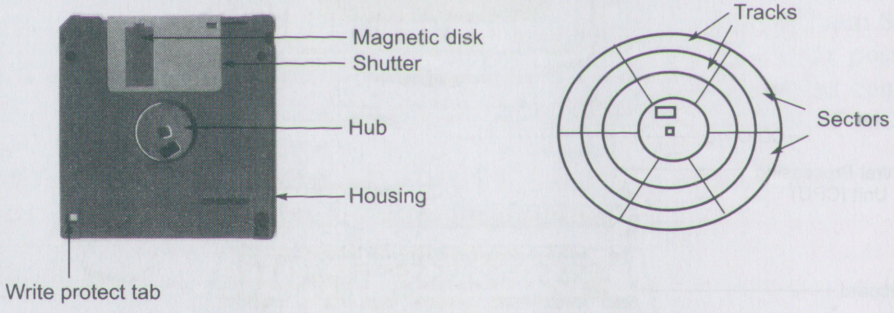
A schematic figure indicating the various parts of a personal computer



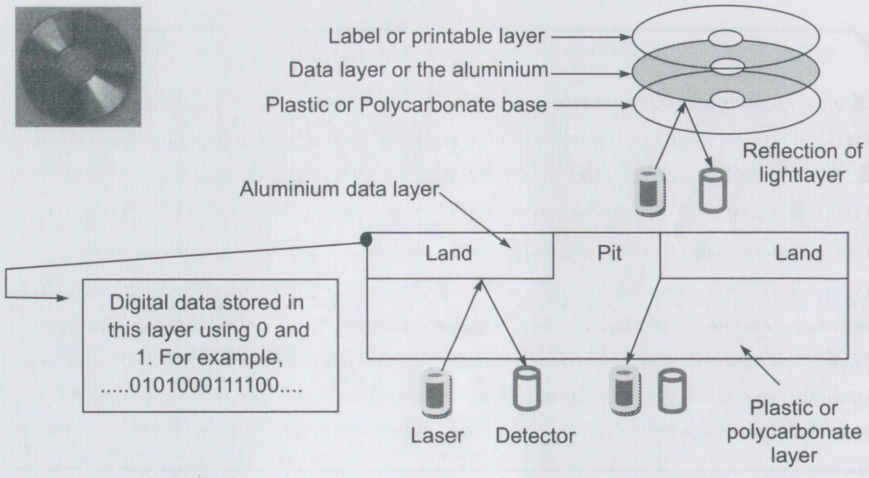
A punched card for storing data



The Hard Disk Drive

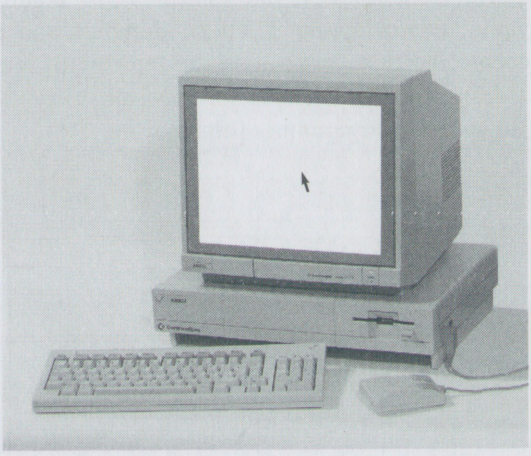


A 3.5" floppy disk



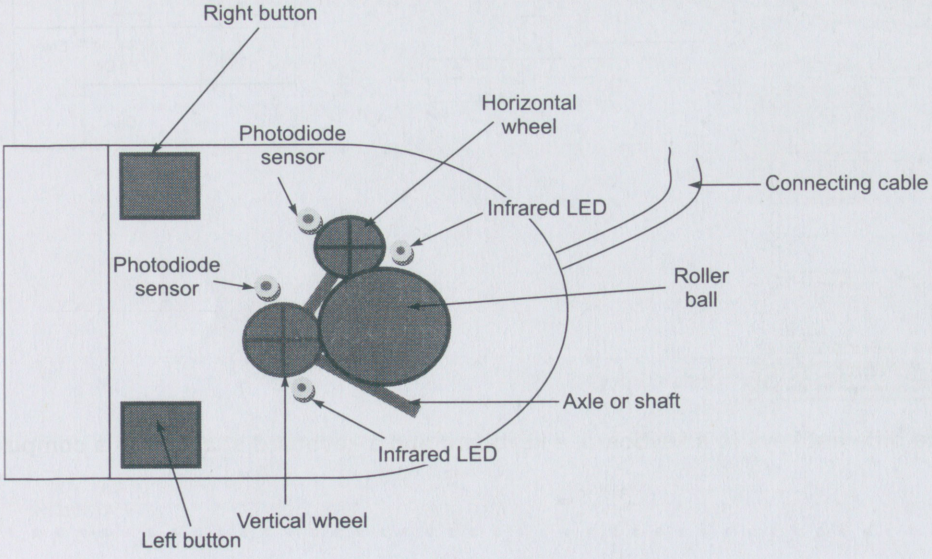
A CD ROM

In 1964, Douglas Engelbart invented the mouse. The first computer mouse prototype was made. This was used with a graphical user interface (GUI). The first mouse which was patented was made of a wooden shell with two metal wheels. He described it as an "X-Y position indicator for a display system. The reason why it was nicknamed the mouse was because tail came out the end'.



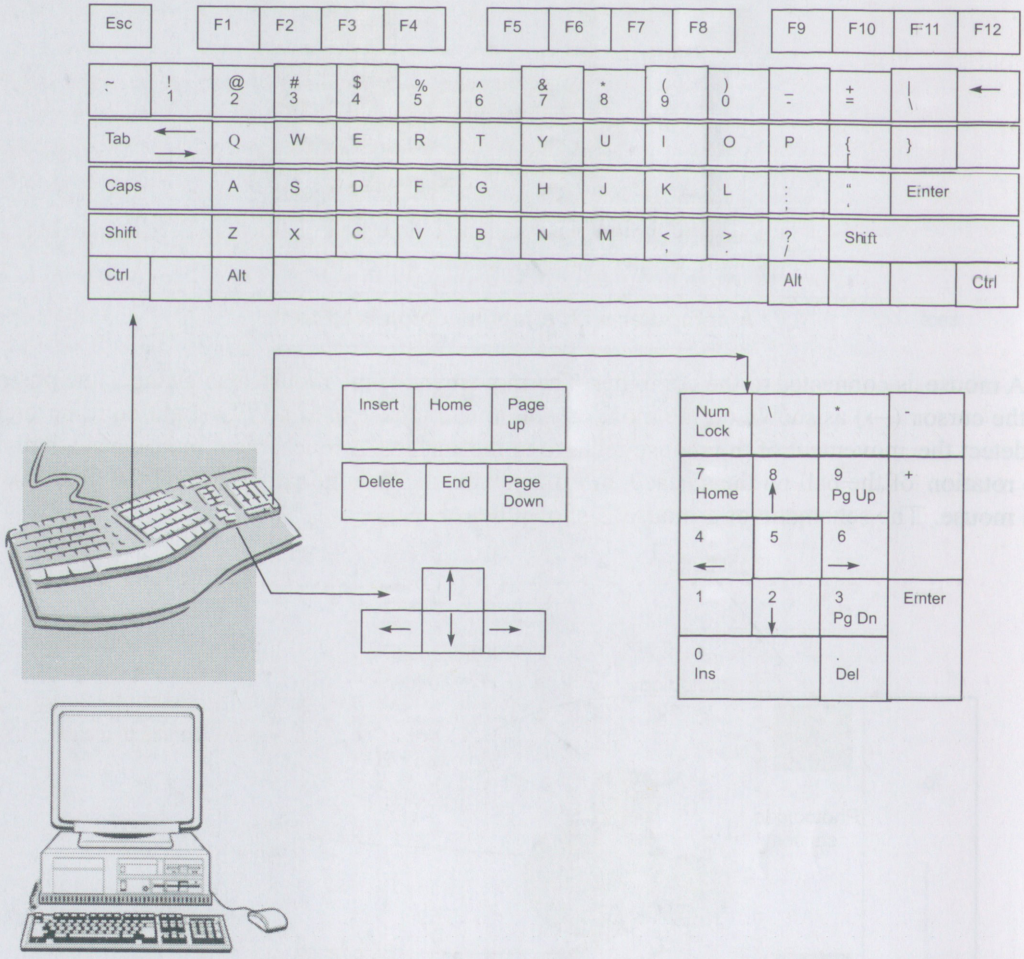
A computer with a mouse connected to it.

A mouse is connected to the computer. The movement of the mouse also changes the position of the cursor (→) as shown on the monitor. An optical mouse uses a LED (light emitting diode) to detect the movement of the mouse relative to the surface. A mechanical mouse has a ball and the rotation of the ball on the surface, in turn, rotates the orthogonal shafts and the gears inside the mouse. The schematic of a mouse is shown below.



The schematic diagram of a mouse

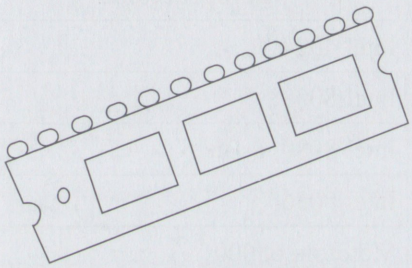
Keyboard is one of the primary input devices of a computer. Keyboards allow a computer use to input letters, numbers, and other symbols into a computer. A keyboard has a set of typewriter like keys that allows us to enter data into the computer. Computer keyboards are very similar t the electric-typewriter keyboards but they contain a few additional keys. The keys on compute keyboards are mainly of three types. alphanumeric (a, b, c, 1, 2, 3,...and so on) punctuatio (, , . , ; , and so on) and special (function keys like F1, F2 , F3..., control keys, arrow keys, Cap Lock key, and so on). The standard layout of letters, numbers and punctuation is known a a QWERTY keyboard because the first six keys on the top row of letters spell QWERTY. The QWERTY keyboard was designed in the 1800s for mechanical typewriters and was actuall designed to slow typists down to avoid jamming the keys.



The different keys in a keyboard, a keyboard and a keyboard attached to a computer

1.2 RANDOM ACCESS MEMORY (RAM)

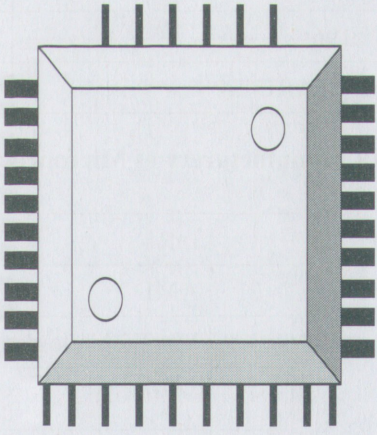
RAM or random access memory is a computer memory that can be accessed randomly. This implies that any byte of the RAM can be accessed without touching the preceding bytes whereas in a hard disk, CD, DVD or the primitive drum disks data could be read and written in a predetermined order. This is due to the difference in design of the two memories. The first RAM was developed in 1951. A RAM allows us to access data directly in any random order. RAM is a common type of memory found in computers and devices like the printer. There are two types of RAM, namely SRAM and DARAM. SRAM stands for static random access memory and DRAM stands for dynamic random access memory. SRAM is faster compared to DRAM because DRAM needs to be refreshed several times per second. SRAM does not need to be refreshed. Both the RAMs are volatile and they lose their data when the power is turned off. The technology used in the two RAMs is different. DRAM is more commonly used although it is slower compared to SRAM because it is cheaper.



A RAM

1.3 THE MICROPROCESSOR

A microprocessor is the CPU (central processing unit) of the computer. It is the most important device in the computer. It is a single integrated circuit (IC) or sometimes a few ICs. It is a programmable device. Digital data can be input into the microprocessor which are later processed as per instructions given and finally provides an output. It performs by using digital logic. Microprocessors perform several functions like computing, editing and also graphics and multimedia. It is fabricated on a single chip. The first microprocessor was the Intel 4004 which was built in 1971. The first microprocessor that was built on a single chip was not very powerful and it was capable of only addition and subtraction and could process only 4 bits at a time. The microprocessor reduced the size of computer dramatically and it was now possible to build portable calculators. Microprocessors vary based on the instruction it can process, the number of bits it can process at a time or the bandwidth and the clock speed or the number of instructions it can process per second. The clock speed has a unit of MHz. A microprocessor could be a 64-bit microprocessor and its speed could be 50 MHz. Microprocessors are built by several companies like Intel, Oracle.



A schematic figure of a microprocessor

1.3.1 The History of Microprocessors

Microprocessor	Year	Clock speed
Intel 4004	1971	1080 KHz
Intel 8008	1972	0.2–0.8 MHz
Intel 8080, 8-Bit	1974	2 MHz
Intel 8086	1978	4.77, 8 and 10 MHz
Motorola 68000	1979	5–8 MHz, Lisa had a clock speed of 5 MHz whereas Macintosh SE FDHD had a clock speed of 8 MHz
Intel 80286	1982	12.5 MHz
Intel 80386 DX	1985	33 MHz
Intel 80486 DX	1989	25 MHz
Pentium Processor, Intel	1993	60, 66 MHz
Pentium Processor, Intel	1994	75 MHz
Pentium Processor, Intel	1994	90 and 100 MHz
IMB z196	2010	3.8 to 5.2 GHz
Oracle SPARC T4	2011	3 GHz

Top 10 Manufacturers of Microprocessors

1.	Intel	6.	Transmeta
2.	AMD	7.	MIPS
3.	Via Technologies	8.	Texas Instruments
4.	Motorola	9.	National Semiconductor
5.	DEC	10.	NEC

1.3.2 x86

x86 is the most predominant architecture of the microprocessor for personal computers. Intel processors released after 8086 processor are given the generic name x86. 8086 is a 16-bit processor which is the first in this series was introduced in 1978. This was introduced after the 8-bit 8080 microprocessor. The processors in the x86 series are 286, 386, 486, 586 and Core. x86 is actually the short for 80x86 which means 286 is actually 80286 and 586 is actually 80586

and it x86 actually stands for 80x86. 80 is redundant and not included. Intel 586 processor is called Pentium processor. The Pentium processor released in 1993 used the fifth generation microarchitecture P5 and is above the Atom and Celeron processor but below the more advanced multi-core processors. It has a CPU clock speed of 60-3.8 GHz. Multi-core processor has two or more independent CPUs or cores. Core 2 introduced in 2006 has a clock speed of 1.06 to 3.5 GHz. The new processors in the x86 series can run all programs that the old processor could run. x86 is a microprocessor architecture that is used to build all the microprocessors in this series. Intel and AMD are the major manufacturers of the x86 processors.

Processor	Bits	Bus Size (Bits)	CPU clock speed
Core 2 Duo	64	64	1.8–2.6 GHz
586, Pentium	32	64	60–200 MHz
486DX	32	32	25–100 MHz
386DX	32	32	16–40 MHz
286	16	16	6–12 MHz
8086	16	8	5–10 MHz

1.4 MOTHERBOARD

The motherboard or the mainboard is the main circuit board of a computer. The motherboard is the unit that integrates all the components in a computer. The motherboard contains the CPU or the processor, card slots for attachments where memory cards can be attached, BIOS (basic input/output system also known as system BIOS, checks devices to ensure that they function properly and starts them), memory, mass storage interfaces, serial and parallel ports, expansion slots. Initially, motherboards used to hold a few components but today the entire board looks very crowded and filled with components. All the controllers that control the devices like monitor, keyword, mouse, floppy drives, disk drives, sound and graphics are attached to the motherboard. The first motherboard was developed by Intel and only had the processor and slots for attaching cards in it.

1.5 SERVER

A server is a mainframe computer that serves other computers attached to it. It is a dedicated computer system that performs only server tasks. A server is connected to several computers or a computer network and provides help and assists the computers in the network. Servers operate in a client-server architecture. This why it is possibly called the server as it serves the other computers in the network. There may be more than one server also. Even the websites and email services provided by the website also use the client-server network model. The various types of server are database server which processes database queries, file server which is a storage device dedicated to storing files, the mail server serves as an electronic post office for email,

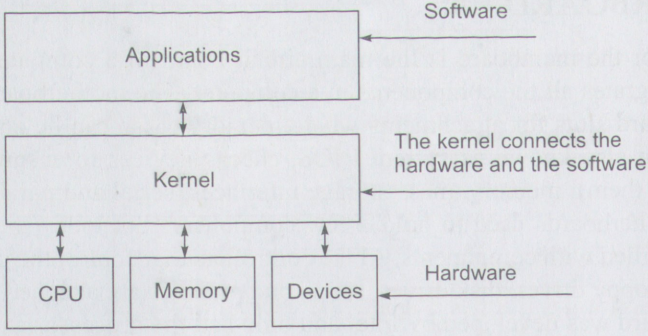
application server also known as the appserver handles all application operations between user and an organization's databases, a print server which manages one or more printers etc.

1.6 ALGORITHM

An algorithm provides a procedure for solving a problem. In order to do anything on compute one has to write a program. A computer program is written which tells the computer step by step what has to be done. Once the computer program is executed an outcome is achieved. An algorithm helps us to develop the program for the computer in order to achieve the objective. An algorithm is a set of instructions that need to be carried out to achieve an objective. The instructions in an algorithm starts the process and finally ends the process once the objective is achieved. The dictionary meaning of algorithm is a step-by-step problem-solving procedure especially an established, recursive computational procedure for solving a problem in a finite number of steps. Algorithms are actually like roadmaps for doing a task.

1.7 KERNEL

Kernel is the main component of the operating system which acts as a bridge between the applications and the data processing which is done in the hardware. The kernel communicates between the hardware and the software. Different kernels operate differently. Their operation depends on their design.

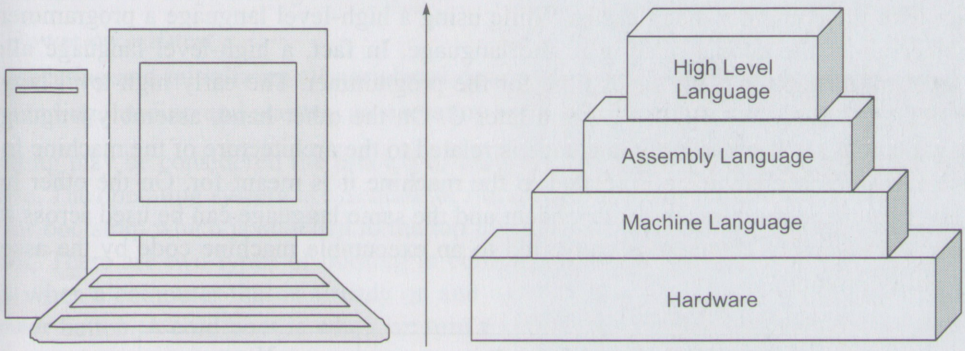


The kernel is a program which is the central module of the computer operating system. It is the core of the operating system. It is stored in the main memory and controls everything that occurs in the system. The kernel takes care of the management of memory, process and disk. The kernel does not directly interact with the programmer. The kernel interacts with the shell and other programs and also the hardware of the computer like the processor, memory and disk. It is the kernel which loads into the memory first during booting and remains there till the computer is turned off. As the kernel resides in the main memory its size is kept as small as possible. The kernel cannot be overwritten and it is stored in a protected area in the memory. The crashing of a computer is synonymous with the crashing of the kernel.

1.8 PROGRAMMING LANGUAGES

The language used in communicating with a computer is known as programming language. Programming languages can be classified into three categories:

(i) Machine language
(ii) Assembly language
(iii) Procedure-oriented language



Machine language is the basic lowest level programming language. It is the only language that can be comprehended by a computer. A program in the machine language consists only of sequence of a large number of binary numbers called numerical codes. It is easily understood by the computer hardware but it is very difficult to understand by the user. It consists only of numbers 0 and 1. Programming in machine language is very tedious and cumbersome process. To help reduce the difficulty of the programmer assembly languages were introduced in the 1950s. The assembly languages use mnemonic codes and addresses. Now the user programs uses names instead of numbers. Alphanumeric symbols are used in the program instead of numeric operation codes and addresses. Writing programs in assembly languages were still be found to be a cumbersome job so high level languages were introduced as a language that is between human language and machine language. A high level language consists of a set of words and symbols which the programmer can use to write a program. It is more similar to the language we communicate in. These "close to the English language" high level programming languages are more human-centered than machine-centered. They are more user-friendly than the machine language. They are known as the user-oriented or procedure-oriented languages. A few high level languages are C, FORTRAN, PASCAL, COBOL, ALGOL, etc. Programs in high-level language are converted to assembly language or machine language using a compiler. An assembly language is converted to a machine language using an assembler. High level languages offer a number of advantages over the machine language or the assembly language:

(i) Simplicity
(ii) Uniformity
(iii) Machine independence
(iv) Diagnostic error delectability

1.9 HIGH-LEVEL LANGUAGE AND ASSEMBLY LANGUAGE

A high-level language is a programming language uses natural language elements which make i easy to use. In computer science, a high-level programming language is a programming languag which may use natural language elements which makes it easier for us to use. In other word it is far from the machine language and close to a language the user can understand. This i why it is easier to understand compared to a lower-level language. This also makes the proces of programming easier and simpler. The aim of high-level language is to help the programme interact with the computer much easily. While using a high-level language a programmer nee not bother about the details relating to the language. In fact, a high-level language allows high level of abstraction. This saves time for the programmer. The early high-level language are FORTRAN, Basic, COBOL, PL/I and later C. On the other hand, assembly language is low-level language. The assembly language is related to the architecture of the machine in othe words each assembly language is related to the machine it is meant for. On the other hand, high level language is not machine dependent and the same language can be used across severa systems. An assembly language is converted to an executable machine code by the assemble or the utility program.

High-Level Language	Year
FORTRAN	1954
Lisp	1958
COBOL	1959
Simula	1964
Basic	1964
Smalltalk	1969
Prolog	1970
Pascal	1970
C	1971
ML	1973
Scheme	1975
Ada	1979
C++	1983
Perl	1987
Python	1991
Java	1995
C#	2000
VB.NET	2001

1.10 SIMPLE TERMS RELATED TO COMPUTERS

1.10.1 What is Booting?

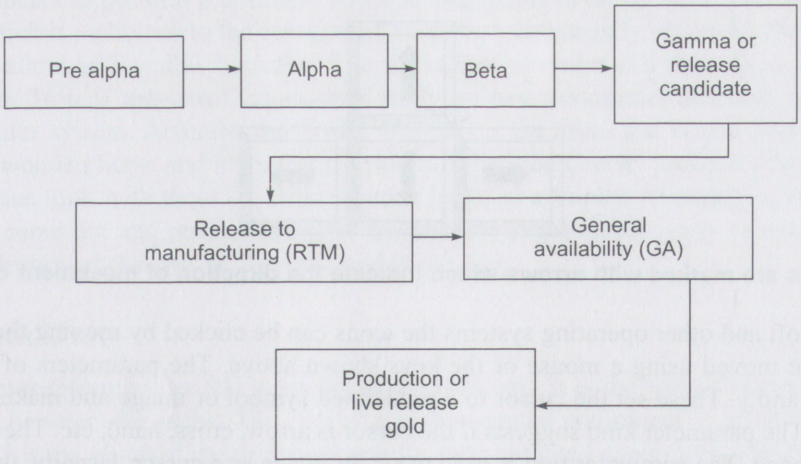
Boot is to start the computer. Booting or booting up is the set of operations that a computer performs when the power to the computer is turned on. This process happens whenever the computer is turned on. After the booting is over the computer is ready to do all normal operations. The booting up process does not take much time and the computer becomes ready to use within a few minutes. The steps during booting are:

(i) Power on self-test
(ii) Locating and initializing all peripheral devices
(iii) Finding, loading and starting an operating system.

Booting may be considered the process of loading of the operating system, which is the first software. The operating system is essential for running any program on the computer. Boot is the short for bootstrap which is attached to the top of the boot and had to be pulled in order to wear the boot. There are two types of booting in computers, a warm boot and a cold boot. A warm boot is when a computer that is already on and working is reset. This can be done by pressing the power button. A cold boot is when we turn a computer on from an off position.

1.10.2 Golden Master

Golden master is like the master copy of a final version of the software that is used to make retail copies for distribution. The release of a software also has several phases. It starts with initial development and ends with the release. It also has updated versions of the earlier released version.

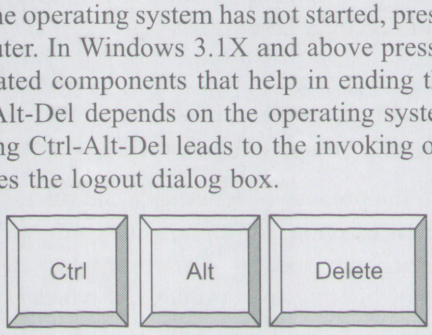


Different stages in the release of software

The term RTM or relapse to manufacturing is also known as going gold. This term is used when the software is ready to be delivered to the customer. The software is generally digitally signed so that the customer can verify its authenticity. A copy of the RTM is known as the "gold master" which is sent for duplication and distribution to the market.

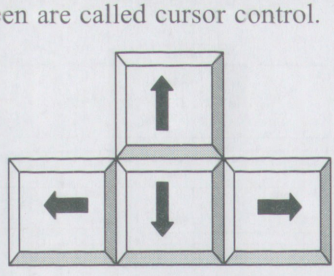
1.10.3 Control-Alt-Delete or Ctrl-Alt-Del

Control-Alt-Delete or Ctrl-Alt-Del is also known as the three figure salute. Ctrl-Alt-Del use outside the computer terminology, in day to day life, means to do away with or get rid o Ctrl-Alt-Del is a computer command used in IBM PC compatible computers. All the three key Ctrl, Alt and Delete have to be pressed together. The purpose of pressing the three keys togethe is to terminate or interrupt a function or facilitate the interruption and termination of a function In pre-boot condition when the operating system has not started, pressing the Ctrl-Alt-Del button restarts or reboots the computer. In Windows 3.1X and above pressing Ctrl-Alt-Del invokes th task manager or security related components that help in ending the current Windows session The result of pressing Ctrl-Alt-Del depends on the operating system and version of Windows For example, in OS/2 pressing Ctrl-Alt-Del leads to the invoking of the session manager and i X Windows System it invokes the logout dialog box.



1.10.4 Cursor

The cursor is a symbol on screen that indicates the position where the next character that wil be entered is going to be displayed. The cursor usually blinks and is positioned where the nex character can be entered. When a C program is run the command window shows a blinkin cursor (|) which is a blinking black vertical line in a white background. The keyboard keys use to position the cursor on the screen are called cursor control.



These keys are marked with arrows which indicate the direction of movement of a cursor

In Microsoft and other operating systems the icons can be clicked by moving the cursor. Th cursor can be moved using a mouse or the keys shown above. The parameters of a cursor ar kind, img, x and y. These set the cursor to a predefined symbol or image and makes it visible i it is hidden. The parameter kind suggests if the cursor is arrow, cross, hand, etc. The cursor coul also be an image. The parameter img is used to set an image as a cursor. Usually, the image siz is very small (16×16 or 32×32 pixels size). x and y are the coordinates of the active spot o the cursor and x and y must be less than the dimensions of the image.

1.10.5 Hibernate

Hibernate mode is a power management mode which conserves power. When hibernate mod is selected the computer saves the current state from the RAM to the hard disk and then shut

down the computer. When we restart the computer the typical boot sequence is avoided and the information saved in the hard disk is used to restore the settings in which the computer went into hibernation. The programs that had been running during the previous session before going to hibernation are loaded. Hibernate mode is similar to sleep mode but in sleep mode or standby mode the power cannot be turned off. This mode is useful for laptops where power conservation is very important.

1.10.6 Syntax

Syntax is an ancient Greek word meaning arrangement, together or an ordering. In English language syntax is the way in which words are put together to form phrases, clauses or sentences are formed. From linguistics point of view, it is the study of the principles, processes and rules that govern the formation of grammatical sentences in a particular language using words.

From computers point of view, in programming language the term syntax refers to the rules and structural patterns that govern the use of appropriate words and symbols, in the particular computer language, for issuing commands or writing codes. Syntax is also used to refer to the rules that govern the behaviour of mathematical systems.

1.10.7 Trojan

A Trojan is a dangerous virus that is loaded in a computer through a host program. It is named after the wooden horse with which the Greeks used to infiltrate Troy. A Trojan or sometimes known as Trojan Horse unlike the worm is a non-self-replicating type of virus or malware. Trojans spread when the user interacts with it. Opening and downloading a file from the Internet or opening an e-mail attachment from the Internet can cause Trojans to spread in a computer system. It appears to perform a desirable function and seems to do no harm but instead it drops a payload which is malicious to the computer. Computer users usually get tricked by the Trojans. It may even allow an invisible backdoor into the target computer and thus allow illegal access to the system. Trojans may steal information from the target computer and may cause damage to the computer system. Actually, the Trojan Horse is a tale from the Trojan War. The Greeks built a huge wooden horse and hid a force inside it. When the Greeks pretended to have lost the war the Trojans took with them the large wooden horse as a trophy. At night the Greeks hiding in the horse came out and opened the gates for the rest of the Greek army to enter the city of Troy. They destroyed the city of Troy and ended the war.

1.10.8 Rack Unit

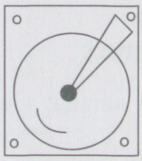
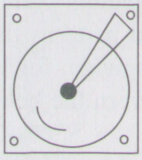
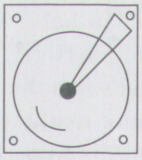
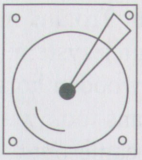
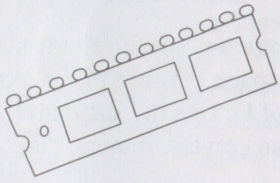
U or sometimes referred to as RU is known as the rack unit. It is the unit of measure that refers to the space between the racks. The size of a rack mounted equipment is often referred to in terms of U. U is the standard measurement unit for rack-mounted equipment. The racks are used to house servers, routers, hard drives, audio and video equipment or other hardware.

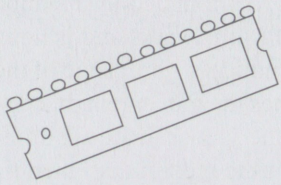
$$1U = 1.75 \text{ inches or } 44.45 \text{ mm}$$

1U is one unit of rack space. Usually, the racks are 24 inches wide but the height is variable. A rack of size 10 U would mean (10 × 1.75") 17.5" vertical usable space. The equipment that fit into them are also measured in terms of U."

1.11 STEPS TO EXECUTE A PROGRAM

A program undergoes six steps before being executed. They are explained in the following Tabl

	Step	Place where the step is executed
1.	*Edit:* The program is developed using the editor and stored in the hard disk. The programmer types the program using the editor and makes necessary corrections and stores the program in the hard disk. The programs are stored with .m extension in case of MATLAB script file.	 Hard Disk
2.	*Preprocessor:* The preprocessor processes the program. The preprocessor executes the preprocessor directives, which perform some manipulations on the program before it compiles.	 Hard Disk
3.	*Compile:* The compiler creates an object code and stores it in the hard disk. The compiler translates the program into the machine language code or the object code.	 Hard Disk
4.	*Link:* The likner links the object code with the libraries and stores it in the hard disk. A program contains references to several functions which have been defined in standard libraries or other libraries developed for a particular problem. A linker links the object code with the code of the missing function so that the program becomes executable.	 Hard Disk
5.	*Loader:* The next event is loading. The loader puts the program in the memory. The loader loads the program in the memory. The executable form of the program is loaded in the memory.	 Memory

6.	*Execute:* Finally, the CPU executes each the program. All the instructions given in the program is executed by the CPU.	 Memory

1.12 HISTORY OF MATLAB

Cleve Barry Moler, a mathematician and professor of computer science developed MATLAB. The term MATLAB comes from "Matrix Laboratory". Moler was a professor of mathematics and computer science for almost 20 years at several universities in the United States like the University of Michigan, the Stanford University and the University of New Mexico. MATLAB started to gain popularity and became a very useful tool for both teaching and research. Initially MATLAB was free. Jack Little, an engineer, came across MATLAB during a lecture by Cleve Moler at Stanford University in 1983. He realized the commercial potential of MATLAB and rewrote MATLAB in C with Cleve Moler and Steve Bangert. They introduced m-files which enable to write programs and store them. A number of library files were also added. In order to develop and market MATLAB they founded Math Wroks in 1984. Cleve Moler is currently the Chairman and Chief Scientist at Math Works.



The location of Math Works Headquarters in USA

Cleve Moler himself wrote about the Math Works logo in the Math Works website. This logo is present everywhere where Math Works is associated and has stood as a symbol of Math Works. It is even present in the sponsor a highway signboard on Route 9 in Natick, Massachusetts where Math Works is located. This logo is a L shaped membrane and has been used ever since the Math

Works was founded as a symbol which represented the company. It was later made the offici logo of Math Works and is now on every product of Math Works. Cleve Moler has described th mathematical significance of the logo. The wave equation is a model in mathematical physics tha describes the movement of a disturbance through matter. If t is considered time, x the x-coordinat and y the y-coordinate with units chosen such that the speed of propagation of the disturbanc through the medium is l then the partial differential equation, that is, a solution to the amplitud of the wave is given as,

$$\partial^2 u/\partial t^2 = \partial^2 u/\partial x^2 + \partial^2 u/\partial y^2$$

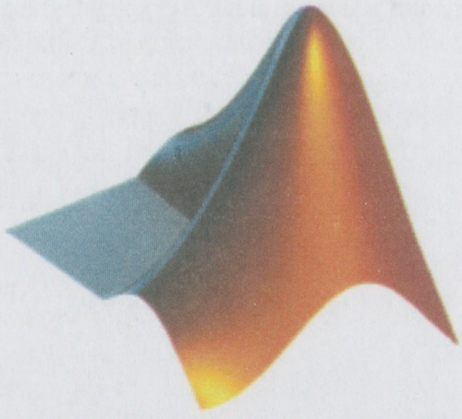
The solution of this equation is,

$$u(t, x, y) = \sin\left(\sqrt{\lambda t}\right) v(t, x)$$

and

$$\partial^2 v/\partial x^2 + \partial^2 v/\partial y^2 + \lambda v = 0$$

λ are the eigenvalues and the function $v(x, y)$ are the eigenfunctions of vibration. A solution t the wave equation can be expressed as a linear combination of the eigenfunctions. The abov logo can be obtained by just giving the command, logo.



The Math Works logo

MATLAB and Simulink are the two major products of Math Works. At present there are abou 1 million users of MATLAB and there are about 1500 books on MATLAB that are available i 28 different languages. This shows how widely MATLAB is used around the world. Toda MATLAB and Simulink are used widely in research and development. It has become a ver important tool. MATLAB is the programming language for technical computing and a programmin environment for algorithm development, data analysis, visualization and computation. Simulin on the other hand is a graphical environment for simulation and model-based design.

MATLAB or Matrix Laboratory has several advantages over the other computer languages. A the name suggests MATLAB's basic data element on the matrix. In MATLAB even as integer i

considered a matrix having a single row and a single column. MATLAB was initially developed to give easy access to the matrix software developed by LINPACK and EISPACK projects. These two software pieces are the most developed ones in matrix computation. MATLAB allows us to manage the variables in the workspace and also import and export data.

MATLAB is not just a programming language but it is also a programming environment. One can perform quick operations from the command line in the command window or write detailed programs in m-file and save it for modifying it later. The command line in the command window acts just like a usual calculator having enhanced features but it does not allow us to save the commands for modifying it later. For example, we can enter mathematical expressions in the command line and get results quickly without writing programs to find the result.

```
>> (10*2) + (5*2)
ans =
  30
>>
```

In order to run an m-file one can give the command,

```
>> myfirstprogram
```

Here myfirstprogram.m is the name of the file created by the programmer.

MATLAB has a vast library of mathematical functions ranging from elementary functions like sin, cos, abs, etc., to sophisticated functions like inv, eig, erf etc.

Over the years MATLAB has evolved. It has become a standard software used in several introductory and advanced leek courses. It is high-level language. It allows us to create small as well as large programs. In academic institutions it has become a standard. In industries it is the tool of choice for high-productivity research, development and analysis. MATLAB is used in the area of Maths and computation, modeling, simulation, algorithm development, data analysis and visualization of the output, graphics applications involving graphical user interface (GUI). MATLAB allows both two-dimensional and three-dimensional data visualization, image processing, animation and graphics. It includes tools for developing, managing, debugging and profiling m-files. MATLAB allows us to solve many technical computing problems very fast and with high precision.

MATLAB features number of solutions which are specific to an application. These are called toolboxes. Toolboxes allow us to apply specialized technology. Toolboxes are comprehensive collections of functions which are m-files. The MATLAB environment can be extended to solve particular classes of problems using toolboxes. A few areas in which toolboxes are available are signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, etc. The functionality of MATLAB is enhanced by the toolboxes. The toolboxes are a set of functions that provides specialized functions. For example, the statistics toolbox enables more specialized statistical manipulation. The MATLAB graphics system allows us to handle graphics. It has low-level commands that allow us to specify the look of the plot or the graphics. On the other hand MATLAB also enables us to build complete graphical user interfaces (GUI) on our MTALAB applications. The MATLAB Application Program Interface (API) is a library that allows us to write C language and Fortran language programs that can interact with MATLAB.

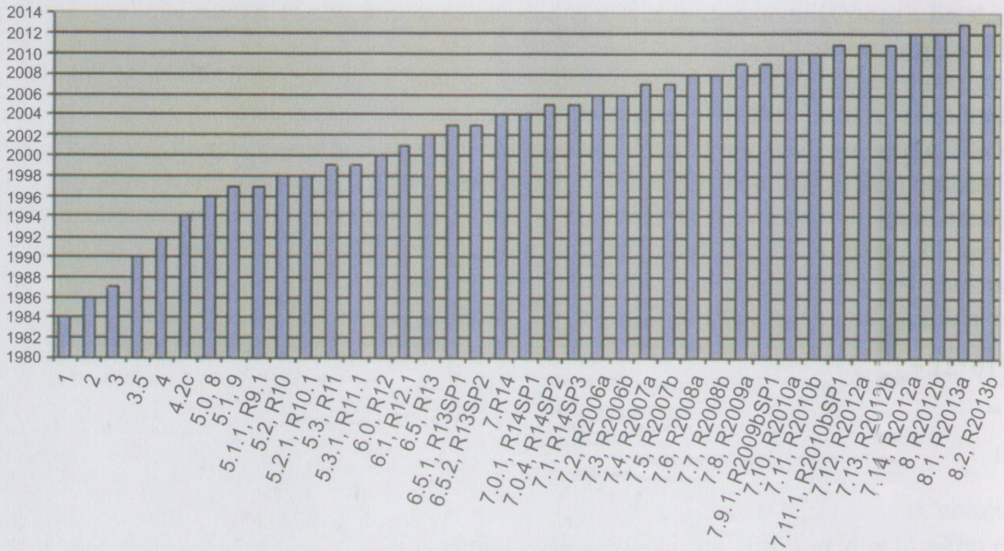


Chart showing the various versions and release names of MATLAB

The various versions and release names are listed in the chart above. For example, in 8.2 R2013b, 8.2 is the version and R2013b is the release name. The first version was released in 1984 and this was MATLAB 1.0. Currently, the version available is MATLAB version 8.2 and has a release name R2013b. This is also the version we are using for writing this book.



Advertisement of the R2013b. It was released on September 5th, 2013.

A few similarities and differences between MATLAB and C.

MATLAB	C
MATLAB is proprietary and we need to pay for MATLAB.	C compilers are available free of cost and are open source.

Case sensitive A is different from a	Case sensitive A is different from a

Variables need not be declared at all. They can be created inside the program and used in the program. For example,
k=10;
l=10.11;
	Variable type has to be declared before they can be used in the program. For example,
	int k, float b, char c;
	k=10;
	b=10.11;
	c='d';

There is no need to define the number of rows and columns while declaring the array. Auto dimension. a=[2 4 6;8 10 12];	The number of rows and columns has to be explicitly mentioned before declaring an array. float a[10]; float a[10][10]; int a[3][3]={{2,4,6},{8,10,12}} int a[2][2]={{2,3},{4,5}};
Array elements, A(2,3)	Array elements, A[2][3]
In MATLAB the printing of an array element is possible by, a(11)	A print command for printing an array element in C will be, printf("%d", a[11]);
In MATLAB it is possible to execute commands in the command window	This is not possible in C
% comment	/*comment*/

for i=1:4:100
...
end
	for(i=1;i<=100;++4)
	{
	...;
	}

MATLAB is used to do many things and it can very easily interface with C.

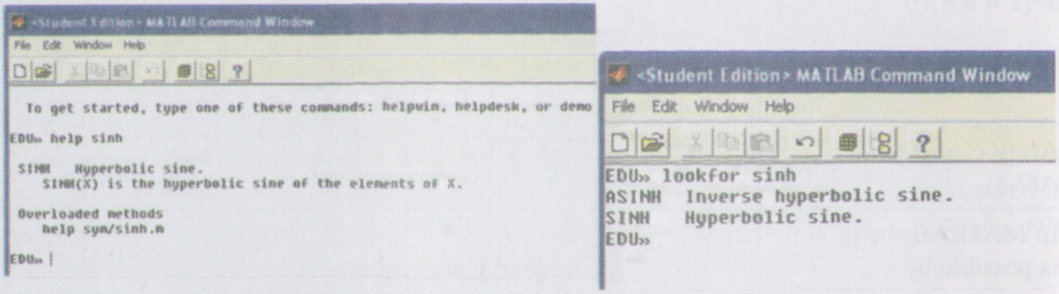
1.13 THE SCRIPT FILES

For problems which are short and simple the Command Window provides an easy way to enter the data and obtain results. One could consider it as a calculator where we enter data and get fast answers. On the other hand when the problem we are dealing with is complex and needs lot of time and decisions regarding the mathematical operations that need to be taken entering the commands in the MATLAB Command Window become difficult. For this MATLAB provides a solution by allowing us to write a text file containing all the commands in a proper sequence

and store the file in a folder. The sequence of commands in the file can be executed by calling the file name in the Command Window. Once the file is called MATLAB executes the commands exactly as it had been mentioned in the file. These files are called M-files since the file has been stored with a file name that ends with the extension .m. As the file is stored permanently it can be called again and again and run to give us the results. The file could be modified later. The MATLAB Editor/Debugger is used for writing and editing m-files.

1.14 HELP

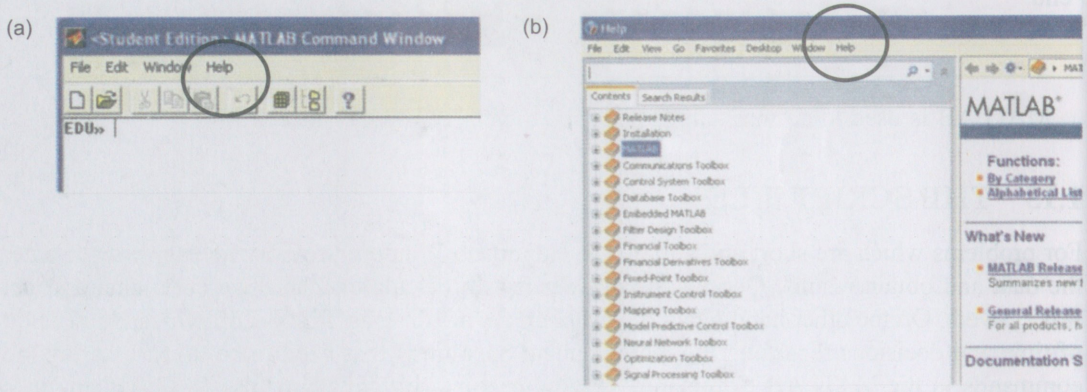
Help is one of the most important features of MATLAB. It is very comprehensive and detailed and any information about the various features of MATLAB can be obtained from it. One can learn MATLAB form help itself. The help command gives us information about a MATLAB function. One must know the name of the MATLAB function itself in order to find the information about the function.



Help and look for commands in MATLAB Command window

Here one must note that MATLAB distinguishes upper case and lower case characters in function names. So one must be careful in specifying the function names. If one knows the name of the function about which you wish to find the information the help command is very useful but in case the function name is not known then you could use the look for function. The look for command is used for searching all MATLAB files for the keywords that we provide.

For example, besides this one could also use the Help option form the pull-down menu.



Help in (a) MATLAB Student edition (b) MATLAB Version 7.11.0.584 (R2010b)

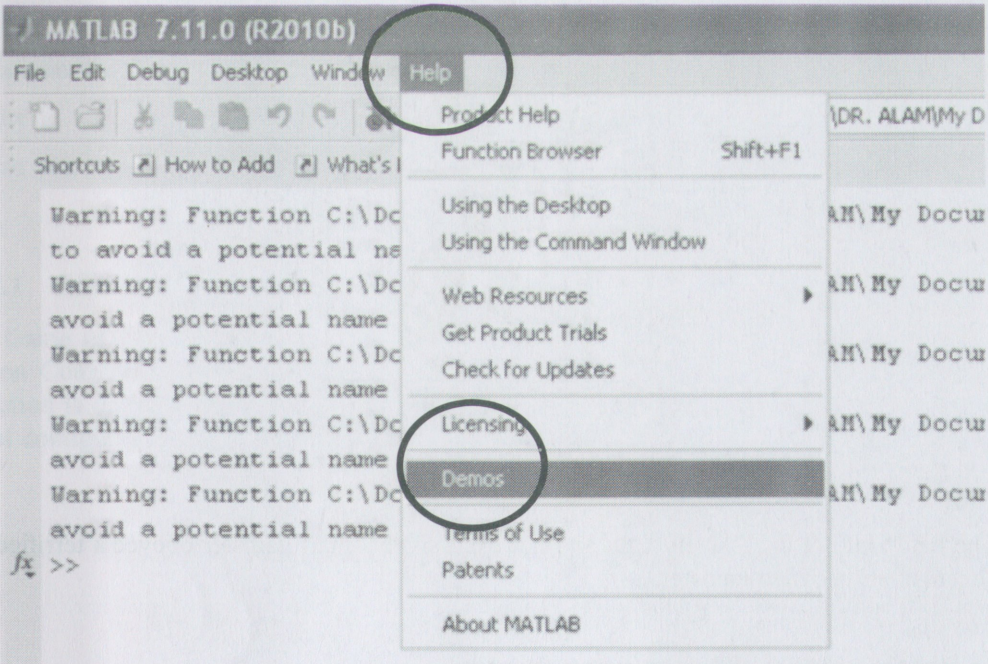
MATLAB Help can be accessed by clicking the HELP. Help has a pull-down menu form which we could choose the topic of help required. MATLAB Help is contained theoretical details and also shows demos. Help could be used to learn MATALB also.

1.15 STORAGE DURATION OF A VARIABLE

Storage duration is the time between the creation of a variable and its destruction. For example, an auto variable disappears when it goes out of scope. The memory space in which it was stored is free once an auto variable is out of scope.

1.16 FUN WITH MATLAB AND BUILT-IN DEMOS

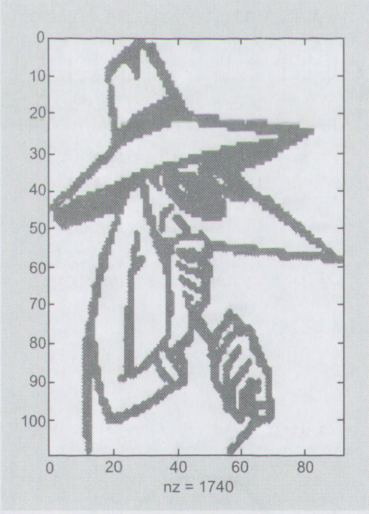
MATLAB includes several demonstrations and also their script files (.m files) that show the various features and functions available in MATLAB. For the complete list of the demos available in MATLAB one can click the 'Help' icon and scroll down to 'Demos' go through the complete list of demos available. This is shown in the following figure.



On typing the command help demos one will be guided how to use the demos. This is shown in the following figure.

```
>> help demos
  Examples and demonstrations.

    demo                - Open the Help browser and select the MATLAB demos.
    echodemo            - Run a cell script as an echo-and-pause command line demo.


demos is both a directory and a function.

    DEMOS Start DEMO.

    See also demo.

fx >>
```

>>spy



>>why

The bald and not excessively bald and not excessively smart hamster obeyed a terrified and not excessively terrified hamster.

>>why

To fool the tall good and smart system manager.

>>why

The rich and tall and good system manager suggested it.

```
>>why
   He wanted it that way.
>>why
   The programmer suggested it.
>>why
   Barney suggested it.
>>why
   To please a very terrified and smart and tall engineer.
>>why
   The tall system manager obeyed some engineer.
>>why
   To satisfy some programmer.
>>why
   Damian wanted it that way.
>>why
   Can you rephrase that?
>>
```

1.16.1 Simulation of Conway's Game of Life

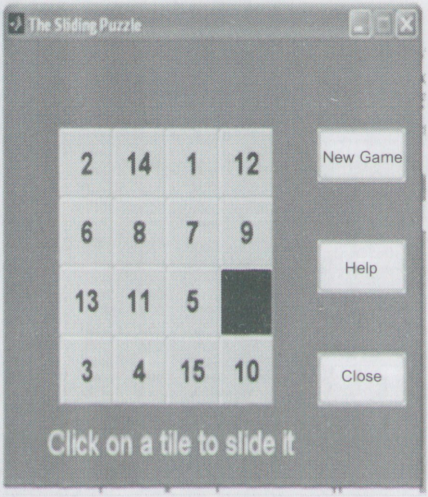
The Game of Life, popularly known as Life, was developed by British mathematician John Horton Conway in the year 1970. It is cellular automation and is a zero player game. The evolution is determined by the initial state and no further input is required. The user gives the initial configuration and observes how it evolves.



1.16.2 fifteen

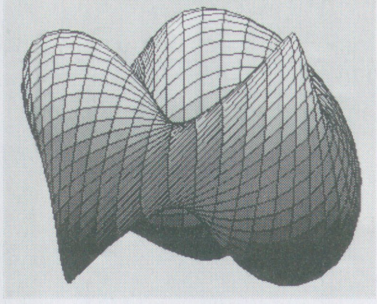
Fifteen is a sliding numbers game. It has 15 numbered squares and 16 slots. You win the game by sequentially lining all fifteen squares one after the other in increasing order and leave the last square unfilled. The slots can be filled by double clicking a tile near the slot.

>>fifteen



1.16.3 Cruller

Cruller gives a 3D image that can be rotated. A cruller can be constructed by revolving the eccentric ellipse defined by the function XYCRULL.

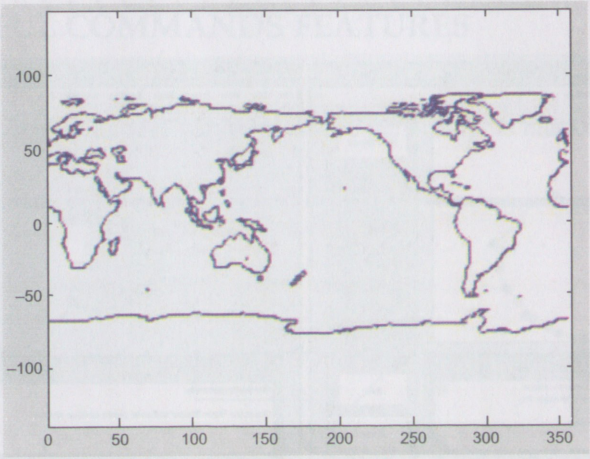


1.16.4 earthmap

>>earthmap

Name	Size	Bytes	Class	Attributes
topo	180x360	518400	double	
topomap1	64x3	1536	double	

>>



1.16.5 imageext

This function imageextlets you see several images.

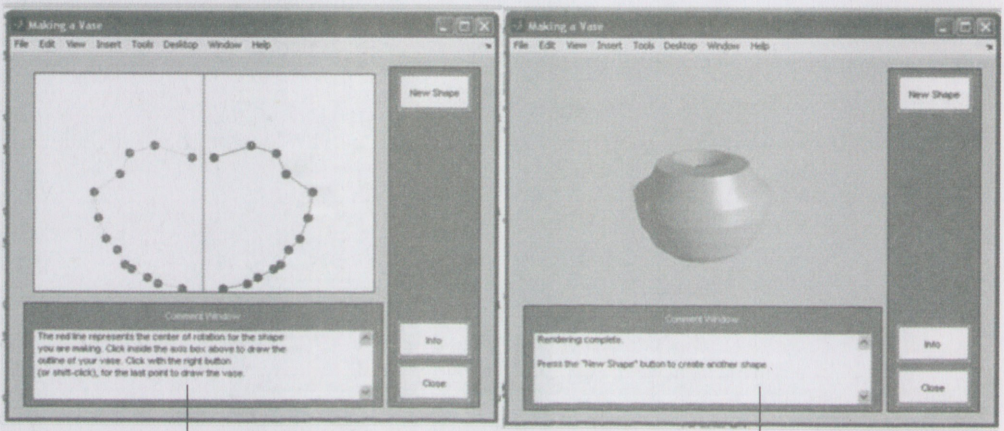
imageext



1.16.6 makevase

Here the red line represents the center of rotation of the shape we are generating. Click the right button or shift-click for the last point the draw the vase and we can see the solid vase that has been designed.

makevase



The red line represents the center of rotation for the shape you are making. Click inside the axis box above to draw the outline of you case. Click with the right button (or shift-click), for the last point to draw he vase.

Rendering complete

Press the "New Shape" button to create another shape

1.16.7 Truss

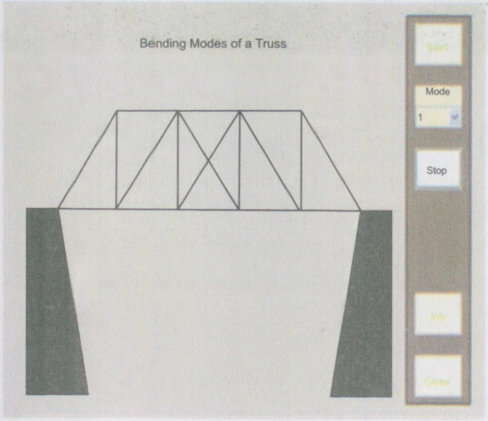
The truss command gives the animation of a bending truss. Here a bridge truss has bee considered. The demo demonstrates 12 natural bending modes of a 2-D truss. The 12 bendin modes are the result of eigenvalue analysis and they have been ordered by natural frequency Here 1 is the slowest frequency and 12 is the fastest frequency. 1 is also the frequency whic is easiest to excite.

>> edit truss

Command allows you to look at the truss.m file.

If we give the command truss the truss.m file will be executed.

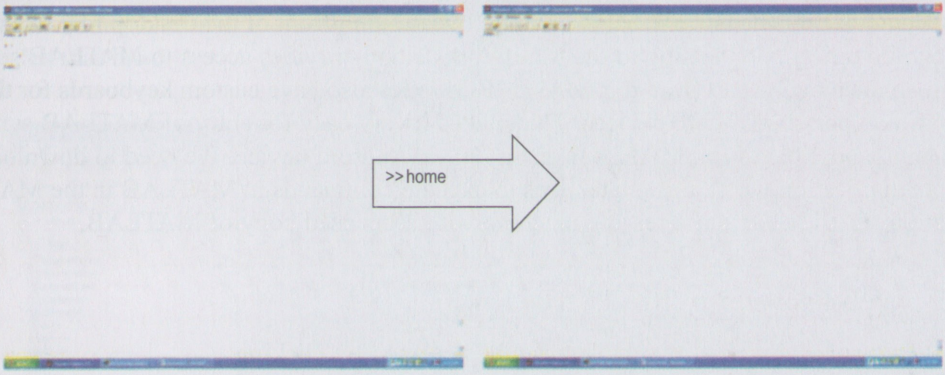
>> truss



1.17 FEW USEFUL COMMANDS FEATURES

1.17.1 Home

home command sends cursor home. Everything above the prompt is hidden by sending it outside the screen.

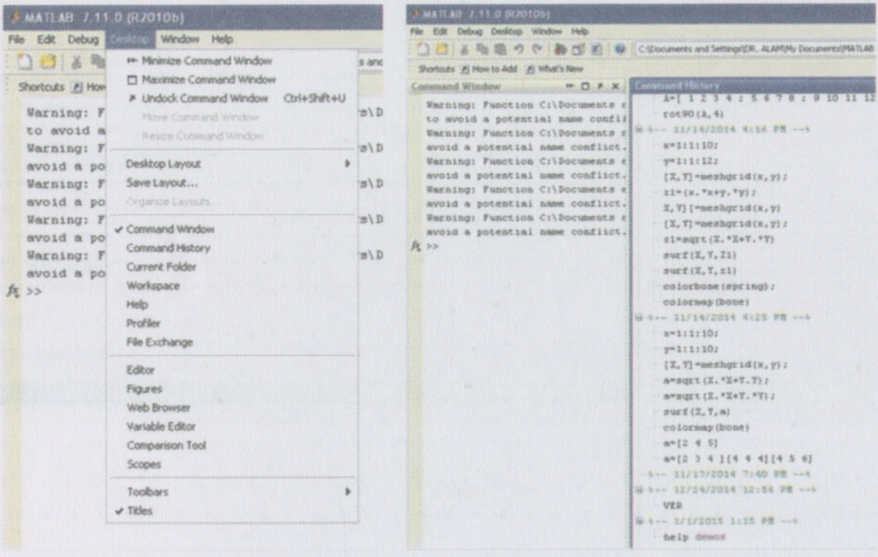


1.17.2 Prompt (>>)

Prompt >> requests the user to give the input. The prompt appears in the command window. In the command window after the MATLAB prompt >> a command can be entered. After the return key is entered the command is executed.

1.17.3 Command History

It is possible to know the command history in MATLAB. This can be obtained by clicking the Desktop option followed by Command History option. As can be seen the command history with date and time is given.

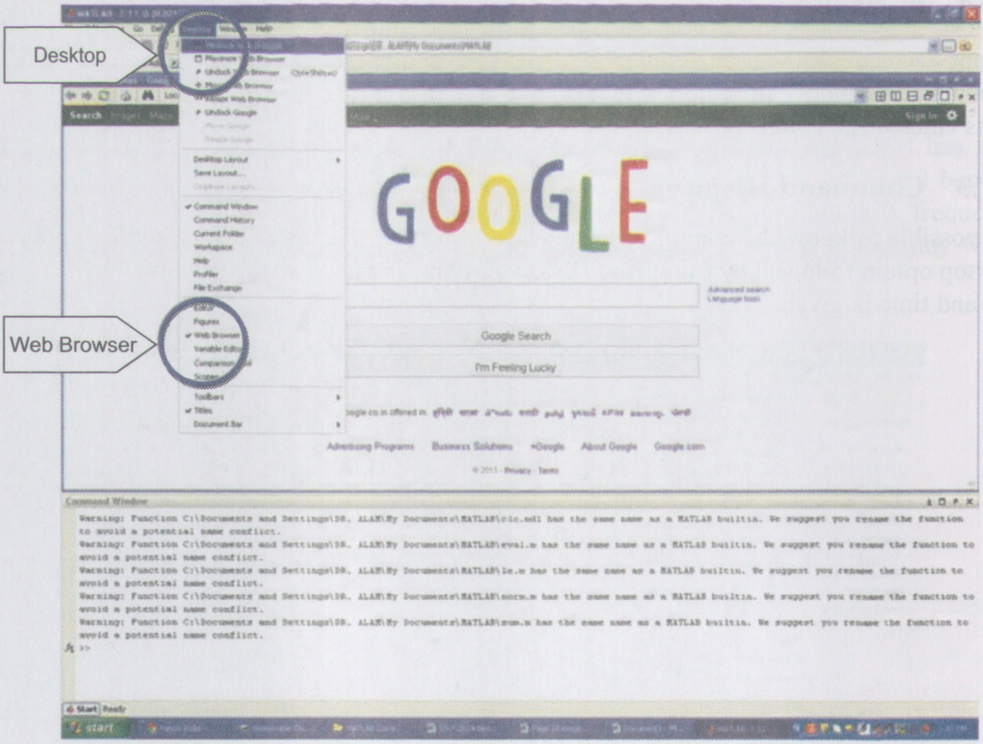


1.17.4 MATLAB® Mobile™

The past decade has seen a phenomenal rise in the use of hand-held and portable devices MATLAB® Mobile™ is just like a desktop on your handheld device like the iPad, the iPhon or an android device. It connects to a MATLAB session running on the MathWorks cloud o on a remote computer. This enables us to interact with the MATLAB session. From the mobil hand-held device one can do all that is possible by sitting in front of the desktop like run scripts and view the output. Connecting to the MathWorks cloud provides access to MATLAB session running on the cloud. Apart from this MATLAB Mobiles also have custom keyboards for the iO and Android operating system devices. This makes it very easy for entry of MATLAB syntax. is very much possible to run MATLAB on an iOS or Android device. We need to download an install an application which then enables us to execute commands in MATLAB in the MATLA application (R2012a+) from a remote machine with a licensed copy of MATLAB.

1.17.5 Web Browser in MATLAB

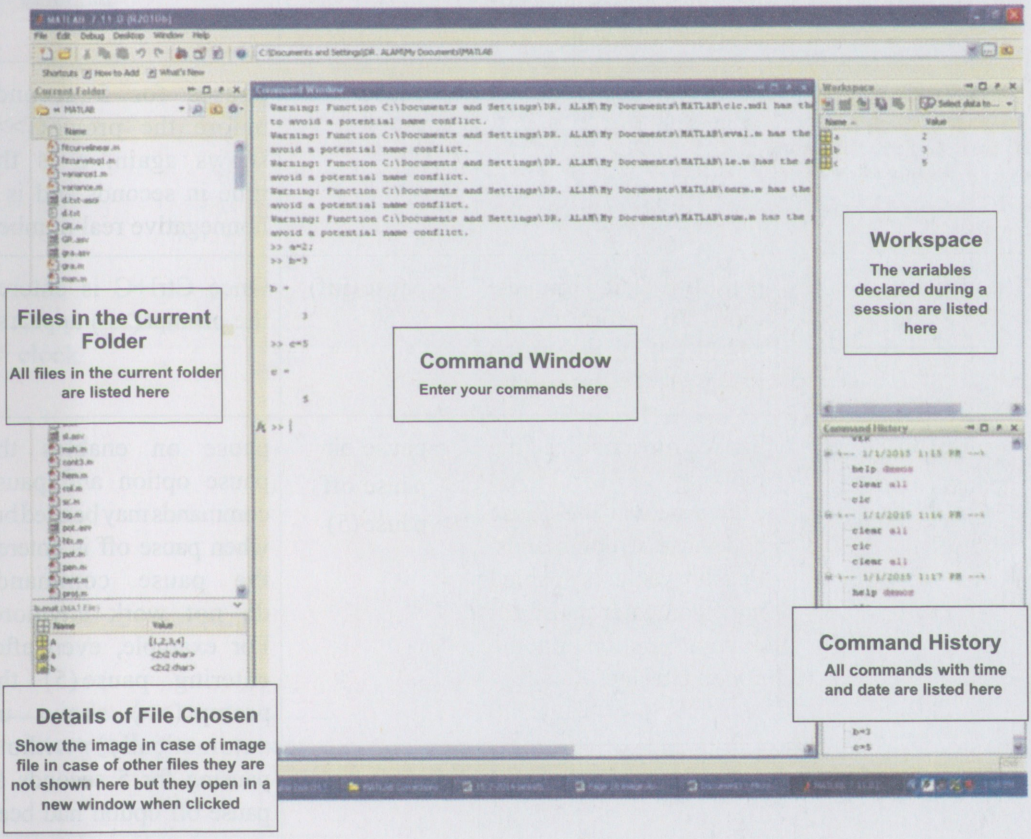
MATLAB also has a web browser. This can be accessed by clicking



As you can see the browser is loaded and one can visit sites just the way we use the Internet explorer or chrome or the safari browser.

1.17.6 The Default Layout

The default desktop layout of MATLAB looks as shown below. It includes the command window, the files in the current folder, the details of the file chosen, the workspace which shows the variables that have been declared and the command history.



1.17.7 Pause

The pause function halts the execution temporarily. The prompt command (>>) does not appear on the command window.

The various syntaxes available for pause are as follows:

The various syntaxes for pause function	Definition	Command	Explanation
pause	pause, by itself, causes the currently executing function to stop and wait for you to press any key before continuing.	>> pause	Once any key is entered the prompt command (>>) is shown
pause(n)	pause(n) pauses execution for n seconds before continuing.	>> pause(5)	Pauses for 5 seconds before the prompt(>>) shows again. n is the time in seconds and is a nonnegative real number.
pause(inf)	pause(inf) starts an infinite loop. To return to the MATLAB prompt (>>) type Ctrl+C is enetered.	>> pause(inf)	Once Ctrl+C is entered the prompt(>>) appears
pause(on) pause(off)	pause on enables the pausing of MATLAB execution via the pause and pause(n) commands. Pausing remains enabled until you enter pause off in your function or at the command line.	>> pause on >> pause off >> pause(5)	pause on enables the pause option and pause commands may be used but when pause off is entered the pause commands do not work anymore. For example, even after entering pause(5) the prompt(>>) shows up automatically without pausing for 5 seconds as pause off option had been entered earlier.
pause query	pause query tells us 'on' if pausing is currently enabled and it displays 'off' if pausing is disabled.	>> pause query ans = off	As pausing is disabled it is displaying off. By default pausing is enabled.

1.17.8 Date, Calendar in MATLAB

It is possible to know the date and calendar of the current month in MATLAB.

Command	Output	Comments
>> date	ans = 19-Mar-2014	Current date can be obtained
>> calendar	Mar 2014 S M Tu W Th F S 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 0 0 0 0 0	The calendar of the current month is obtained
>> clock	ans = 1.0e+003 * 2.0140 0.0030 0.0270 0.0120 0.0130 0.0186	When all the terms in the array are multiplied by 1.0e+003 we get, 2014 3 27 12 13 18.6 2014=year 3= month 27=date 12=hour 13=minutes 18.6=seconds
>> datestr(now)	ans = 27-Mar-2014 12:37:51	Date in string format is obtained with time
>> datestr(today)	ans = 27-Mar-2014	Date in string format is obtained
>> a=eomdate(2001, 5) a = 731002 >> datestr(a)	ans = 31-May-2001	End of month date is obtained. Helpful to know number of days in a month

c = calendar

gives the calendar of the current month. This has been shown above.

c = calendar(d)

Here d is a serial date number or a date string and it returns a calendar for the specified month

>> calendar(731002)

May 2001

S	M	Tu	W	Th	F	S
0	0	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	0	0
0	0	0	0	0	0	0

c = calendar(y, m)

gives the calendar of a particular month.

>> calendar(2000,1)

June 2000

S	M	Tu	W	Th	F	S
0	0	0	0	0	0	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	0	0	0	0	0

So the first day of the millennium was a Saturday.

>> clock

ans =

1.0e+003 *

2.0140 0.0060 0.0160 0.0170 0.0260 0.0235

The above array is,

2014 6 16 17 26 23.5

So the current year is 2014 month is June date is 16th time is 5:26:23.5 PM.

The possibilities with MATLAB seem endless. The more we explore the more number of functions we discover. It is impossible to learn all of them in a single session bit a regular use of MATLAB can definitely get us acquainted with its vast library functions.

2

CHAPTER

The Compiler

*"**Leonhard Euler** is one of the greatest mathematicians of the 18th century. Euler was born in Basel, Switzerland on 15th April 1707. In 2013 his 306th birth anniversary was celebrated. Euler's eyesight deteriorated throughout his career. Three years after suffering from fever he became blind in 1735. Although Euler blamed the work on cartography he performed for St. Petersburg Academy for his condition. He did most of his work during the last two decades of his life when he was totally blind. After Euler became blind he did most of the calculations in his mind. His memory and concentration power was legendary. His computational capability was outstanding and his blindness did not stop him from producing amazing mathematical results. He produced one paper every week in the year 1775. His contributions are Euler's theorem, Euler's equation for motion of fluids, Euler-Lagrange equation and Euler's angles. He introduced symbols like e, i and f(x). Euler published in all 886 papers and the St. Petersburg Academy could not keep up to the pace at which Euler wrote papers and many of them were published posthumously for more than 30 years."*

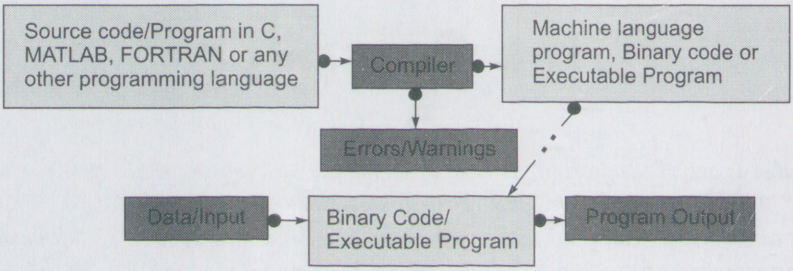
2.1 WHAT IS A COMPILER?

Today, in our everyday life, interactions with the electronic devices or computers have become natural form of our every expressions that we think are more productive way to interface with the rest of the universe. These devices have a lot of power to process information that help us stay ahead in various areas of our choosing.

The interaction enablers on these devices are the various pieces of software that present interfaces for us to use the devices the way we want. Based the usages, these pieces of software create appropriate processing action for devices and instruct them to perform the same and therefore, we see they work for us. These various pieces of software essentially help us to utilize the computational powder of the device.

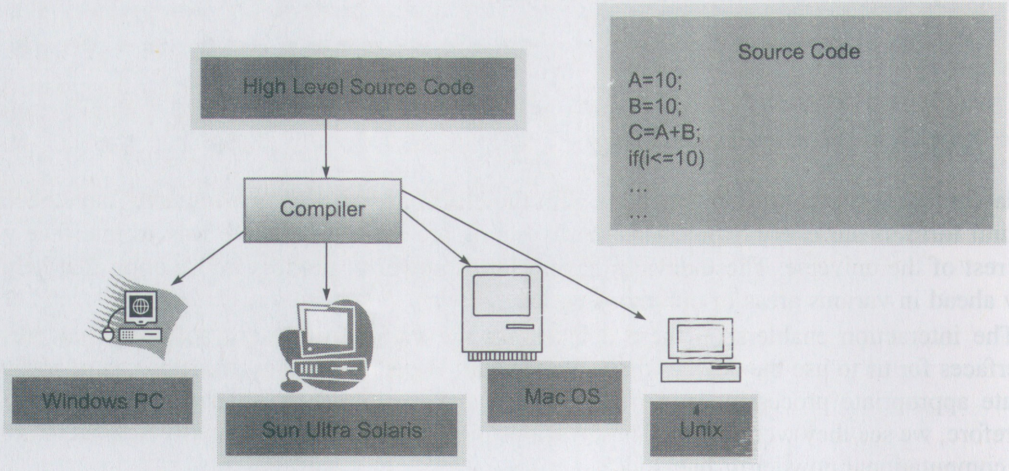
It is critical that we are able to create an elegant software that utilizes the devices efficiently. Also important is the need to have a standard mechanism to design and develop such software. The mechanism must allow us to specify various tasks without having to worry about how computers will understand them. As pieces of software need to instruct the device to perform tasks efficiently and, at the same time, we need to be able to dvelop a software that we can converse with, there grew the need to be able to build pieces of software from a form of human language (programming language) that will interface the device with machine language. Here is where we got ourselves what is known as the "compiler" that provides the transparency between human (programming) and machine language.

A compiler is a process that converts a software program written by human-understandabl language (high-level programming language) into a computer-understandable language (low level machine language) in order for the program to work on a computer. The code in which w write a program is in a high-level language but the computer would not understand anything bu the machine language. So we need someone to translate the high-level code into machine code The translator that does this work for us is called the compiler.



Compiler overview

The machine code is CPU specific. Compilers allow the source code to be independent of th target processor. This implies that the source code is independent of the target processor. So it i not needed to generate source code for different processors but only a suitable compiler need to be purchased.



The program is platform independent

The generated machine code can be executed later many times against different data input An interpreter reads an executable source program written in a high-level programming language as well as data for this program and it runs the program against the data to produce the output Both the interpreters and the compilers are written in some high-level programming language just like any other program and it may be different from the language they accept and they are later translated into machine code.

So a compiler is a program that translates a source program written in some high-level programming language into a machine code for some computer architecture. Computer architecture is conceptual design and fundamental operational structure of a computer system. It is how to design and implement the various parts of a computer. It focuses largely on the way by which the central processing unit (CPU) operates internally and how it accesses addresses in memory. Computer architecture is the technical drawings and functional description of all design requirements.

A compiler is a computer program that transforms the source code into an object code. It is a computer program that transforms the human readable source code of another computer program into the machine readable code that the CPU can execute. This process of converting the source code to a machine code is called compilation. The first compiler was developed by Grace Hooper in the year 1952 while working on the Harvard Mark I computer. He developed this compiler for the A-0 programming language. The compiler converts a high-level programming language into a lower level language, assembly language or a machine language. Compilers translate source code to object code which are unique for each type of computer. Many compilers are available for the same programming language. For example, there is a FORTRAN compiler that is available for Windows PC and another compiler for Apple computers (MAC operating system) or Intel Linux.

2.2 HOW DOES A COMPILER WORK?

As we would like to be able to interact with computers using a language that we are comfortable with, we need compilers to translate out interactions into simple commands making up a software program which will actually interact with computers using machine (assembly) language.

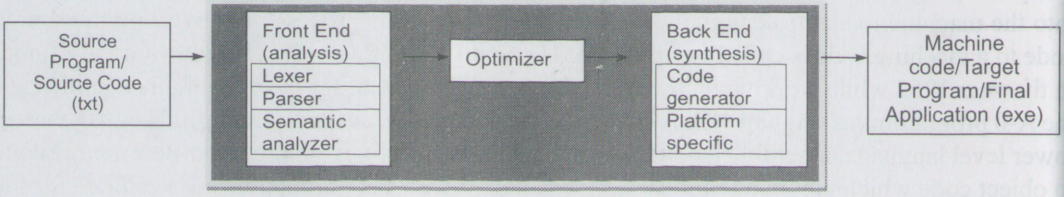
Compiler works as a language processor that reads a program in the "source" language and translates it into an equivalent program in another "target" language. During the process of translation, compiler reports any errors in the "source" program which requires to be fixed in order for a successful translated "target" program. The "target" program is typically a machine-language software program that can be used by users to process inputs and produce the desired output.

The way a compiler works is usually split into the following phases with well-defined interfaces.

1. *Lexical analysis* (*scanning*)—first, the program is read, analyzed and divided into tokens, each of which corresponds to a symbol in the programming language, such as, a variable, a keyword or a number.
2. *Syntax analysis* (*parsing*)—the tokens produced above are arranged in a tree-structure (called the syntax tree) that reflects the structure of the program.
3. *Type checking*—the syntax tree is analyzed to determine if the program violates certain consistency requirements. For example, a variable is used but is not declared, a variable is used in a context that does not make sense due to its type etc.
4. *Intermediate code generation*—program is translated to a simple machine independent intermediate language and the code is generated.
5. *Register allocation*—from the generated intermediate code, the symbolic variable names are translated to numbers and each of these numbers corresponds to a register in the target machine code.

6. *Machine code generation*—the intermediate language is translated to assembly language for a specific machine architechture.
7. *Assembley and linking*—finally, the assembley language code is translated into binary representation after determining addresses of variables, funactions, etc.

The first three phases are called the frontend of the compiler and the last three phases are called the backend. The middle part of the compiler is comprised of the intermediate code generation, but this often also includes various optimizations and transformations on the intermediate code.



Parts of a compiler

Assembly and linking are typically done by programs supplied by the machine or, the operating system. They work with the compiler to complete the compilation process but they are not part of the compiler.

2.3 INTERPRETER AND COMPILER

An "interpreter" is another kind of language processor. Instead of translating "source" code into "target" program an interpreter directly executes operations specified in the source program on inputs provided by the user. As there is no compilation happening on the source code but computer understands only the machine language, interpreter simulates a virtual machine using the basic set of instructions on a programming language as its machine language and that allows interpreter to directly execute operations from the source program.

The target program produced by a compiler is usually much faster than an interpreter processing user actions (inputs) and producing results (outputs). An interpreter, however, can provide better error diagnostics than a compiler as it executes the source program statement by statement.

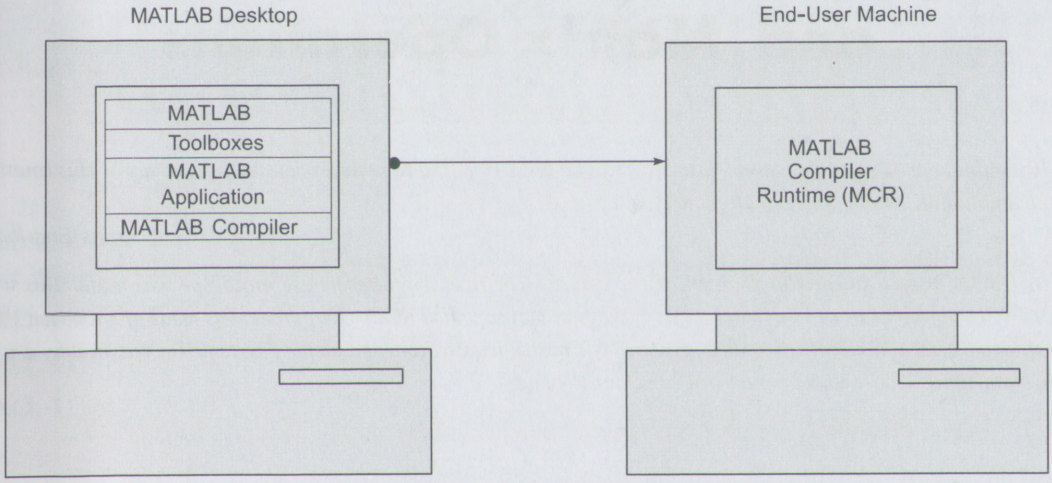
In today's enterprise cloud computing, Java language processors combine an optimum blend of compilation and interpretation. Java source code will be compiled into an intermediate form called "bytecodes". The bytecodes are then interpreted by a virtual machine. The benefit and the power of this combination is that the bytecodes are compiled once, on a single machine and they can be interpreted on a number of virtual machine environments across various networks.

2.4 THE MATLAB COMPILER

The MATLAB Compiler™ converts MATLAB scripts written by the user into stand-alone applications that do not need to be run within the MTALAB environment. When we run these applications they do not require the use of liscences. The MATLAB compiler lets us share MTALAB programs as stand-alone applications or shared libraries for integration with common

programming languages. Applications and libraries created with MATLAB Compiler use the MATLAB Compiler Runtime, which enables royalty-free deployment to users who do not have MATLAB.

The MATLAB Compiler Runime also known as MCR is a stand-alone set of shared libraries that enables the execution of compiled MTALAB application or components on computers where MATLAB is not installed. Thus, when used together, MATLAB, MATLAB Compiler and the MCR enable us to create and distribute numerical applications or software components quickly and securely.



A MATLAB based application compiled as an executable using MATLAB Comipler and deployed on an end-user machine running MATLAB Compiler Runtime

So the MATLAB Compiler only generates wrapper files that interface with the MCR (MATLAB Compiler Runtime) that implements standalone set of shared libraries that enable execution of MATLAB algorithms, even on computers which do not have MATLAB Compiler installed in them. As the MATLAB program is portable they should work on any computer that has MATLAB interpreter. A separate MATLAB compiler is available which can compile a MTATLAB program into a single stand-alone executable which can run without MTALAB licence. This helps in converting prototype MATLAB program into an executable suitable for sale and distribution.

3
CHAPTER

Numbers, Review of Matrices and Matrix Operations

"It would be wonderful if I can inspire others, who are struggling to realize their dreams, to say 'if this countr kid could do it, let me keep slogging away."

– Douglas Engelbar

*"In 1964, **Douglas Engelbart** invented the mouse. The first computer mouse prototype was made. This wa used with a graphical user interface (GUI). The first mouse which received a patent was made of a wooden she with two metal wheels. He described it as an "X-Y position indicator for a display system. The reason why it wa nicknamed the mouse was because tail came out the end."*

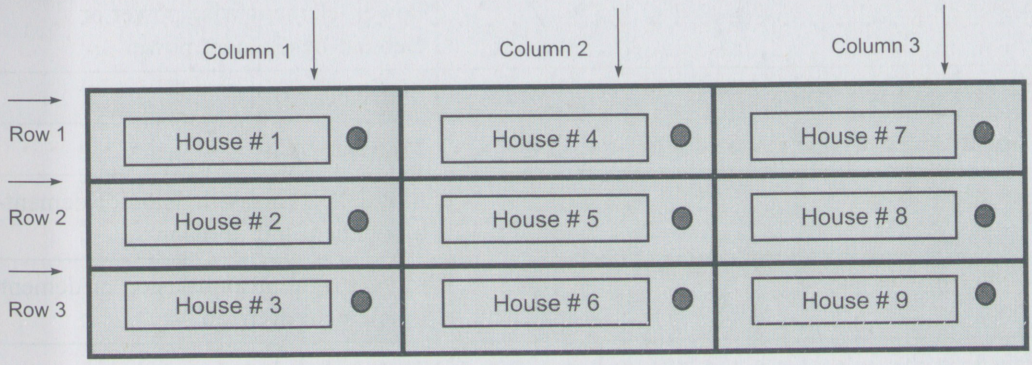
3.1 ARRAYS AND MATRICES

It is essential to know how to handle matrices in order to understand MATLAB well. An array can be defined as number of contiguous memory locations, each of which can store the same data type and which can be referenced through the same variable name. An array is a collective name given to a group of similar elements. Thus, an array is a collection of similar elements. These elements could be all integers, all floating point numbers or all characters. All elements of any given array must be of the same type. In other words there cannot be an array of few integer numbers and other floating point numbers. An array is a series of similar elements in contiguous memory locations. For example, an array of 5 integers could be represented as given below.

	1	2	3	4	5
a	100	200	210	300	310

These elements are numbered from 1 to 5. By using an array all the five integers are stored in contiguous memory locations and all of them can be accessed using the same identifier, with the proper index, *a* which is the name of the array is the identifier of the array and 5 in a[5] specifies the length of the array or the number of elements in the array. Each element of the array can be individually referenced by adding an index to a unique identifier. Array and matrix are terms commonly used to refer to two-dimensional data sets. Generally, those working with images

tend to call a two-dimensional data set an array. On the other hand mathematicians usually call two-dimensional data sets as matrices. The two terms are usually interchangeable.



A 2-dimensional array of letter boxes

It is also possible to make triangular array in mathematics. A triangular array is a doubly indexed sequence in which each row is only as long as the row's own index. Triangular arrays are arrays of random variables in the form as shown below.

A(1, 1)

A(2, 1) A(2, 2)

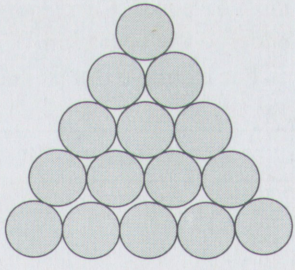
A(3, 1) A(3, 2) A(3, 3)

...

...

A(n, 1) A(n, 2) A(n, 3) ... A(n, n)

Triangular arrays have not been considered here.



Triangular Array

3.2 ARRAY OPERATIONS IN MATLAB

Operation	Sign/Symbol	MATLAB Function
Plus	+	Addition, A + B
Unary plus	+	+A returns A.
Minus	–	Subtraction, A – B
Unary minus	–	– A negates the elements of A.
Matrix multiplication	*	Multiplication, A*B
Array multiplication	.*	Element-wise multiplication or element-by-element multiplication

Power or exponentiation	^	Power, A^B
	.^	Power, element-wise power or element-by-element power
Backslash or left matrix divide	\	Mldivide, A\B
Forward slash or right matrix divide	/	Mrdivide, A/B
Left array divide	.\	rdivide Left array division or element-by-element left division
Right array divide	./	rdivide right array division or element-by-element division
transpose	.'	Array transpose Gives the array transpose of A or unconjugated array transpose

Array operations operate on the corresponding elements of arrays having equal dimensions. For matrices or multidimensional arrays both the number of rows and columns in the operands must be equal. Corresponding elements located at the same cell address (both row and column) undergo the operation.

For example, let us take an array and do the element-wise power operation (.^).

Defining i	**i.^3**	**i^3**
>> i=(1:10)'; >> i i = 1 2 3 4 5 6 7 8 9 10	>> i.^3 ans = 1 8 27 64 125 216 343 512 729 1000	>> i^3 ??? Error using ==>mpower Inputs must be a scalar and a square matrix.

As can be seen every element of i is raised to the power 3 but when i^3 command is given it shows an error. Another example also illustrates how the operation is done element-wise,

Array j	(Array i).^(Array j)
>> j=[1;2;3;4;1;2;3;4;1;2];	>>i.^j ans = 1 4 27 256 5 36 343 4096 9 100

Command	Output	Comments
ceil(9.2)	ans = 10	ceil stands for ceiling and gives the next integer just above the floating point number
a=[2.1 3.5 7.8 9.2]; ceil(a)	ans = 3 4 8 10	ceil(a) in fact rounds the element of a to the next nearest integer that is greater than or equal to a. Here a could also be an array of elements. If a is a complex number, the imaginary and real parts are rounded independently Round towards $+\infty$.
a=[1.22 1.222 1.33 1.333 1.55 1.555];ceil(a)	ans = 2 2 2 2 2 2	
b=[1.22 – 1.222 1.33 – 1.333 1.55 – 1.555];ceil(b)	ans = 2 –1 2 –1 2 –1	
c=3.11 – 2.113i;ceil(c)	ans = 4.0000 – 2.0000i	

floor(9.2) a=[2.1 3.5 7.8 9.2]; b=[2.2 –3.3 4.6 7.1];.	ans = 9 floor(a) ans = 2 3 7 9 floor(b) ans = 2 –4 4 7	floor gives the number just before the floating point number floor(a) rounds the elements of a to the nearest integers less than or equal to A and for complex a, the imaginary and real parts are rounded independently If a is positive then floor(a) and fix(a) will give the same results but if they are negative then floor(a) and fix(a) will give different results
double(9.2)	ans = 9.2000	Double converts a number to double precision floating point number
a=[2.1 3.5 7.8 9.2]; fix(a) b=[2.2 –3.3 4.6 7.1];.	ans = 2 3 7 9 fix(b) ans = 2 –3 4 7	fix(a) rounds the elements of a toward zero. It gives an array of integers. For complex a, the imaginary and real parts are rounded independently
a=[2.333 4.555 6.666 7.777]; round(a) a=[1.22 1.222 1.33 1.333 1.55 1.555]; round(a) b=[1.22 – 1.222 1.33 – 1.333 1.55 – 1.555]; round(b) c=3.11 – 2.113i; round(c)	ans = 2 5 7 8 ans = 1 1 1 1 2 2 ans = 1 –1 1 –1 2 –2 ans = 3.0000 – 2.0000i	round(a) rounds the elements of array a to the nearest integers. Positive numbers with a fractional part of 0.5 is rounded to the nearest positive integer whereas the negative numbers with fractional part of –0.5 is rounded to the nearest negative integer. In case a is a complex number, the imaginary and real parts are rounded independent of each other

3.3 INTEGER REPRESENTATION IN VARIOUS FORMATS

In MATLAB the numeric data are stored as double-precision floating point. This is the default option. To store a data as an integer, one needs to convert from the double precision floating type to the desired integer type. Several conversion functions are available to convert to the various signed and unsigned integer type of variable. The various conversion functions are shown in the following Table.

Class	Range of Values	Conversion Function
Signed 8-bit integer	-2^7 to $2^7 - 1$	int8
Signed 16-bit integer	-2^{15} to $2^{15} - 1$	int16
Signed 32-bit integer	-2^{31} to $2^{31} - 1$	int32
Signed 64-bit integer	-2^{63} to $2^{63} - 1$	int64
Unsigned 8-bit integer	0 to $2^8 - 1$	uint8
Unsigned 16-bit integer	0 to $2^{16} - 1$	uint16
Unsigned 32-bit integer	0 to $2^{32} - 1$	uint32
Unsigned 64-bit integer	0 to $2^{64} - 1$	uint64

For example, in order to store the number 222 as a 16-bit signed integer the command should be,

```
>> a=int16(222)
a =
   222
```

In case the number to be converted to an integer has a fractional part, MATLAB rounds it to the nearest integer. In case the fractional part is 0.5, then from the two equally nearby integers the integer which has the larger absolute value is chosen:

For example,

```
>> a=int16(222.2222)
a =
   222
>> b=int16(222.5)
b =
   223
```

Integer format	Result
>> int64(2000.000)	ans = 2000
>> int32(2000.000)	ans = 2000
>> int16(2000.000)	ans = 2000
>> int8(2000.000)	ans = 127

The largest value of an integer depends on the class of the integer. For example, the largest integer in int32 class containing 32 bits is

+/–	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The decimal equivalent of the above binary number is 2147483647. This is the maximum integer of 32-bit integer number. This can be obtained using the command,

intmax('classname')

The various classnames, int8, int16, int32, int64, unit8, unit16, unit32 and unit64. intmax('int32'), give the same result as intmax.

3.3.1 Single and Double

The function single converts a number to single precision. The function double converts a number to double precision.

MATLAB Command	Output	Explanation
>> a=10.123456789	a = 10.1235	
>> single(a)	ans = 10.1235	
>>whos	Name Size Bytes Class Attributes a 1x1 8 double ans 1x1 4 single	single(a) has 4 bytes and a has 8 bytes as it is a double precision number.
>> a=10.111234567	a = 10.1112	
>> b=1.23456789 b = 1.2346	>> double(b) ans = 1.2346 >> whos Name Size Bytes Class Attributes ans 1x1 8 double b 1x1 8 double	Double converts the number a to a double precision number having 8 bytes.

Note here that a bit is a single numeric value. It has a value either 1 or 0 and encodes a single unit of digital information. Digital information is stored in the bits. A byte is a sequence of bits and usually eight bits make one byte. So a double precision number is stored in 8 bytes which implies it requires 64 bits whereas a single precision number which requires 4 bytes implies it requires 32 bits.

3.3.2 Floating Point Number

In computing floating point is a method for representing an approximation of a real number in such way that it can support a wide range of values. A floating point number is represented approximately to affixed number of significant digits, known as the significant and scaled using an exponent. The base for scaling in most cases is 2, 10 or 16. In a decimal floating point number the base is 10.

$$\text{Floating point number} = \text{mantissa} \times (\text{base})^{(\text{exponent})}$$

For example $1234.56 = 1.23456 \times 10^3$

MATLAB Command	Output
a = 1.1111	>>isfloat(a) ans = 1

In a byte there are 8 bits or 8 bits form a byte. In computers we use a binary system so if we have a number 25 in decimal system then that number would be,

$$1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 11001$$

in binary system.

So if there are 8 bits in a byte the bits could all be filled with 0 or all could be filled with 1. If all are filled with 1 then it becomes,

1	1	1	1	1	1	1	1

And the equivalent decimal number for this is 255. So if there are 8 bits in a byte we can represent decimal numbers 0 to 255 using this byte.

So if we have two bytes the maximum number that can be represented is,

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The maximum number that can be represented is, 65535.

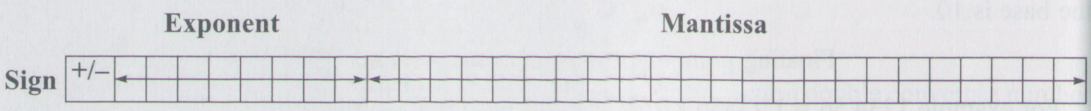
4 bytes are used to store a floating point number. As 1 byte is made of 8 bits so 4 bytes have 32 bits. In a floating point number there are two parts, i.e., one is the exponent and the other mantissa.

Floating point number = mantissa × (base)$^{(exponent)}$

For example, in a number line $2.134567 = 2124567 \times 10^{-6}$ the mantissa is 2124567 and the exponent is 10^{-6}.

Floating point number	Mantissa	Exponent
2.134567 =	2124567	10^{-6}

Similarly, a floating point number that is stored in the memory has two components: mantissa and exponent. Any floating point number is stored as a binary number consisting of 1 and 0 in an array of maximum 32 bits.



Here 1 bit is reserved for sign of the number, 8 bits for the exponent and 23 bits of the mantissa.

The range for single precision is $1.175494351 \times 10^{-38}$ and $3.402823466 \times 10^{38}$ and $-3.402823466 \times 10^{38}$ to $-1.175494351 \times 10^{-38}$.

realmax('double') is the same as realmax with no arguments.

realmax('double') is the same as realmax with no arguments.

EDU» realmax

ans =

1.7977e+308

realmax('single') returns the largest finite floating-point number in IEEE single precision.

ans =

See Also

The program to find the mantissa and the exponent of a floating point number in C is given below.

```
class(x)
ans =
single
```

double precision

eps(1/2) = 2^(–53)
eps(1) = 2^(–52)
eps(2) = 2^(–51)
eps(realmax) = 2^971
eps(0) = 2^(–1074)

By default MATLAB adopts the double-precision format given by IEEE in which 8 bytes which is equivalent to 64 bits are used to represent a floating-point number.

1 bit	11 bits	52 bits
Single bit used for sign	**Used for exponent**	**Mantissa**

The 52 bits used for the mantissa can be used for a mantissa of length of about 15 to 16 base-10 digits.

```
format long
pi
ans =
   3.141592653589793
```

3.4 DEFINING A VARIABLE IN MATLAB

In MATLAB there are three basic types of variables. They are scalars, vectors and matrices. Variables are defined by typing the name of the variable followed by an equal sign and the value assigned to the variable. Vectors and matrices are denoted by enclosing the value in square brackets and rows within a matrix are separated by semicolons. The three types of variables are illustrated below.

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

Here in the first case a is a scalar equal to 1. In the second case a is vector and a=[1 2 3]. These are the three components of the vector. In the third case A is a matrix and a=1 2 3; 4 5 6;7 8 9]. A is a 3×3 matrix.

3.5 REAL NUMBER

Positive, negative, large, small, whole numbers, rational numbers, square roots, cube roots, decimal numbers, etc., are all real numbers. The field of rational and irrational numbers is called real numbers and it is often demoted by R. In fact all numbers on the number line are real numbers. Any number that is not imaginary is real. For example, 1, –1.23, $\sqrt{2}$, π, etc.

Command	Result
>> realmax	ans = 1.7977e+308
>> format long >> realmax	ans = 1.797693134862316e+308
>> realmin	ans = 2.2251e-308
>> format long >> realmin	ans = 2.225073858507201e-308

3.6 ABSOLUTE VALUE

abs( ) gives the absolute value and complex magnitude. abs ( ) returns the absolute value (|variable|) or the magnitude of a real number or argument. The argument could be real or complex vector or matrix. The value returned is always positive. A negative value is changed to positive and returned whereas a positive value remains unchanged. When x is complex, abs(x) returns the complex modulus or magnitude of the elements of x.

abs(variable) returns,

$$\text{abs(variable)} = \begin{matrix}\text{variable, for variable} \geq 0 \\ \text{variable, for variable} < 0\end{matrix}$$

```
>> abs(–1)
ans =
    1
>> abs(–1.1)
ans =
    1.1000
>> abs(–2.234)
```

```
ans =
    2.2340
>> abs(–2)
ans =
    2
>>abs(2–3i)
ans =
    3.6056
```

3.7 ASCII, EBCDIC

In order to standardize character representations, computer manufacturers have designed their computers on ASCII or EBCDIC. These are the two most popular coding schemes, known as the ASCII or EBCDIC character CODE or character sets. EBCDIC stands for Extended Binary Coded Decimal Interchange Code. ASCII stands for the American Standard Code for Information Interchange. ASCII and EBCDIC are the coding schemes that are most commonly used. Strings or character manipulation does not manipulate the character themselves but manipulate the numeric codes. Using the numeric codes it is possible to compare various character strings. When two strings are compared the computer compares the numeric codes of the characters in each string.

ASCII has been developed from telegraphic codes. It was first used in a 7-bit teleprinter code at Bell data services. ASCII standard started in 1960 during the first meeting of the American Standards Association. The first edition of the standard was published in 1963 and revised in 1967. Both the ASCII and the Bell code were meant for more convenient sorting of lists and features that could be used with devices other than the teleprinter. ASCII has the definition of 128 charters. Among them 33 are non-printing and 95 printable characters including the space character.

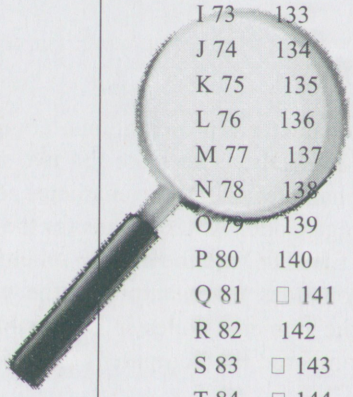
ASCII stands for American Standard Code for Information Interchange. It is a character-encoding scheme that encodes 128 specified characters. They include the numbers 0–9, the small letters a–z and capital letters A–Z. Apart from this they include basic punctuation symbols, control codes which originated from the teletype machines, and a blank space. Here each of these characters is converted into a 7-bit binary integer.

The ASCII values of the alphabets and symbols can be obtained using the program given below.

```
for K = 32 : 91
    L = K+60;
string = [char(K) ' ' num2str(K)'  'char(K+60)' ' num2str(K+60)];
disp(string)
end
```

Output

32 \ 92	A 65 } 125
! 33] 93	B 66 ~ 126
“ 34 ^ 94	C 67 □ 127
# 35 _ 95	D 68 128
$ 36 ` 96	E 69 □ 129
% 37 a 97	F 70 130
& 38 b 98	G 71 131
‘ 39 c 99	H 72 132
(40 d 100	I 73 133
) 41 e 101	J 74 134
* 42 f 102	K 75 135
+ 43 g 103	L 76 136
, 44 h 104	M 77 137
- 45 i 105	N 78 138
. 46 j 106	O 79 139
/ 47 k 107	P 80 140
0 48 l 108	Q 81 □ 141
1 49 m 109	R 82 142
2 50 n 110	S 83 □ 143
3 51 o 111	T 84 □ 144
4 52 p 112	U 85 145
5 53 q 113	V 86 146
6 54 r 114	W 87 147
7 55 s 115	X 88 148
8 56 t 116	Y 89 149
9 57 u 117	Z 90 150
: 58 v 118	[91 151
; 59 w 119	>>
< 60 x 120	
= 61 y 121	
> 62 z 122	
? 63 { 123	
@ 64 | 124	



Here the function num2str converts an ASCII code into the character that it represents. Here M has an ASCII code of 77 and N has an ASCII code of 78.

3.8 INFINITY (∞)

Infinity represents something which does not have limit. Infinity finds its applications in mathematics and physics. The word "infinity" comes from the Latin word "infinitas" which means unboundedness. The reciprocal of an infinitesimal is infinity. Infinity is greater than any real number that can be imagined. Infinity is endless, something that never ends. Infinity is not a real number it is a concept or an idea. It is something that cannot be measured.

A few examples of infinity are,

$$22/7 = 3.142857\ 142857\ 142857\ 142857\ 142857\ 1.....$$

This goes on and on infinitely and never ends. 22/7 is an approximation of π.

or $$1/3 = 0.3333333333.....$$

The number of points in a line or a circle is infinite.

In MATLAB Inf sands for infinity.

```
EDU» 1/0
Warning Divide by zero.
ans =
Inf
EDU» inf
ans =
Inf
EDU» inf*2
ans =
Inf
EDU» 1/inf
ans =
     0
```

3.9 IMAGINARY NUMBER

Rene Descartes used the term "imaginary number" originally and wrote about these numbers in La Géométrie to refer to the numbers that are today known as the complex numbers. The term imaginary was considered derogatory. An imaginary number is a number that does not exist. i is called the imaginary unit.

$$i = \sqrt{-1} \quad \text{and}$$
$$i^2 = -1$$

i multiplied by a real number converts the number into an imaginary number. An imaginar number when squared is either 0 or negative. This is a nonexistent number because a numbe multiplied by itself cannot be negative. Initially, imaginary numbers were considered useless an derogatory. Later during the Enlightenment Era or the Age of Reason in the 17th and 18th century mathematicians like Euler, Gauss and Wessel demonstrated their applications and usefulness Imaginary numbers are square root of real negative numbers. i is used in signal processing electromagnetism, fluid dynamics, quantum mechanics and several other areas. Imaginar number is also used in several equations like the Euler equation. An imaginary number whe added to a real number gives a complex number. a+ib is a complex number where 'a' is th real part and 'b' is the imaginary part of the complex number. Thus, an imaginary number is complex number with real part equal to 0

```
EDU» i

ans =

    0+1.0000i

EDU» i*i

ans =

    -1

EDU» 2+i

ans =

    2.0000+1.0000i
```

3.10 MANTISSA

The mantissa is the significant or coefficient of a number in scientific notation or floating poin number. For a real number mantissa is the positive fractional part. Mathematically mantissa can be expressed as,

For a real number x,

Mantissa = x – floor(x)

For a real number x, floor(x) = largest previous integer

For example,

Let us consider a real number 10.1111

```
EDU» x=2.3;
EDU» floor(x)

ans =

    2
```

$$\text{Mantissa} = 10.1111 - 10 = 0.1111$$

Here 0.1111 is the mantissa.

Another way of representing a real number is,

$$10.1111 = 101111 \times 10^{-4}$$

In this case 101111 is the mantissa and 10^{-4} is the exponent.

The word mantissa is commonly used in computer science. The use of mantissa has not been encouraged by some professionals like William Kahan and Donald Knuth. As mantissa is also the

term used for the fractional part of a logarithm, it could cause confusion. William Kahan is the inventor of floating point notation. Donald Knuth is an American computer scientist at Stanford University. He is the author of the multi-volume work "The Art of Computer Language". He is regarded as the father of the analysis of algorithms.

If we analyze the logarithm of a number then we find that it has an integer and fractional part. For example,

$$\log(50) = 1.69897000433601880478626110527555$$

Here 1 is the integer part or characteristic and 0.69897000433601880478626110527555 is the fractional part or mantissa. In fact few dictionaries give the meaning of mantissa as the decimal part of a logarithm only.

3.11 eps

eps is the floating-point relative accuracy. eps returns the distance from 1.0 to the next largest double-precision floating point number. eps = 2^(–52) which is equal to 2.2204e-016. eps actually stands for epsilon or machine epsilon. Machine epsilon gives an upper bound on the relative error due to rounding while using floating point numbers. For a binary64 double precision floating point format the value of machine epsilon is 2^{-52} = 2.22e-16. The built-in eps is used as the tolerance for all internal calculations.

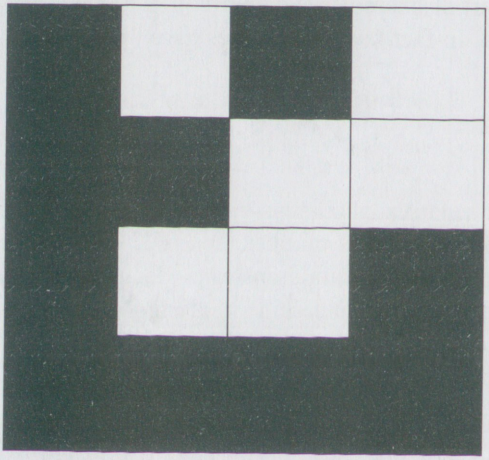
eps	**eps = 2^ (–52)**	**eps ('single')**	**eps ('double')**
>> eps ans = 2.2204e-016	>> 2^(–52) ans = 2.2204e-016 Note that the value of eps is equal to 2^(–52).	>> eps('single') ans = 1.1921e-007 >> eps(single(1.0)) ans = 1.1921e-007 Note: eps('single') is the same as eps(single(1.0)) or single(2^-23)	>> eps('double') ans = 2.2204e-016 >> eps(1.0) ans = 2.2204e-016 Note: eps('double') is the same as eps or eps(1.0). eps is equal to 2^(-52)

Expressions like eps(X) = eps(–X) = eps(abs(X)) are also valid.

3.12 HADAMARD MATRIX

Hadamard matrix is a type of square matrix where placing any two columns or rows side by side then half the adjacent cells has the same sign while half has the other sign. The cells of the matrix with value 1 could be colored black and those with value –1 could be colored white. Thus a n × n Hadamard matrix H_n must have n(n–1)/2 white squares (–1 value) and n(n+1)/2 black squares (1 value).





A 2 × 2 and 4 × 4 Hadamard Matrix

In MATLAB hadamard(n) returns a Hadamard matrix of order n.
For a Hadamard matrix a,

a'*a = n*I
Where I = eye(n,n)

hadamard(2)	**hadamard(2)'**	**hadamard(2)'* hadamard(2)**
>> a=hadamard(2) a = 1 1 1 –1	>> a' ans = 1 1 1 –1	>> a'*a ans = 2 0 0 2
So, a' * a =2*I (I= eye(2,2))		

Hadamard matrices are matrices made up of 1's and –1's, whose columns are orthogonal.

3.13 UPPER AND LOWER TRIANGULAR MATRIX

The upper triangular matrix is of the form,

$$U_{ij} = A_{ij} \quad \text{for } i \leq j$$
$$= 0 \quad \text{for } i > j$$

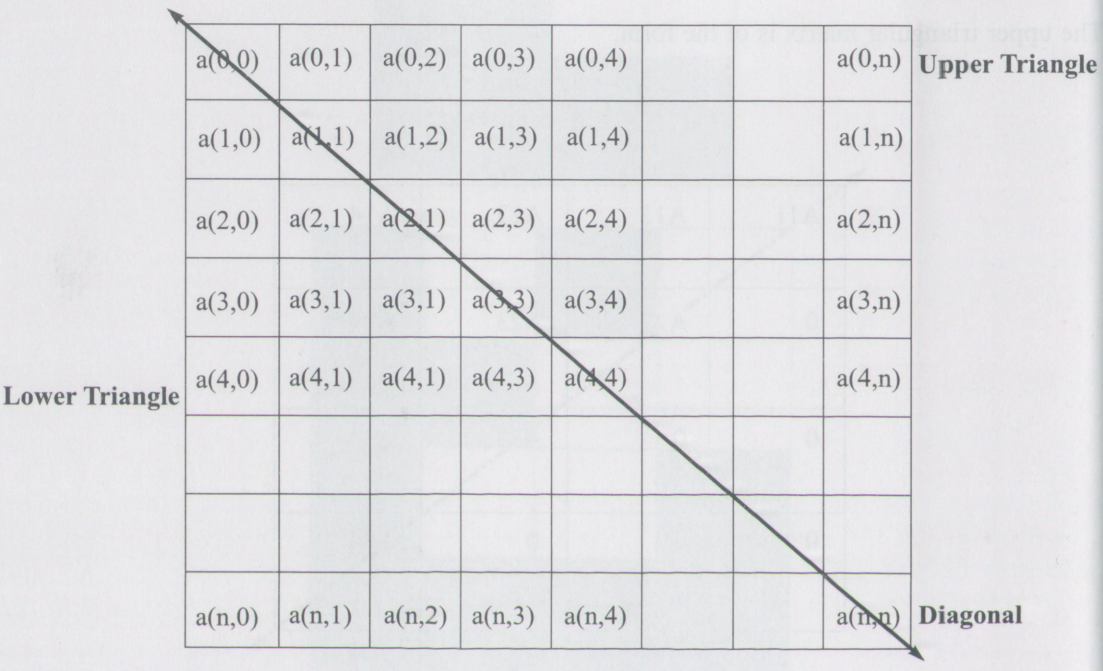
A11	A12	A13	A14
0	A22	A23	A24
0	0	A33	A34
0	0	0	A44

The lower triangular matrix is of the form,

$$L_{ij} = A_{ij} \quad \text{for } i \geq j$$
$$= 0 \quad \text{for } i < j$$

A11	0	0	0
A21	A22	0	0
A31	A32	A33	0
A41	A42	A43	A44

3.13.1 Upper Triangle, Lower Triangle



In the upper triangle the column index (j) is always higher than the row index (i). Here $j > i$. In the lower triangle row index (i) is always higher than the column index (j). Here $i > j$. For the diagonal elements the row index (i) is equal to the column index (j). Here $i = j$.

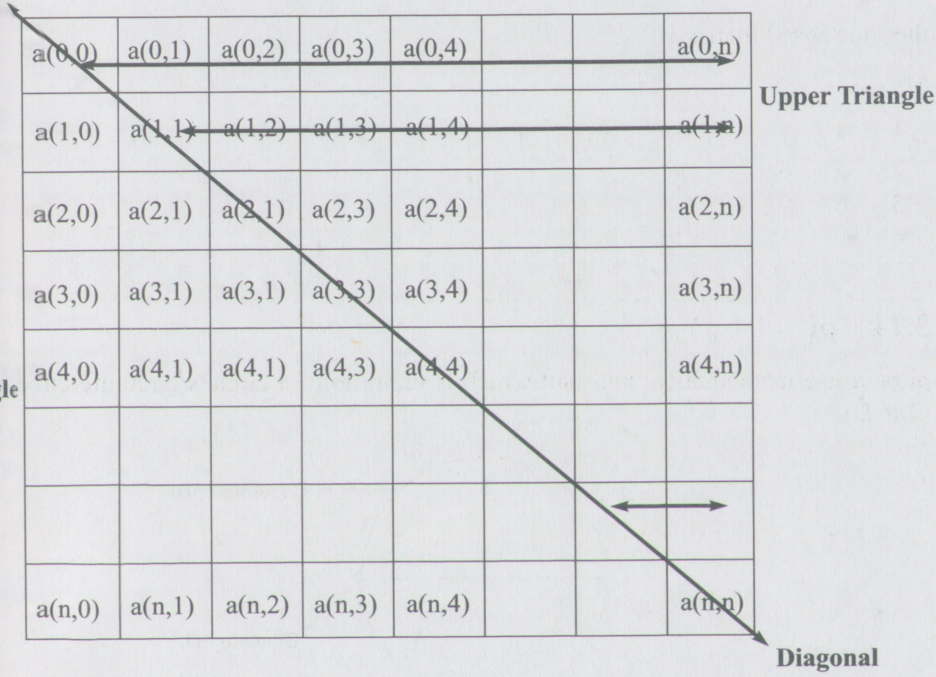
3.13.2 Transpose of a Matrix

Only the elements that have been marked are to be interchanged with the corresponding elements in the lower triangle. Then it will be possible to transpose the matrix.

```
m=input('enter the value of rows:');
n=input('enter the value of columns:')
for i=1:m
for j=1:n
        a(i,j)=input('enter the element');
end
end
disp('the matrix is:')
a
for i=1:m
for j=i:n
        temp=a(i,j);
        a(i,j)=a(j,i);
        a(j,i)=temp;
```

Here i is chosen from 1 to m but j is chosen from i to n. Which implies that in row i the columns start from i and continue to j.

```
end
end
disp('the transposed matrix is:')
  a
```




```
enter the value of rows:3
enter the value of columns:3
n =
     3
enter the element1
enter the element2
enter the element3
enter the element4
enter the element5
enter the element6
enter the element7
enter the element8
enter the element9
the matrix is:
```

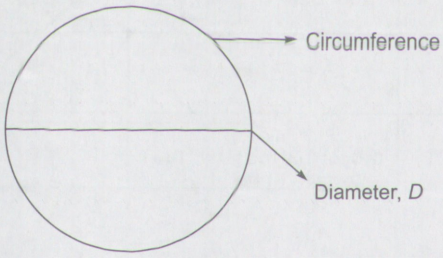
```
a =

     1     2     3
     4     5     6
     7     8     9
```

the transposed matrix is:

```
a =

     1     4     7
     2     5     8
     3     6     9
>>
```

3.14 pi

pi or π is a mathematical constant which is the ratio of a circle's circumference to its diamete ($2\pi r/D$).



A Circle

Its value is approximately 3.141592653589793.... . It is represented by the Greek letter π. It i an irrational number as it cannot be expressed exactly by the ratio of two integers. As a resul the decimal representation of pi is never ending. Finding the exact value of pi has been an are of research for a long time. pi is also a transcendental number which implies that it is not a roo of a polynomial which has rational coefficients. There are a number of series to evaluate th value of pi. Some of them are given below:

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + }}}}}}}$$

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdots$$

One such series which converges fast is the Gregory-Leibniz series given by,

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} \cdots\cdots$$

This series has been put in a for loop to evaluate the value of pi up to a definite number of erms. There are several other series which also give the value of pi approximately.

```
EDU» pi
ans =
   3.1416
```

3.15 meshgrid

The meshgrid command generates two arrays. One of the arrays contains the x-coordinates and he other contains the y-coordinates. For example,

```
>> a=[1 2 3];
b=[5 6 7];
[X,Y] = meshgrid(a,b)
X =
     1     2     3
     1     2     3
     1     2     3
Y =
     5     5     5
     6     6     6
     7     7     7
```

This returns two 3 by 3 arrays. The X matrix defines the x-coordinates and the Y matrix the y-coordinates at each position in the 3 by 3 grid.

1	2	3
1	2	3
1	2	3

The x-coordinates formed from the a matrix

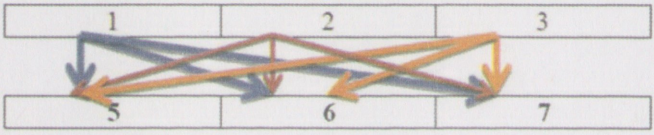
5	5	5
6	6	6
7	7	7

The y-coordinates formed from the b matrix

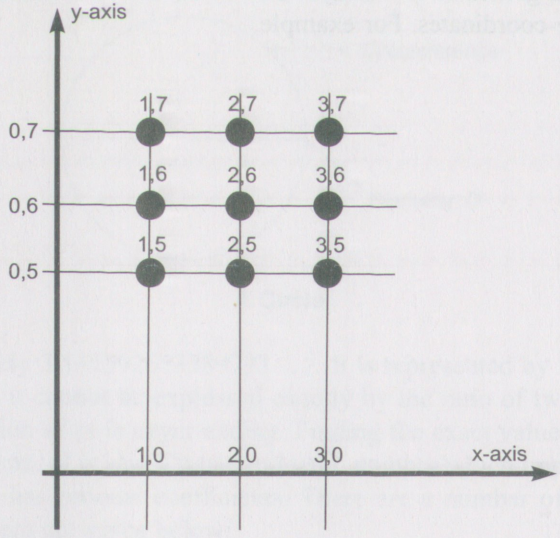
1,5	2,5	3,5
1,6	2,6	3,6
1,7	2,7	3.7

The (x,y) coordinates of the area generated by meshgrid (a,b)

The combination of the 'a' and the 'b' matrices.



The meshgrid is often used for computing functions having 2 variables in a rectangular ar in the x-y coordinate system.

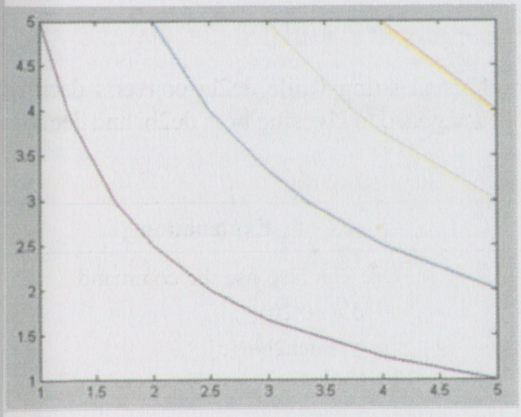



```
>> x=[1 2 3 4 5];
>> y=[10 20 30 40 50];
>> [X,Y]=meshgrid(x,y)
```

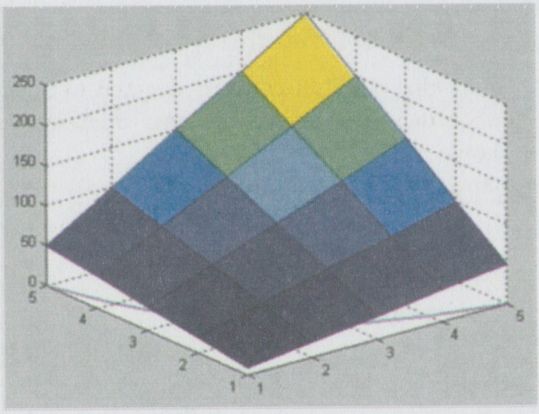
X =

```
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
1   2   3   4   5
```

```
Y =
    10    10    10    10    10
    20    20    20    20    20
    30    30    30    30    30
    40    40    40    40    40
    50    50    50    50    50
>> Z=X.*Y;
>> contour(Z)
>> surfc(Z)
```



contour(Z)



surfc (Z)

The arrays X and Y when overlapped becomes,

(1,10)	(2,10)	(3,10)	(4,10)	(5,10)
(1,20)	(2,20)	(3,20)	(4,20)	(5,20)
(1,30)	(2,30)	(3,30)	(4,30)	(5,30)
(1,40)	(2,40)	(3,40)	(4,40)	(5,40)
(1,50)	(2,50)	(3,50)	(4,50)	(5,50)

Here the x-axis and y-axis in the plots show the row and column number of the element chosen. For example X as 5 and Y as 5 implies that the row number is 5 and the column number is 5 in the meshgrid and the value of X and Y in this position are (5, 50). When X and Y are multiplied it gives 250.

3.16 CONVERT DECIMAL TO BINARY AND BINARY TO DECIMAL, dec2bin, de2bi AND bin2dec

The function dec2bin converts decimal to binary number in string. dec2bin(a) converts d to a binary number as a string. A should be a non-negative integer smaller than 2^{52}. dec2bin(a,k) gives the binary representation of d with at least k bits.

```
>> dec2bin(200)
ans =
    11001000
>> dec2bin(100)
ans =
    1100100
```

Note that dec2bin converts decimal to binary number in string while de2bi converts decimal numbers to binary vectors. The binary representation of integers 0 to 20 using both de2bi and dec2bin is shown below.

Command	Output	Explanation
>> d = (0:20); b = de2bi(d)	b = 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 1 1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 0 1 0 1	One can also use the command >> d = (0:20); >> b = dec2bin(d) b = 00000 00001 00010 00011 00100 00101 00110 00111 01000 01001 01010 01011 01100 01101 01110 01111 10000 10001 10010 10011 10100

The function dec2bin converts a decimal number to a binary number in the form of a string. Here the decimal number must be a non-negative integer smaller than 2^{52}. The two syntaxes for dec2bin are,

b = dec2bin(a)

b = dec2bin(a,n)

b = dec2bin(a,n) gives the binary representation of the decimal integer a with at least n bits in the result.

```
>> dec2bin(123)
ans =
    1111011
>> dec2bin(123,4)
ans 1111011
>> dec2bin(123,10)
ans =
    0001111011
```

The binary number 1111011 exceeds 4 bits

10 bits

A MATLAB program to convert a decimal number to binary has been given below.

```
d=input('enter the number:');
i=1;
while d>0
a(i)=rem(d,2);
d=floor(d/2);
i=i+1;
end
b=fliplr(a);
disp('the  binary number is:');
b
enter the number:123
```

the binary number is:

```
b =
  1111011
```

The function bin2dec on the other hand converts a binary string to a decimal number. For example,

```
>> bin2dec('1111011')
ans =
    123
```

Here the binary is considered a string and should be enclosed in single inverted commas. The function bin2dec ignores any space (' ') in the input string.

For example,

```
>> bin2dec('  1111011   ')
ans =
   123
```

The space provided before and after the binary number is ignored.

The program which converts binary to decimal is given below.

```
binary='111';
decimal=0;
l=length(b);
for i=1: length(binary)
decimal=decimal+str2num(binary(i))*2^(l-1);
l=l-1;
end
disp('the decimal equivalent is:');
decimal
```

the decimal equivalent is:

```
decimal =
      56
```

3.17 ONES' COMPLIMENT

The ones' compliment of a binary number is defined as the number obtained by inverting all the bits in the binary representation of the number. The value of 0 is swapped for 1 and 1 is swapped for 0.

C = bitcmp(a) returns the bitwise complement of a, where a is an unsigned integer or an array of unsigned integers. In case of 8-bit arithmetic, the ones' complement of decimal88 is decimal 167.

```
>> C = bitcmp(uint8(88))
C =
  167
```

Let us check this by converting the decimal numbers to binary numbers.

```
>> dec2bin(88)
ans =
    1011000
>> dec2bin(167)
ans =
    10100111
```

As can be seen the ones' compliment of 1011000 (binary of 88) is 10100111(binary of 167)

01011000	binary of 88
10100111	binary of 167

Let us consider the decimal number 7 which is 0111 in binary.
So the ones' compliment of 0111 is

```
 0111
======
 1000
```

So its ones' compliment is 1000 in binary system which is 8 in decimal.

3.18 LOGICAL OPERATORS &, |, ~ AND xor

The logical operators are:

Logical Operator	Property	Defining a and b matrices	Output
& (logical and)	It returns 1(nonzero) for every element location that is true in both arrays and 0 for all other elements.	>> a=[1 1;0 0]; >> b=[1 0;1 0]; >> a a = 1 1 0 0 >> b b = 1 0 1 0	>> a&b ans = 1 0 0 0
| (logical or)	It returns 1(nonzero) for every element location that is true in either one or the other, or both arrays, and 0 for all other elements.		>> a|b ans = 1 1 1 0
~(logical not)	Complements each element of the input array.		>> ~a ans = 0 0 1 1
xor	Returns 1(nonzero) for every element location that is true in only one array and 0 for all other elements.		>> xor(a,b) ans = 0 1 1 0

The result of a logical operation is either 1 or 0. The result is integer type. For example, th logical & operator produces the value 1 if both operands have a nonzero value. In the case o any operand is 0 the ultimate value is 0. In case the first operand is 0 then the second operan is not evaluated. The operands are evaluated from left to right.

3.19 RELATIONAL OPERATORS

==	equal
~=	not equal
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

```
>> a=5;
>> b=[5 7 9;9 6 5];
>> a>=b
ans =
    1    0    0
    0    0    1
>> a==b
ans =
    1    0    0
    0    0    1
```

The relational operators in MATLAB are <, >, <=, >=, == and ~=. Relational operators between two arrays perform element-by-element comparisons and return a logical array of the same size. The elements are set to logical 1 (true) when the relation is true, and the elements are set to logical 0(false) when it is not true. The operators <, >, <=, and >= use only the real part of their operands for the comparison whereas the operators == and ~= test both real and imaginary parts.

A simple MATLAB program to find the maximum of two numbers is,

```
a=input('enter a:');
b=input('enter b:');
```

Contd.

Contd.

```
if a==b
disp('a=b')
end
if a>b
disp('a>b')
end
if b>a
disp('b>a')
end
```

enter a:6

enter b:3

a>b

3.20 THE ':' OPERATOR

The easiest way of making an array is by using the ':' operator. It is a very effective and fast method to create an array. For example,

```
>> a=1:10
a =
  1   2   3   4   5   6   7   8   9   10
```

This results in a row vector containing integers 1 to 10. The spacing between the integers is 1 by default.

```
>> b=1:2:10
b =
  1   3   5   7   9
```

results in a row vector containing integers lying between 1 to 10 with the spacing between the integers as 2. A negative step between the integers can also be used. For example,

```
>> c=10:-1:1
c =
  10   9   8   7   6   5   4   3   2   1
```

We can also use a fractional step. For example,

```
>> d=0:0.1:1
d =
  0   0.1000   0.2000   0.3000   0.4000   0.5000   0.6000   0.7000   0.8000   0.9000   1.0000
```

Few more operations using ':' operator are shown below.

>> a=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16];	>> a a = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	
>>sum(a(:))	ans = 136	Sum of all the elements in the array a.
>> a(:,2)	ans = 2 6 10 14	The column 2 in array a.
>> sum(a(:,2))	ans = 32	Sum of all the elements in column 2 in array a.
>> b=a; >> b(6,7) = 9; >> b	b = 1 2 3 4 0 0 0 5 6 7 8 0 0 0 9 10 11 12 0 0 0 13 14 15 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9	One can also store a value outside the matrix dimension. The size of the matrix automatically increases to store this value in the designated address.

3.21 USE OF TEMPORARY VARIABLE FOR SORTING AN ARRAY

```
n=input('enter the value of n:')
for i=1:n
    a(i)=input('enter value');
end
i=1;
for j=1:n
for i=1:(n-1)
if a(i)>a(i+1)
        temp=a(i);
```

Output

enter the value of n:10
n =
10
enter value1
enter value2
enter value4

Contd.

Contd.

```
        a(i)=a(i+1);
        a(i+1)=temp;

end
end
end
disp('The sorted array is:')
a
m=input('enter the value 1 if OK if NOT
  OK enter 0:')
for m=0
for j=1:n
for i=1:(n-1)
if a(i)>a(i+1)
        temp=a(i);
        a(i)=a(i+1);
        a(i+1)=temp;

end
end
end
end
disp('The sorted array is:')
a
```

```
enter value6
enter value8
enter value9
enter value2
enter value5
enter value6
enter value8
The sorted array is:
a =
  1   2   2   4   5   6   6
8   8   9
enter the value 1 if OK if NOT
OK enter 0:1
m =
   1
The sorted array is:
a =
  1   2   2   4   5   6   6
8   8   9
```

3.22 PRODUCT OF TWO MATRICES

A matrix is an array of numbers. A matrix multiplication falls in two categories. When a number is multiplied to a matrix we call the number a scalar, this is why this multiplication is called scalar multiplication. In a scalar multiplication a single number is multiplied with every entry of a matrix. On the other hand in order to multiply a matrix by another matrix you need to do the dot product of the rows and the columns.

Let a be a m by n matrix.

$$a = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{1(n-1)} \\ a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \cdots & \cdots & \cdots & \cdots \\ a_{(m-1)0} & a_{(m-1)1} & \cdots & a_{(m-1)(n-1)} \end{bmatrix}$$

Let b be a k by 1 matrix

$$b = \begin{bmatrix} b_{00} & b_{01} & \cdots & b_{1(l-1)} \\ b_{10} & b_{11} & \cdots & b_{1(n-1)} \\ \cdots & \cdots & \cdots & \cdots \\ b_{(k-1)0} & b_{(k-1)1} & \cdots & b_{(k-1)(l-1)} \end{bmatrix}$$

In order to find the product of matrices a and b, n=k and the size of the resultant matrix is m by l.

$$c = \begin{bmatrix} c_{00} & c_{01} & \cdots & c_{0(l-1)} \\ c_{10} & c_{11} & \cdots & c_{1(l-1)} \\ \cdots & \cdots & \cdots & \cdots \\ c_{(m-1)0} & c_{(m-1)1} & \cdots & c_{(m-1)(l-1)} \end{bmatrix}$$

Let us try to find out an element $c_{(m-1)0}$ in the matrix c. The element $c_{(m-1)0}$ as shown in below is equal to the sum of the products of the corresponding elements of the $(m-1)^{th}$ row elements of matrix a and 0th row of matrix b.

$$\begin{bmatrix} c_{00} & c_{01} & \cdots & c_{0(l-1)} \\ c_{10} & c_{11} & \cdots & c_{1(l-1)} \\ \cdots & \cdots & \cdots & \cdots \\ c_{(m-1)0} & c_{(m-1)1} & \cdots & c_{(m-1)(l-1)} \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{1(n-1)} \\ a_{10} & a_{11} & \cdots & a_{1(n-1)} \\ \cdots & \cdots & \cdots & \cdots \\ a_{(m-1)0} & a_{(m-1)1} & \cdots & a_{(m-1)(n-1)} \end{bmatrix}$$

$$\begin{bmatrix} b_{00} & b_{01} & \cdots & b_{1(l-1)} \\ b_{10} & b_{11} & \cdots & b_{1(n-1)} \\ \cdots & \cdots & \cdots & \cdots \\ b_{(k-1)0} & b_{(k-1)1} & \cdots & b_{(k-1)(l-1)} \end{bmatrix}$$

Therefore,

$c_{(m-1)0} = a_{(m-1)0}b_{00} + a_{(m-1)1}b_{10} + a_{(m-1)2}b_{20} + \ldots\ldots\ldots\ldots\ldots + a_{(m-1)(n-1)}b_{(k-1)0}$
as $n = k$,

$$c(m-1)0 = \sum_{t=0}^{t<n} a_{(m-1)t}b_{t0}$$

So for any element of the matrix c,

$$c_{ij} = \sum_{i=0}^{i<m} \sum_{j=0}^{j<l} \sum_{t=0}^{t<k} a_{it}b_{tj}$$

Program	Solution

m=input('enter the number of rows in matrix A:');
n=input('enter the number of columns in matrix A:');
k=input('enter the number of rows in matrix B:');
l=input('enter the number of columns in matrix B:');
for i=1:m
for j=1:n
A(i,j)=input('enter element:');
end
end
disp('the A matrix is:');
A
for i=1:k
for j=1:l
B(i,j)=input('enter element:');
end
end
disp('the B matrix is:');
B
 	
X=zeros(m,l);
 	
if(n~=k)
  disp('Error, not able to multiply matrices');
end
if(n==k)
for i=1:m;
for j=1:l;
for p=1:n;
  (i,j)=X(i,j)+ A(i,p)*B(p,j);
end
end
end
end
disp('the product of matrices A and B is:');
X
	enter the number of rows in matrix A:3
	enter the number of columns in matrix A:2
	enter the number of rows in matrix B:2
	enter the number of columns in matrix B:3
	enter element:1
	enter element:2
	enter element:3
	enter element:4
	enter element:5
	enter element:6
	the A matrix is:
	A =
	1 2
	3 4
	5 6
	enter element:1
	enter element:2
	enter element:3
	enter element:4
	enter element:5
	enter element:6
	the B matrix is:
	B =
	1 2 3
	4 5 6
	the product of matrices A and B is:
	X =
	9 12 15
	19 26 33
	29 40 51

The product of the two matrices can be obtained using the '*' operation.

```
>> a=[1 2;3 4;5 6];
>> a
a =
    1     2
    3     4
    5     6
>> b=[1 2 3;4 5 6];
>> b
b =
    1     2     3
    4     5     6
>> a*b
ans =
     9    12    15
    19    26    33
    29    40    51
```

The product can also be obtained using the mtimes function.

```
>> mtimes(a,b)
ans =
     9    12    15
    19    26    33
    29    40    51
```

3.23 DETERMINANT, FINDING DETERMINANT OF A SQUARE MATRIX

The determinant of the matrix is,

$$\det(A) = a_{j1}C_{j1} + a_{j2}C_{j2} + \ldots + a_{jn}C_{jn} \text{ for } j = 1, 2, \ldots, \text{or } m$$

Here a_{jk} are the elements of the matrix and C_{jk} are their corresponding cofactors. The determinant can be found by choosing all the elements of a particular row and considering their corresponding cofactors. j can be assigned any number from 1 to m, where m is the maximum number of rows in the matrix.

The determinant could also be found out by choosing all the elements of a particular column and considering their corresponding cofactors using the following equation,

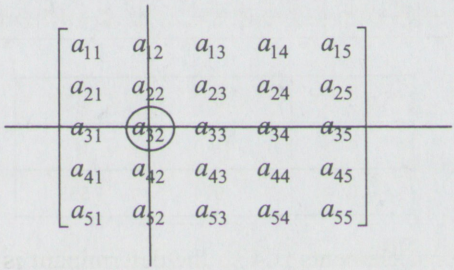
$$\det(A) = a_{1k}C_{1k} + a_{2k}C_{2k} + \ldots + a_{mk}C_{mk} \text{ for } k = 1, 2, \ldots, \text{or } n$$

Here k can be assigned any number from 1 to n, where n is the maximum number of columns in the matrix.

C_{jk} is known as the cofactor of a_{jk} and can be expressed in terms of M_{jk} which is the minor of a_{jk}.

$$C_{jk} = (-1)^{(j+k)}M_{jk}$$

M_{jk} is the determinant found by deleting all the elements of the jth row and k^{th} column of the matrix. In the example shown below, M_{32} of a 5 by 5 matrix would be,



The new matrix obtained by deleting all the elements of the 3rd row and 2nd column is,

$$\begin{bmatrix} a_{11} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{23} & a_{24} & a_{25} \\ a_{41} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{53} & a_{54} & a_{55} \end{bmatrix}$$

This is the minor or M_{32} of the element a_{32}.

In the above system of m equations if all the equations converge at a point then they will have a unique solution, x_{1s}, x_{2s}, x_{3s}.....and x_{ns}. This is a set of solution to the above system of linear equations as this set of solution satisfies all the m equations simultaneously. Below we have shown how the unique set of solution to a system of three equations satisfies all the three equations simultaneously.

In this process the upper triangle or the lower triangle elements are all made 0 and the determinant is found by finding the product of the diagonal elements of the new array. Let us consider a 3 by 3 square matrix.

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

i=0 j=0	i=0 j=1	I=0 j=2	i=1 j=0	i=1 j=1	i=1 j=2	i=2 j=0	i=2 j=1	I=2 j=2
	1/2–=0.5	½=0.5	2/3=0.666		1/3=0.333	0.5	3/8=0.375	
	a[1][0]=0	a[2][0]=0	a[0][0]=2		a[2][0]=0	a[0][0]=2	a[1][0]=3/8	
	a[1][1]=3/2=1.5	a[2][1]=1/2=0.5	a[0][1]=0		a[2][1]=0	a[0][1]=0	a[1][1]=3/2	
	a[1][2]=1/2=0.5	a[2][2]=3/2=1.5	a[0][2]=2/3=0.666		a[2][2]=4/3=1.333	a[0][2]=0	a[1][2]=0	

2 → 2 → 2	1 → 0 → 0	1 → 0.666 → 0
1 → 0 → 0	2 → 3/2 → 3/2	1 → ½ → 0
1 → 0 → 0	1 → 0 → 0	2 → 1.5 → 1.333

2	0	0
0	1.5	0
0	0	1.333

The product of the diagonal elements is 4 so the determinant is 4.
Now let us find the determinant of a 2 by 2 matrix.

$$\begin{bmatrix} 4 & 5 \\ 6 & 7 \end{bmatrix}$$

ir=0 jc=0	ir=0 jc=1	ir=1 jc=0	ir=1 jc=1
	1.5	–10	
	a[1][0]=0	a[0][1]=0	
	a[1][1]=-0.5	a[0][0]=4	

4 → 4	5 → 0
6 → 0	7 → –0.5

Now multiplying the elements of the main diagonal we get –2 which is the determinant of the matrix. The determinant of an array can be found using the det function shown below.

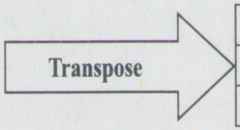
```
>> A=[2 1 1;1 2 1;1 1 2]

A =

     2     1     1
     1     2     1
     1     1     2
```

```
>> det(A)
ans =
    4
```

3.24 TRANSPOSE OF A MATRIX

The transpose of a matrix is formed by turning all the rows of the matrix into columns and all the column of the matrix to rows. When the rows of the matrix are rotated to columns the columns become automatically rotated to rows. The transpose of a matrix is written as A^T.

1	2	3
4	5	6
7	8	9
10	11	12



1	4	7	10
2	5	8	11
3	6	9	12

A MATLAB program to transpose a matrix is given below.

```
m=input('enter the value of rows:');
n=input('enter the value of columns:')
for i=1:m
for j=1:n
        a(i,j)=input('enter the element');
end
end
disp('the matrix is:')
a
for i=1:m
for j=i:n
        temp=a(i,j);
        a(i,j)=a(j,i);
        a(j,i)=temp;
end
end
disp('the transposed matrix is:')
  a
```

```
enter the value of rows:3
enter the value of columns:2
n =
   2
enter the element1
enter the element2
```

```
enter the element3
enter the element4
enter the element5
enter the element6
the matrix is:
a =
    1    2
    3    4
    5    6
the transposed matrix is:
a =
    1    3
    2    4
    5    6
>> a'
ans =
    1    2    5
    3    4    6
```

Transpose of an array can also be done using the commands,

```
>> a=[1 2;3 4;5 6];
>> a
a =
    1    2
    3    4
    5    6
>> a'
ans =
    1    3    5
    2    4    6
>> transpose(a)
ans =
    1    3    5
    2    4    6
```

3.25 EIGENVALUE AND EIGENVECTOR

Certain vectors x are in the same direction as Ax. These vectors are called the eigenvectors. If we multiply an eigenvector (x) by A then the vector Ax is a number λ multiplied by the original vector x. The eigenvalue λ tells us whether the vector x is stretched, shrunk, reversed or is unchanged. Almost all vectors are known to change direction when it is multiplied by A but the eigenvector is a special vector that retains the same direction as A. Most n by n square matrix haven eigenvector directions and n eigenvalues.

The command,

eig(A)

gives a column vector containing the eigenvalues.

```
>> A=[2 4;3 10];
>> eig(A)
ans =

   0.7085
  11.2915
```

With two output arguments the MATLAB function eig computes the eigenvectors and the eigenvalues. The eigenvalues are stored in a diagonal matrix.

```
>> [vv,vl]=eig(A)
vv =

  –0.9516  –0.3954
   0.3073  –0.9185

vl =

   0.7085        0
        0   11.2915
```

Here vv is the eigenvector and vl are the eigenvalues of A.

An eigenvector of a square matrix A is a non-zero vector x which when multiplied with A results in a constant multiplied by x. In other words,

$$Ax = \lambda x$$

In other words an eigenvalue and an eigenvector of a square matrix A are a scalar λ and a nonzero vector x respectively that satisfies the above condition $Ax = \lambda x$.

Here x is the eigenvector and λ is called the eigenvalue of the matrix A. The prefix eigen in the words eigenvector and eigenvalue is adopted from the German word eigen which means "self", "unique to", "peculiar to", or "belonging to."

$$Ax = \lambda x$$
$$(A - \lambda I)x = 0$$
$$\det(A - \lambda I)x = 0$$

Let us consider the matrix,

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

As $\det(A - \lambda I) = 0$

So,

$$\begin{bmatrix} 1-\lambda & 2 \\ 3 & 4-\lambda \end{bmatrix} = (1-\lambda)(4-\lambda) - 6 = 0$$

or,

$$\lambda^2 - 5\lambda - 2 = 0$$

or,

$$\lambda = \frac{5 \pm \sqrt{33}}{2}$$

or, $\lambda = 5.3723$ and -0.3723

Here $\lambda^2 - 5\lambda - 2$ is called the characteristic polynomial.

The same eigenvalues have been obtained using the MATLAB function eig,

```
>> A=[1 2; 3 4]
A =
   1   2
   3   4
>> eig(A)
ans =
  -0.3723
   5.3723
```

The eigenvectors can also be calculated using the equations,

$$(A - \lambda_1 I)x = 0$$

and

$$(A - \lambda_1 I)x = 0$$

For the eigenvalue –0.372

The eigenvector is (–0.825, 0.566)

and for the eigenvalue 5.372

The eigenvector is (–0.416, –0.909).

This can also be determined using the MATLAB function

```
>> A=[1 2;3 4]
A =
   1   2
   3   4
>> [evt,evl]=eig(A)
evt =
  -0.8246  -0.4160
   0.5658  -0.9094
evl =
  -0.3723        0
        0   5.3723
```

Here evt is the eigenvector and evl is the eigenvalue. The determination of the eigenvectors and eigenvalues of a matrix is very important in certain areas of physics and engineering. Each eigenvector is coupled with its corresponding eigenvalue. The decomposition of a square matrix into its eigenvalues and eigenvectors is known as eigen decomposition.

3.26 PALINDROMIC NUMBER, REVERSING A NUMBER, PALINDROME

A palindromic number is a number that remains same when the number is written forwards or backwards. The number could be in any base. Few examples of palindromic number are 0,1,2,3,4,5,6,7,8, 9,11,22,33,44,....etc. Here a program to reverse an integer and to check if it is a palindromic number has been given below.

Program	Output

`number=input('enter the number n: ');`
`k=input('enter the number of digits in n: ');`
`n=number;`
`for i=1:100`
`if n~=0`
`d=rem(n,10);`
`a(i)=d;`
`n = fix(n/10);`
`end`
`end`
	enter the number n: 121121
	enter the number of digits in n: 6
	the reverse of the number is:
	sum =
	121121
	the number palindromic!
	>>

Contd.

Contd.

`sum=0;`
`for i=1:k`
`  sum=sum+a(i)*power(10,k-i);`
`end`
`disp('the reverse of the number is:');`
`sum`
 	
`if (sum==number)`
`  disp('the number palindromic!')`
`else`
`  disp('the number is not palindromic')`
`end`	

`a = 'anutforajaroftuna';`
`b=a(end : -1 : 1);`
`disp('the string that has been enetered is: ');`
`a`
`disp('the reverse of the string that has been enetered is: ');`
`b`
`if a==b`
`disp('it is a palindrome!')`
`else`
`disp('it is not a palindrome!')`
`end`
	the string that has been enetered is:
	a =
	anutforajaroftuna
	the reverse of the string that has been enetered is:
	b =
	anutforajaroftuna
	it is a palindrome!

A palindrome is a word or phrase or any other sequence of symbols or elements whose meaning does not change when the word or phrase or the sequence of symbols is written in reverse direction. A palindrome could also be a number in which case it is known as a palindromic number. A few examples of palindromes are, radar, civic, kayak, rotor, etc.. Even a sentence can be a palindrome for example,"A nut for a jar of tuna".

3.27 IDENTITY MATRIX

An identity matrix of size n is a n by n square matrix whose diagonal elements are all 1 and other elements are 0.

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

The eye function generates the identity matrix in MATLAB.

For example, a scalar value *m* generates an *m*-by-*m* identity matrix. On the other hand a two-element vector, [*m*,*n*], generates an *m*-by-*n* unit-diagonal matrix.

Command	Output	Explanation
>> eye(3)	ans = 1 0 0 0 1 0 0 0 1	Generates a 3 by 3 matrix with all diagonal elements as 1.
>> eye(3,3)	ans = 1 0 0 0 1 0 0 0 1	Generates a 3 by 3 matrix with all diagonal elements as 1.
>> eye(3,2)	ans = 1 0 0 1 0 0	Generates a 3 by 2 unit-diagonal matrix.

Any matrix multiplied by an identity matrix gives the same matrix. For example,

```
>> A=[1 2 3;4 5 6;7 8 9];
>> A*eye(3,3)

ans =

   1   2   3
   4   5   6
   7   8   9
```

3.28 SPARSE MATRIX

In numerical analysis a sparse matrix is a matrix which is populated primarily with zeros. On the other hand if a larger number of elements are nonzero numbers then it is commonly referred to as a dense matrix. The fraction of zero elements is called the sparsity. The fraction of non-zero elements in a matrix is called the density. In sparse matrix special techniques allows us to take advantage of the large number of zeros. In MTALAB the sparse function generates matrices in the MATLAB sparse storage organization. If a matrix A is stored in ordinary or dense format, then the command

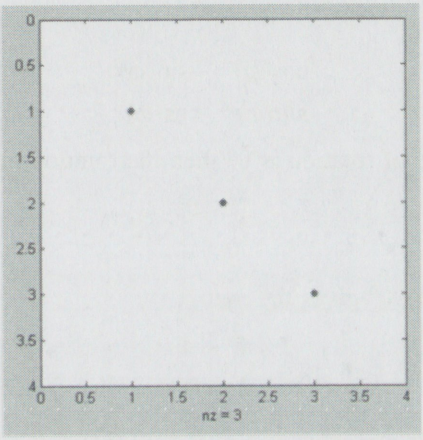
S = sparse(A)

creates a copy of the matrix A stored in sparse format. For example,

```
>> A=[1 0 0 0;4 0 0 9;3 7 0 0;1 0 4 0]
A =
     1     0     0     0
     4     0     0     9
     3     7     0     0
     1     0     4     0
>>sparse(A)
ans =
   (1,1)        1
   (2,1)        4
   (3,1)        3
   (4,1)        1
   (3,2)        7
   (4,3)        4
   (2,4)        9
>> whos
  Name      Size     Bytes  Class     Attributes
   A        4x4        128  double
   ans      4x4        104  double    sparse
```

It can be noticed that the sparse matrix need less space to store the matrix in this format. It should be noted that in a sparse matrix majority of the elements are zero and only then it is advantageous to form a sparse matrix of a given matrix and store in this format. This method is not the best available method for creating a sparse matrix as it can be very time-consuming to create the matrix in a dense format if A is large matrix. On the other hand the command S = sparse(m, n) creates an m × n zero matrix in sparse format. Entries can then be added one after another. For example,

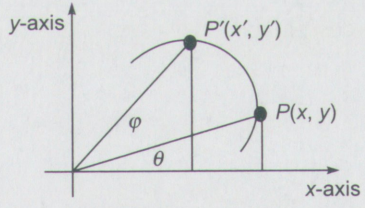
```
>> A=sparse(3,3);
>> A(1,1)=2;
>> A(2,2)=2;
>> A(3,3)=2;
>> A
A =
   (1,1)        2
   (2,2)        2
   (3,3)        2
>> spy(A)
```



The sparsity pattern of matrix A

3.29 ROTATION MATRIX

A point having coordinates $P(x, y)$ is rotated through an angle ϕ to the point $P'(x', y')$.



Rotation of a Point

For the point P,

$$x = a\cos(\theta)$$
$$y = a\sin(\theta)$$

and for the point P',

$$x' = a\cos(\theta + \varphi) = a[\cos\theta\cos\varphi - \sin\theta\sin\varphi]$$
$$y' = a\sin(\theta + \varphi) = a[\sin\theta\cos\varphi + \cos\theta\sin\varphi]$$

As,

$$x = a\cos(\theta)$$
$$y = a\sin(\theta)$$

So,

$$x' = a\cos(\theta + \varphi) = a[\cos\theta\cos\varphi - \sin\theta\sin\varphi] = x\cos(\varphi) - y\sin(\varphi)$$
$$y' = a\sin(\theta + \varphi) = a[\sin\theta\cos\varphi + \cos\theta\sin\varphi] = y\cos(\varphi) + x\sin(\phi)$$

or

$$x' = x\cos(\varphi) - y\sin(\varphi)$$
$$y' = x\sin(\phi) - y\cos(\varphi)$$

So the rotation matrix is,

$$\begin{vmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{vmatrix}$$

As can be seen if the angle of rotation is 0° then the rotation matrix is an identity matrix.

$$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

Whereas a rotation through 90° gives the matrix,

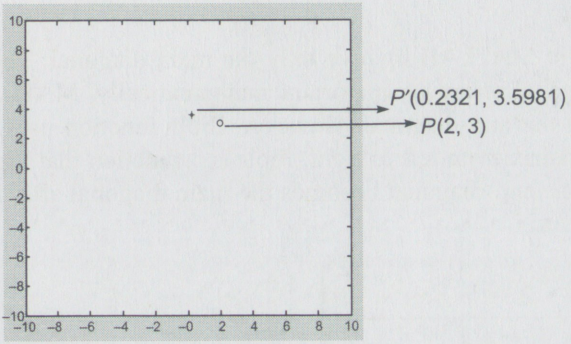
$$\begin{vmatrix} 0 & -1 \\ 1 & 0 \end{vmatrix}$$

So the point (x', y') are now,

$$x' = -y$$
$$y' = x$$

Let us now try to rotate a point $P(2,3)$ through 30°

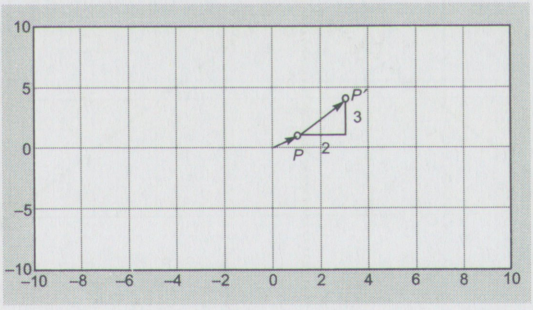
```
>> phi=(30/180)*pi;
>> c=[cos(phi)–sin(phi);sin(phi) cos(phi)];
>> p=[2;3];
>> c*p

ans =

    0.2321
    3.5981

>> 2*cos(phi)–3*sin(phi)

ans =

    0.2321

>> 2*sin(phi)+3*cos(phi)

ans =

    3.5981

>> x=2;
>> y=3;
>> plot(x,y,'ro')
>> hold on
>> xdash=0.2321;
ydash=3.5981;
plot(xdash,ydash,'r*')
>>axis([–10 10 –10 10])
```



3.30 TRANSLATION MATRIX

If a poin is at $P(1,1)$ and is translated by a vector $2i + 3j$ the coordinates of the new point P' is,

$$P' = (i + j) + (2i + 3j) = 3i + 4j$$



3.31 trace

trace gives us the summation of the diagonal elements of a matrix.

trace is actually,

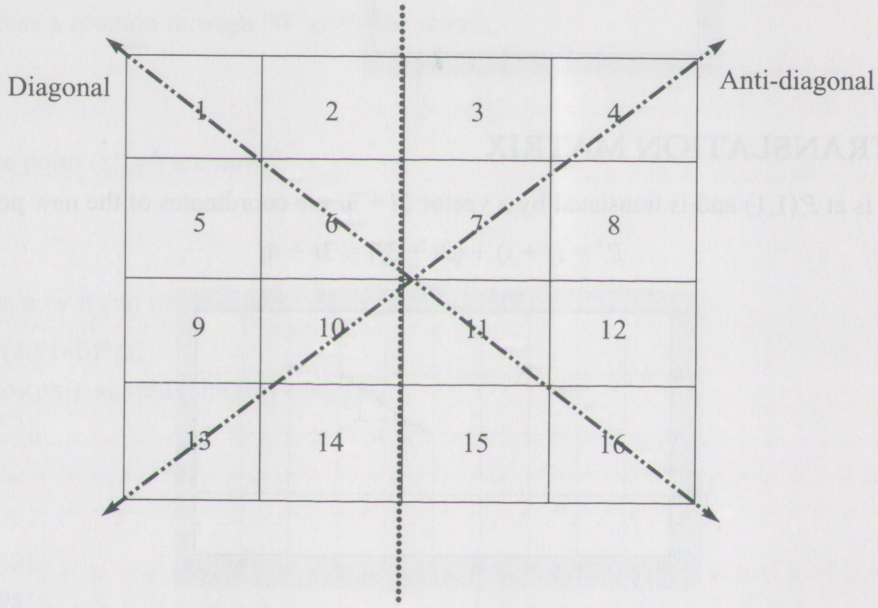
sum (diag(a));

For example,

```
>> a=[1 2 3;4 5 6;7 8 9]
a =
    1    2    3
    4    5    6
    7    8    9
>>trace(a)
ans =
    15
```

3.32 fliplr

Most of the functions in MATLAB involve only the main diagonal. The other diagonal of the matrix or the anti-diagonal is not so important mathematically. MATLAB does not have any in-built function using the anti-diagonal. However, fliplr function uses the anti-diagonal. The fliplr function flips a matrix from left to right. fliplr is a function that flips the matrix by mirror reflection. As a result the anti-diagonal becomes the main diagonal after flipping the matrix. Let us consider a 4 by 4 matrix



The diagonal and anti-diagonal of a matrix

```
>> a=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16]
a =
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
>> fliplr(a)
ans =
     4     3     2     1
     8     7     6     5
    12    11    10     9
    16    15    14    13
```

```
>> diag(a)
ans =
     1
     6
    11
    16
>> diag(fliplr(a))
ans =
     4
     7
    10
    13
```

4

CHAPTER

Plotting Using MATLAB

Srinivas Ramanujan *(22nd December 1887 – 26th April 1920) was a mathematician who, with almost no formal training in mathematics, made remarkable contributions to areas like mathematical analysis, number theory, series, continued and several others. Ramanujan's talent was considered in the same league as legendary mathematicians such as Gauss, Euler, Cauchy, Newton and Archimedes by English mathematician like G.H. Hardy. During his short lifetime, Ramanujan compiled nearly 3900 results and most of his claims have now been proven correct. One of the great anecdotes of Ramanujan is the Ramanujan Hardy number 1729. Hardy arrived at Ramanujan's residence in a cab numbered 1729. Hardy commented that the number 1729 seemed to be uninteresting. Ramanujan refuted and stated on the spot that it was actually a very interesting number mathematically, being the smallest natural number that could be represented in two different ways as a sum of two cubes.*

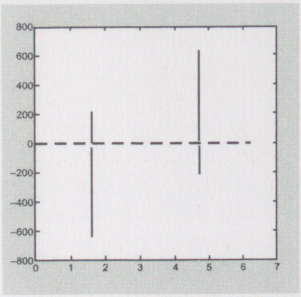
$$1729 = 1^3 + 12^3 = 9^3 + 10^3$$

Such observations could only be made by a mathematical genius. December 22nd has been declared as the National Mathematics Day."

4.1 PLOTTING A LINE

4.1.1 Line Style

Line style can be set using any of the specifiers mentioned in the Table below.

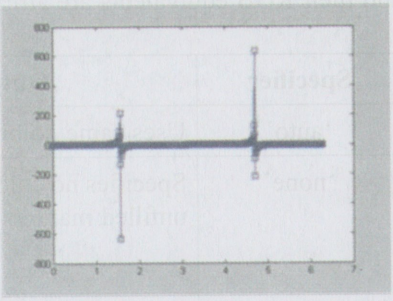
Specifier	Line Style	Example
-	Solid line also the default	>> theta=linspace(0,2*pi,1000); >> y=tan(theta); >> plot(theta,y,'--')  Plots a tan(theta) curve from 0 to 2π using dashed lines.
--	Dashed line	
:	Dotted line	
-.	Dash-dot line	
none	No line	

4.1.2 LineWidth

The width of a line can be set by 'LineWidth' followed by a scalar specifying the width of the line. The width value is specified in points where each point equals 1/72 of an inch (0.03527 cm). If any marker is displayed, then LineWidth also sets the width of the marker edges.

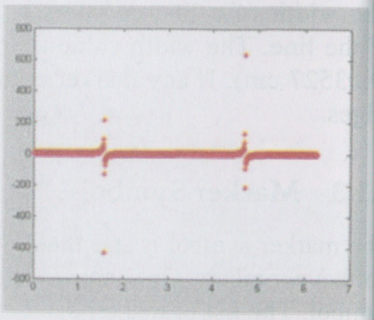
4.1.3 Marker Symbol

The marker symbol is specified by 'Marker' and a marker specifier. By default a plot line does not have markers. One can add markers at each data point along the line by specifying a marker symbol. The list of marker symbols supported by MATLAB is as follows.

Specifier	Symbol	Example
'o'	Circle	> plot(x,theta,'r*') >> y=tan(theta); >> plot(theta,y,'r*') >> plot(theta,y,'marker','s')  Plots a tan(theta) curve from 0 to 2π using squares. Here the marker has been specified after mentioning 'marker' followed by the specifier > plot(x,theta,'r*') >> y=tan(theta); >> plot(theta,y,'r*')
'+'	Plus sign	
'*'	Asterix	
'.'	Point	
'x'	Cross	
'square' or 's'	Square	
'diamond' or 'd'	Diamond	
'^'	Upward-pointing triangle	
'v'	Downward-pointing triangle	
'>'	Right-pointing triangle	
'<'	Left-pointing triangle	
'pentagram' or 'p'	Five-pointed star (pentagram)	
'hexagram' or 'h'	Six-pointed star (hexagram)	
'none'	No marker	

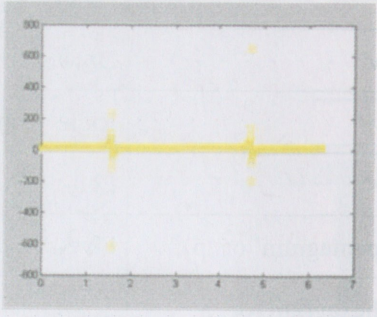
Contd.

Contd.

		 Plots a tan (theta) curve from 0 to 2π using squares. Here the marker has not been specified after mentioning 'marker' followed by the specifier.

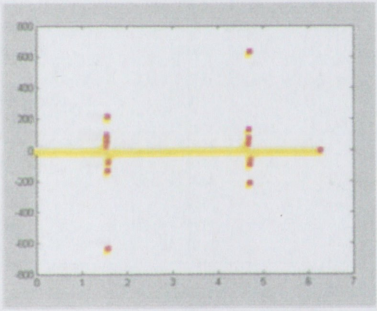
4.1.4 Marker Edge Color

The marker edge color is specified by 'MarkerEdgeColor' and a color value. The color value can be one of the supported strings or an RGB vector. The lists of the predefined string colors and their RGB equivalents are already listed in colors section.

Specifier	Output	Example
'auto'	Uses same color as line color	>> theta=linspace (0,2*pi,1000); >> y=tan (theta); >> plot (theta,y,'s','markeredgecolor','y')  Plots a tan (theta) curve from 0 to 2π using square markers that have yellow edge color.
'none'	Specifies no color, which makes unfilled markers invisible	

4.1.5 Marker Face Color

The marker face color can be set by 'MarkerFaceColor' and a color value. MarkerFaceColor sets the fill color for markers that are closed shapes. The circle, square, diamond, pentagram, hexagram and the four triangles have closed shapes. The colour value can be one of the supported strings as mentioned in the Table of predefined colors or an RGB vector.

Specifier	Output	Example
'auto'	Uses the value of the axes color property to fill the markers. If the axes color property is 'none', then plot uses the figure color value instead.	>> plot(x,y,'r*') >> theta=linspace(0,2*pi,1000); >> y=tan(theta); >> plot(theta,y,'s','markeredgecolor','y','marker facecolor','r')  Plots a tan (theta) curve from 0 to 2π using square markers that have yellow edge colour and face color red.
'none'	Makes the interior of the marker transparent, allowing the background to show through. This is also the default.	

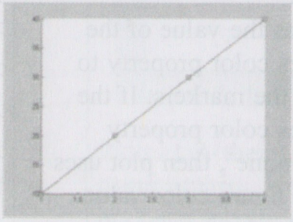
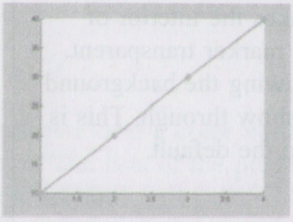
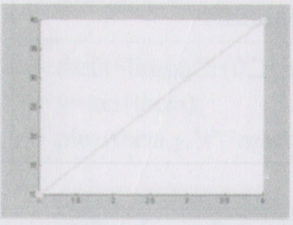
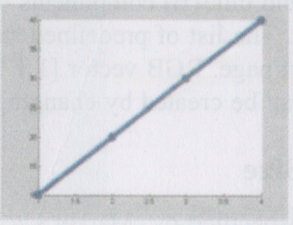
For an RGB vector a three-element row vector whose elements specify the intensities of the red(R), green(G) and blue(B) components of the color are specified. The intensities must be in the range of 0 to 1. The list of predefined string colors and their RGB equivalents are given in Table given on next page. RGB vector [1 1 1] gives white and RGB vector [0 0 0] gives black. All other colours can be created by changing the values of R , G and B between 0 and 1.

4.1.6 Marker Size

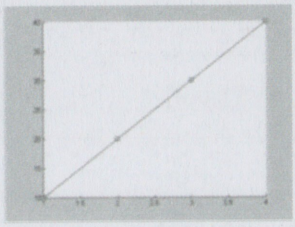
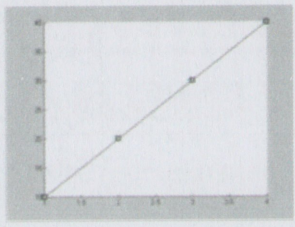
The marker size is specified by 'MarkerSize' and a scalar in points. Marker size 6 is the default marker size where 1 point = 1/72 inch (0.03527 cm). Marker size is a scalar which specifies the size of the marker, in points.

4.1.7 Examples Illustrating the Use of Various Specifiers in Plotting a Line

This is also shown below.

MATLAB Command to set the (x, y) coordinates of various points	MATLAB Command to draw a line through various (x, y) points	Output	Explanation
>> x=[1 2 3 4]; >> y=[10 20 30 40];	>> line(x,y,'LineWidth',2,'Color',[0.5 0.5 0.5],'marker','s')		Marker square, colour [0.5 0.5 0.5] , Line width 2
	>> line(x,y,'LineWidth',4,'Color',[0.7 0.7 0.7],'marker','d') >>		Marker diamond, colour [0.7 0.7 0.7] , Line width 4
	>> line(x,y,'LineWidth',6,'Color',[0.9 0.9 0.9],'marker','o') >>		Marker circle, colour [0.9 0.9 0.9] , Line width 6
	>> line(x,y,'LineWidth',8,'Color',[0.1 0.4 0.7],'marker','o') >>		Marker circle, colour [0.1 0.4 0.7 , Line width 8

The properties of the marker can also be changed. For example,

>> x=[1 2 3 4]; >> y=[10 20 30 40];	>> line(x,y,'LineWidth',2,'Color',[0.5 0.5 0.5],'marker','s','markersize',6) >>	
	>> line(x,y,'LineWidth',2,'Color',[0.5 0.5 0.5],'marker','s','markersize',6,'Marker Edge Color','k','MarkerFaceColor',[0.5 1 0.5])	

.2 COLORS IN MATLAB

Color specification for graphics in MATLAB can be done in three ways. They are:

(i) RGB triple
(ii) Short name
(iii) Long name

There are eight predefined colors in MATLAB. The RGB triple is a three-element row vector whose elements specify the intensities of the red (R), green (G) and blue (B) components of the color. The intensities must be in the range 0 to 1. The short names and long names are MATLAB strings that specify one of eight predefined colors.

The eight predefined colors in MATALAB are listed in the following Table.

RGB Value	Short name or colour code in MATLAB	Colour	Name of the colour
[1 1 0]	y		Yellow
[1 0 1]	m		Magenta

Contd.

Contd.

[0 1 1]	c		Cyan
[1 0 0]	r		Red
[0 1 0]	g		Green
[0 0 1]	b		Blue
[1 1 1]	w		White
[0 0 0]	k		Black

4.3 ginput

ginput stands for graphical input from mouse or cursor. The various syntaxes for ginput are:

[x, y] = ginput(n)

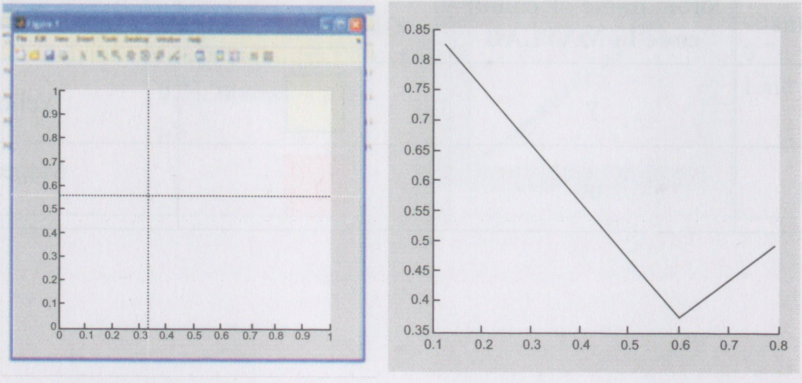
[x, y] = ginput

[x, y,button] = ginput(...)

It is possible to select any number of points from a figure by using the ginput function. Th coordinates of the chosen points are stored in the matrix [x, y]. A line can then be drawn throug these points.

For example,

```
>> [x, y]=ginput(4);
>> line(x, y)
```



Selecting points using ginput

The points are chosen by clicking the mouse after selecting a point by using the cross hairs x, y] = ginput collects an unlimited number of points until you press the return key is pressed. For example,

```
>> [x,y]=ginput
x =
    0.0887
    0.3237
    0.5565
    0.8099
y =
    0.8787
    0.5892
    0.4371
    0.3523
```

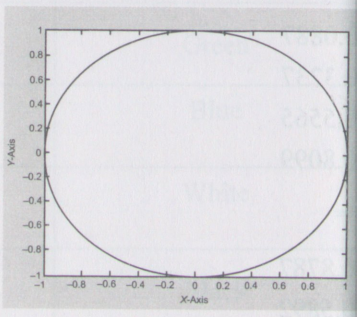
x, y, button] = ginput(...) returns the *x*-coordinates, the *y*-coordinates, and the button or key designation. Button is a vector of integers indicating which mouse buttons you pressed (1 for left, 2 for middle, 3 for right), or ASCII numbers indicating which keys on the keyboard you pressed.

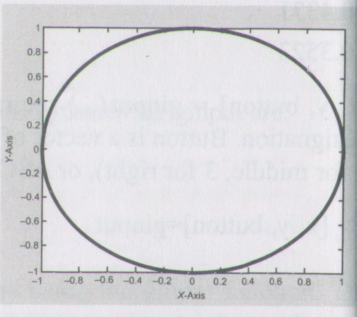
```
>> [x, y, button]=ginput
x =
    0.1233
    0.2385
    0.4689
    0.6855
y =
    1.0015
    0.7003
    0.4547
    0.2266
button =
    1
    3
    3
    1
```

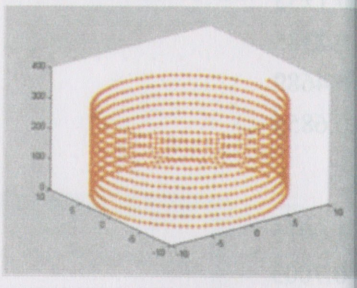
4.4 plot and plot3

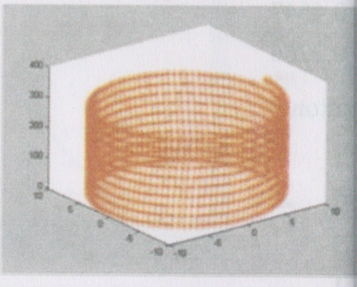
plot is used to get a 2-dimensional line plot whereas plot3 is used to get a 3-dimensional li plot. A few examples of plots using plot and plot3 are given below.

plot/plot3	MATLAB Program	Output

plot(x,y)	`r=1;`
	`theta=0:0.01:2*pi;`
	`x=r*cos(theta);`
	`y=r*sin(theta);`
	`plot(x,y)`
	`xlabel('X-Axis')`
	`ylabel('Y-Axis')`
		

plot(x,y,'Property', value)	`r=1;`
	`theta=0:0.01:2*pi;`
	`x=r*cos(theta);`
	`y=r*sin(theta);`
	`plot(x,y,'linewidth',8)`
	`xlabel('X-Axis')`
	`ylabel('Y-Axis')`
		

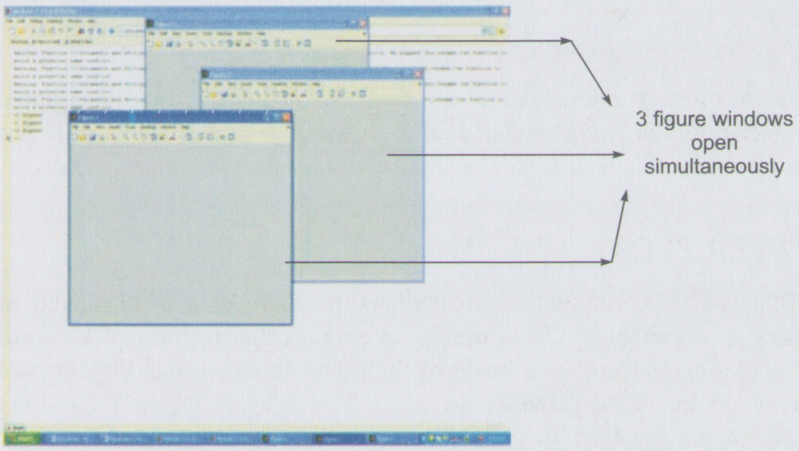
plot3(x,y,z)	`theta=linspace(0,20*pi,1000);`
	`x=10*cos(theta);`
	`y=10*sin(theta);`
	`z=5*(theta);`
	`plot3(x,y,z,'r*');`
		

	`theta=linspace(0,20*pi,1000);`
	`x=10*cos(theta);`
	`y=10*sin(theta);`
	`z=5*(theta);`
	`plot3(x,y,z,'r*','markersize', 12);`
		

4.5 figure

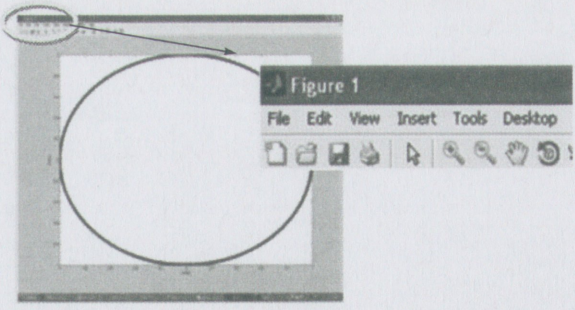
One can open as many figure windows as he wants. For example, it is possible to have 3 figure windows at a time by giving the command,

```
>> figure
>> figure
>> figure
```



We can see three figure windows here. In the example below two figures have been plotted in two different figure windows.

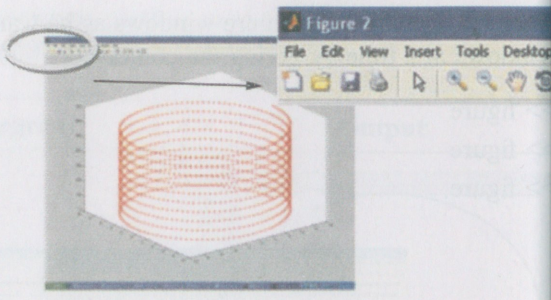
```
r=1;
theta=0:0.01:2*pi;
x=r*cos(theta);
y=r*sin(theta);
figure(1)
plot(x,y,'linewidth',8)
xlabel('X-Axis')
ylabel('Y-Axis')
theta=linspace(0,20*pi,1000);
x=10*cos(theta);
y=10*sin(theta);
```



Contd.

Contd.

```
z=5*(theta);
figure(2)
plot3(x,y,z,'r*','markersize',
12);
```



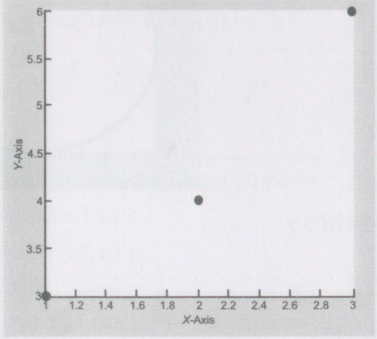
If two different figure windows would not have been chosen using the commands figure(1) and figure(2) then after the circle had been drawn it would be erased and a helix would be drawn in the same window.

4.6 SCATTER PLOT

The scatter plot is a type of mathematical diagram where data is displayed in a Cartesian coordinate using two variables. One variable determines the position of x-coordinate and the other variable determines the y-coordinate of the point. In cases of 3-dimensional plots where three axes are involved, three variables are used. The three variables act as the x-coordinate, y-coordinate and z-coordinate. This type of plot is also called scattergraph or scatterdiagram.

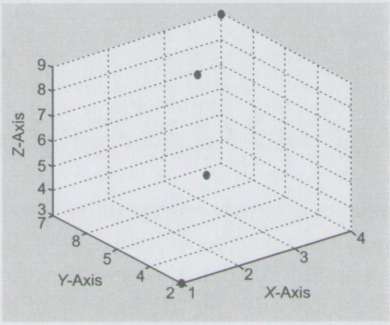
The scatter plot,

```
>> x=[1 2 3];
>> y=[3 4 6];
>> scatter(x,y,'filled')
>> xlabel('X-Axis')
>> ylabel('Y-Axis')
```



A scatter plot

```
>> x=[1 2 3 4];
>> y=[3 4 6 7];
>> z=[3 6 8 9];
>> scatter3(x,y,z,'filled')
>> xlabel('X-Axis')
>> ylabel('Y-Axis')
>> zlabel('Z-Axis')
```

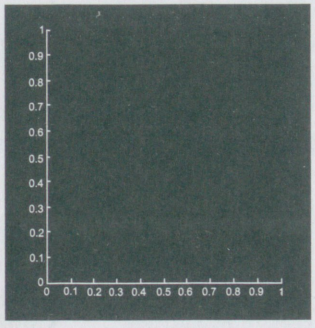
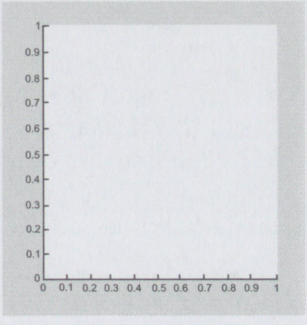
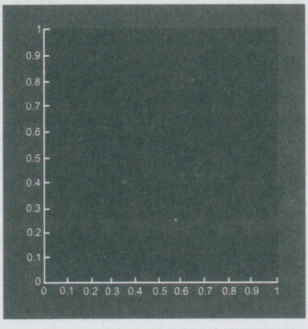


A scatter3 plot

Note that the scatter(x, y, 'filled') orscatter3(x, y, z, 'filled') fills the marker.

4.7 colordef

The function colordef allows us to select either black or white background for the graphics display. The axis lines and labels are set automatically in order to contrast with the background colour that has been chosen. The three different options are as follows:

>> colordef none >> axis ans = 0 1 0 1	>> colordef white >> axis ans = 0 1 0 1	>> colordef black >> axis ans = 0 1 0 1
		

colordef none sets the figure coloring to that used by MATLAB Version 4. Here the axis background color is set to none and the figure background coloring is set to black. This makes the axis background and the figure background colors the same.

colordef white sets the axis background color to white and the figure background color to light gray. The axis lines and labels are set to black.

colordef black sets the axis background color to black and the figure background color to black. The axis lines and labels are set to white.

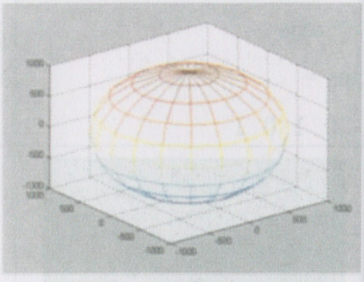
4.8 MESH PLOT

MATLAB defines a surface by the z-coordinates of points above a grid in the x-y plane. These adjacent points are connected using straight lines. The mesh plotting functions gives a 3-dimensional plot of the surface. A mesh plot produces wireframe surfaces that color only the lines connecting the defining points.

A function having two variables is defined. For example, $Z = f(X,Y)$. The X and Y matrices consisting of repeated rows and columns, respectively, over the domain of the function are generated. The value of Z is evaluated using the X and Y values and a graph of the function is plot. The meshgrid function transforms the domain specified by a single vector or two vectors, X and Y, into matrices, X and Y for use in evaluating functions of two variables. The rows of X are copies of the vector x and the columns of Y are copies of the vector y.

The various syntaxes for mesh plot are given below.

MATLAB Program	Output

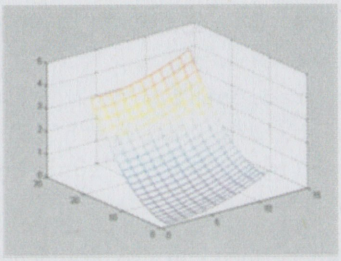
`for i=1:100`
`[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a=10*i*cos(v).*sin(u);`
`b=10*i*sin(v).*sin(u);`
`c=10*i*cos(u);`
`mesh(a,b,c);`
`end`
 	
Here the mesh(X,Y,Z) synatx has been used for getting a 3-dimensional plot. mesh(X,Y,Z) draws a wireframe mesh whose color is determined by the value of Z. Thus, the color is proportional to surface height. If X and Y are vectors whose lengths are, length(X) = n and length(Y) = m, then [m, n] = size(Z). In this case, $(X(j), Y(i), Z(i,j))$ are the intersections of the wireframe grid lines. X corresponds to the columns and Y corresponds to the rows of Z. If X and Y are matrices, $(X(i,j), Y(i,j), Z(i,j))$ are the intersections of the wireframe grid lines.
	

Contd.

Contd.

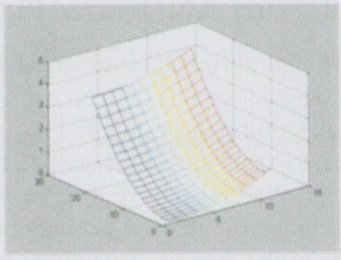
```
for i=1:100
[u,v] = meshgrid((0:0.1:1),(0:0.1:2));
z=u.*u+v.*v;
mesh(z)
end
```

mesh(Z) draws a wireframe mesh using X and Y. length(X) = n and length(Y) = m and [m,n] = size(Z). The height, Z, is a single-valued function defined over a rectangular grid. Colour is proportional to surface height.



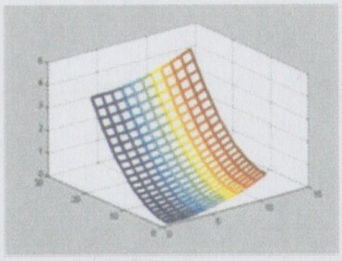
```
for i=1:100
[u,v] = meshgrid((0:0.1:1),(0:0.1:2));
z=u.*u+v.*v;
c=gradient(z);
mesh(z,c)
end
```

mesh(...,C) draws a wireframe mesh whose colour is determined by the matrix C.A linear transformation on the data in C is performed by MATLAB to obtain colors from the current colormap. If X, Y, and Z are matrices, they must have the same size as the matrix C.



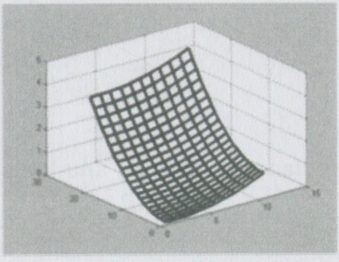
```
for i=1:100
[u,v] = meshgrid((0:0.1:1),(0:0.1:2));
z=u.*u+v.*v;
c=gradient(z);
mesh(z,c,'linewidth',3)
end
```

mesh(...,'Property', Value,...) sets the value of the specified surface property. Multiple property values can be set with a single statement. Here line width has been set to 3.



```
for i=1:100
[u,v] = meshgrid((0:0.1:1),(0:0.1:2));
z=u.*u+v.*v;
c=gradient(z);
mesh(z,c,'linewidth',3,'EdgeColor','black')
end
```

Here line width has been set to 3 and the edge color has been set to black.

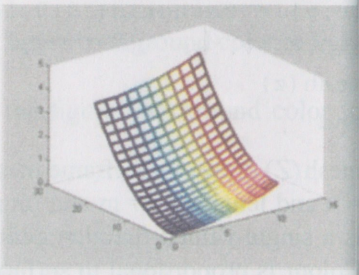


Contd.

Contd.

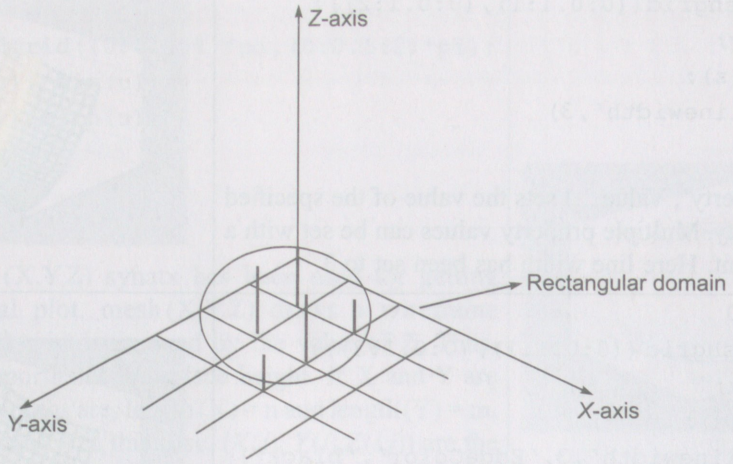
```
for i=1:100
[u,v] = meshgrid((0:0.1:1),(0:0.1:2));
z=u.*u+v.*v;
c=gradient(z);
mesh(z,c,'linewidth',3)
camlight left; lighting phong
end
```

Other effects can also be added to the mesh plot. For example here lighting has been added.



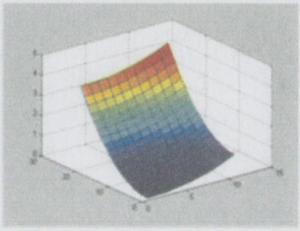
4.9 surf AND surfc PLOTS

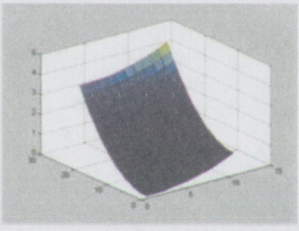
surf and surfc plots are used to view mathematical functions over a rectangular region. A rectangular area is defined using x and y and the parametric equation between x, y and z is used to determine the value of z for a particular value of x and y. z is plot for the various values of x and y. surf and surfc create colored parametric surfaces specified by X, Y and Z whose color is specified by Z or C. It is possible to create a shaded surface whose color is defined by C. A rectangular domain is shown below formed by a range of x values and a range of y values. The size of the domain depends on the length of x and length of y. The corresponding z values are calculated using the parametric equation relating x, y and z. Thus a surface is obtained which shows the height of z for given values of x and y.

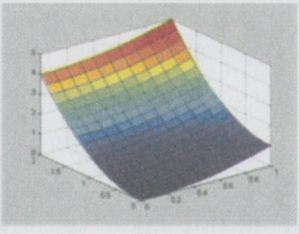


A rectangular domain and the surface formed from the parametric equation relating x, y and z

The various syntaxes for surf and surfc plot are given below with example and explanation.

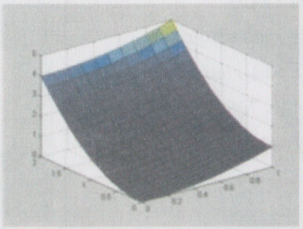
Syntax	MATLAB Program	Output	Explanation
surf(Z)	for i=1:100 [x,y] = meshgrid ((0:0.1:1),(0:0.1:2)); z=x.*x+y.*y; surf(z) end		surf(Z) creates a three-dimensional shaded surface from the *z* components in matrix Z. Here x=1:n and y=1:m, where [m,n] = size(Z). The height, Z, is a single-valued function defined over a rectangular grid. Z specifies the color data as well as the surface height. As a result color is proportional to the surface height.

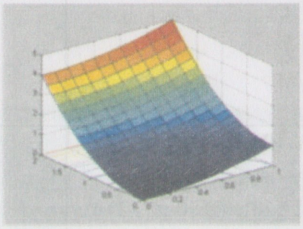
surf(Z,C)	for i=1:100
	[x,y] = meshgrid
	((0:0.1:1),(0:0.1:2));
	z=x.*x+y.*y;
	c=exp(z);
	surf(z,c)
	end
			surf(Z,C) plots the height of Z, a single-valued function defined over a rectangular grid, and uses the matrix C, assumed to be the same size as Z, to color the surface.

surf(X,Y,Z)	for i=1:100 [x,y] = meshgrid ((0:0.1:1),(0:0.1:2)); z=x.*x+y.*y; surf(x,y,z) end		surf(X,Y,Z) creates a shaded surface using Z for the color data as well as surface height. X and Y are vectors or matrices defining the x and y components of a surface. If X and Y are vectors, length(X)=n and length(Y)=m, where [m,n]=size(Z). In this case, the vertices of the surface faces are *(X(j), Y(i), Z(i,j))* triples. In order to create X and Y matrices for an arbitrary domain meshgrid function can be used.

Contd.

Contd.

`surf(X,Y, Z,C)`	`for i=1:100`
	`[x,y] = meshgrid ((0:0.1:1),(0:0.1:2));`
	`z=x.*x+y.*y;`
	`c=exp(z);`
	`surf(x,y,z,c)`
	`end`
			surf(X,Y,Z,C) creates shaded surface whos color is defined by C MATLAB performs linear transformatio on this data to obtai colors from the curren colormap.

`surfc(...)`	`for i=1:100`
	`[x,y] = meshgrid ((0:0.1:1),(0:0.1:2));`
	`z=x.*x+y.*y;`
	`surfc(x,y,z)`
	`end`
			surfc(...) draws contour plot beneat the surface.

Using

surf(...,'*Property*', Value)

the properties of the surface can be modified. This is the same as in the case of mesh plot.

surf(axes_handles,...)

and surfc(axes_handles,...) plot into the axes with handle axes_handle instead of the current axe
h = surf(...)

and h = surfc(...) return a handle to a surface plot graphics object.

4.10 THE BARNSLEY'S FERN PROBLEM

There are almost 12000 species of plant belonging to the botanical group known as Pteridophyt A fern belong to this group. They have stems, leaves and roots like other vascular plants. Fer do not have seed of flowers and reproduce through spores. Ferns do not have any economi importance but they have been a subject of research for their ability to remove chemical pollutan from air. The Barnsley Fern is a fractal named after the British Mathematician, Michael Barnsle He first described it in his book, Fractals Everywhere. He made it look similar to the Blac Splenwort, Asplenium adiantum-nigrum.



Fern collected from a bush in a hill

The transformations that create the fern are of the form f(x,y)=A*[x,y]+C. The total number f such transformations are four.

The Barnsley's fern problem uses four transformations. The formula for each transformation

$$f(x, y) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

Barnsley showed that the code for his Black Spleenwort fern fractal is the the matrix of alues in the Table given below.

	a	b	c	d	e	f	p = probability factor
f1	0	0	0	0.16	0	0	0.01
f2	0.85	0.04	−0.04	0.85	0	1.6	0.85
f3	0.2	−0.26	0.23	0.22	0	1.6	0.07
f4	−0.15	0.28	0.26	0.24	0	0.44	0.07

The probabilities with which each of the transformation is applied is given by f and sum 1+f2+f3+f4.

The values in columns "a" to "f" are the coefficients of the equation. The first point is the rigin which in our case is (0, 0). Any new point is iteratively computed by randomly applying ne of the following four coordinate transformations.

Transformation	Equations for transformations	Probability of concurrence
f_1	$x_{n+1} = 0$ $y_{n+1} = 0.16\ y_n$.	This coordinate transformation is chosen 1% of the total number of times. Probability factor = 0.01.

Contd.

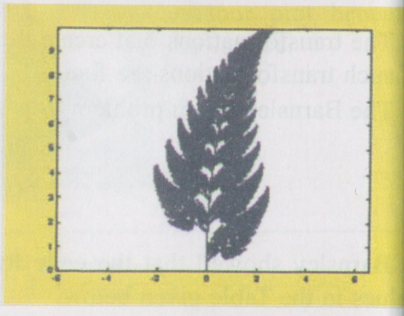
Contd.

f_2	$x_{n+1} = 0.85\, x_n + 0.04\, y_n$ $y_{n+1} = -0.04\, x_n + 0.85\, y_n + 1.6$	This coordinate transformation chosen 85% of the total number times.
f_3	$x_{n+1} = 0.2\, x_n - 0.26\, y_n$ $y_{n+1} = 0.23\, x_n + 0.22\, y_n + 1.6$	This coordinate transformation chosen 7% of the total number times.
f_4	$x_{n+1} = -0.15\, x_n + 0.28\, y_n$ $y_{n+1} = 0.26\, x_n + 0.24\, y_n + 0.44$	This coordinate transformation chosen 7% of total number of time

Program

```
 k=10000;
f1=[0 0 ; 0.0 0.16];
f2=[0.85 0.04 ; -0.04 0.85];
f3=[0.2 -0.26 ; 0.23 0.22];
f4=[-0.15 0.28 ; 0.26 0.24];
c1=[0 ; 0];
c2=[0 ; 1.6];
c3=[0 ; 1.6];
c4=[0 ; 0.44];

p1=0.01;
p2=0.85;
p3=0.07;
p4=0.07;

x(1)=0;
y(1)=0;

i=[0;0];
for n=2:k
  k=rand;
if k<p1
     i=f1*i+c1;
elseif k<p1+p2
     i=f2*i+c2;
elseif k<p1+p2+p3
     i=f3*i+c3;
else
 i=f4*i+c4;
end
```

Output



Cont

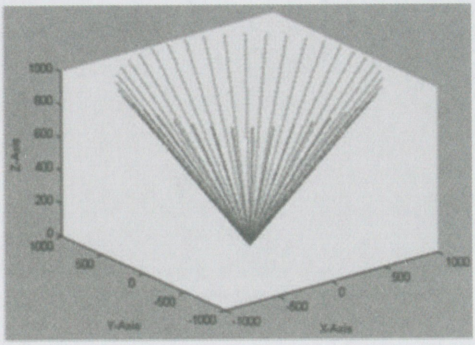
ontd.

```
 x(n)=i(1);
 y(n)=i(2);
end
penGL software
old off
lot(x,y,'.','Color','b','Markersize',1)
xis('equal');
et(gcf, 'color','yellow');
```

.11 A CONE

cone is a 3-dimensional geometric shape that tapers gradually from a flat base to a point known the apex or vertex. A cone is a pyramid with a circular cross-section. The MATLAB program plot a cone is given below.

```
or i=1:10:1000
or theta=0:0.05*pi:2*pi
   x=i*cos(theta);
   y=i*sin(theta);
   z=i;
   plot3(x,y,z)
   hold on
nd
   hold on
nd
   xlabel('X-Axis');
   ylabel('Y-Axis');
   zlabel('Z-Axis');
```



4.12 A PARABOLOID

The surface generated by the revolution of a parabola is called a paraboloid. The surface c be specified as,

$$z = c(x^2 + y^2)$$

The paraboloid with a radius of 'a' and height 'h' has the parametric equation,

$$x(u, v) = a\sqrt{\frac{u}{h}}\cos(v),\ y(u, v) = a\sqrt{\frac{u}{h}}\sin(v) \text{ and } z(u, v) = u$$

where $u \geq 0$ and $\in [0, 2\pi)$.

There can be two types of paraboloid i.e., elliptic and hyperbolic. The elliptical parabolo has the equation,

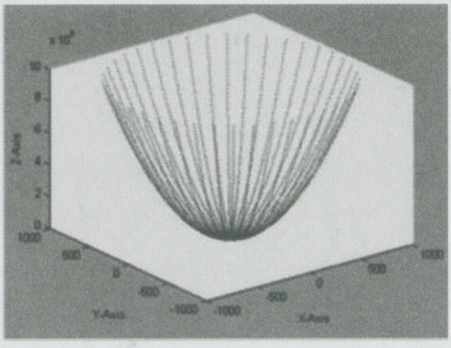
$$\frac{z}{c} = \frac{x^2}{a^2} + \frac{y^2}{b^2}$$

Here 'a' and 'b' are constants that determine the curvature of the x-z and y-z planes respectivel The equation results in an elliptic paraboloid that opens upward for c>0 and downward for c< The hyperbolic paraboloid has the equation given below.

$$\frac{z}{c} = \frac{x^2}{a^2} + \frac{x^2}{a^2}$$

When c>0 the hyperbolic paraboloid opens down along the x-axis and up along the y-axi In other words in the parabola in the x=0 plane opens upward and the parabola in the y=0 plan opens downward.

```
for i=1:10:1000
for theta=0:0.05*pi:2*pi
    x=i*cos(theta);
    y=i*sin(theta);
    z=i.*i;
    plot3(x,y,z)
    hold on
end
    hold on
end
    xlabel('X-Axis');
    ylabel('Y-Axis');
    zlabel('Z-Axis');
```



4.13 CONTOUR PLOTS

A contour plot is a graphical representation of the relationship between three numeric variables (X, Y, and Z) in a two-dimensional plot. The two variables are X and Y are the two axes and the third variable Z shows the contour levels. The contour plot displays isolines of matrix Z. The point having the same value of Z are joined together to give the contours. The contour levels are plotted as curves. The area included between curves could also be colored to indicate interpolated values. The interpolation could be linear or spline to fit the contour surface function. The contour lines in the contour plot can be labeled using the clable command.

The command,

contour(Z)

draws a contour plot of matrix Z. Z is the height with respect to the *x*-*y* plane. Z is obtained by a mathematical relation between x and y. The number of contour lines and the values of the contour lines are automatically chosen based on the minimum and maximum values of Z.

contour(Z, n) draws a contour plot of matrix Z with n contour levels where n is a scalar.

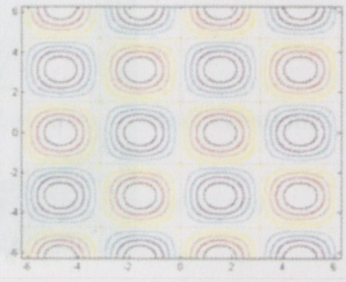
In order to draw a single contour of level i, use contour(Z, [i i]).

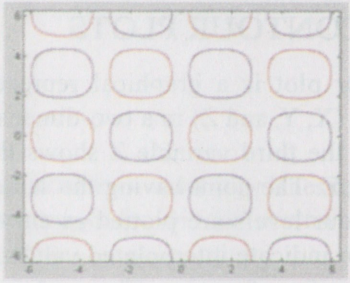
Z must be at least a 2-by-2 matrix that contains at least two different values.

The ranges of the *x*- and *y*-axis are [1:n] and [1:m], where [m, n] = size(Z).

contour(Z, v) draws a contour plot of matrix Z with contour lines at the data values specified in the monotonically increasing vector v. The number of contour levels is equal to length(v).

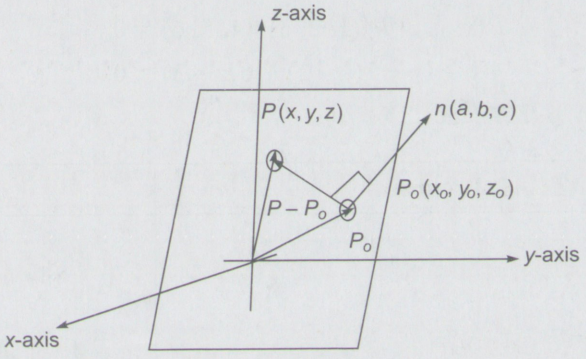
Program	Plots having different contour levels

`x = linspace(-2*pi,2*pi,1000);`
`y = linspace(-2*pi,2*pi,1000);`
`[X,Y] = meshgrid(x,y);`
`Z = sin(X).*cos(Y);`
`contour(X,Y,Z)`
	

`contour(X,Y,Z,2)`	
`contour(X,Y,Z,4)`	
`contour(X,Y,Z,6)`	

4.14 A PLANE

A plane is a two-dimensional surface. The equation of a plane in a 3-dimensional Cartesian coordinate space is defined using a normal vector which is perpendicular to the plane and a point through which the plane passes.



A Plane

Let the normal vector of the plane be n and the known point on the plane be P_o. Let any point on the plane be P. We can define the vector lying on the plane connecting P and P_o as,

$$P - P_o = ((x - x_o), (y - y_o), (z - z_o))$$

Since the vector $P - P_o$ is perpendicular to the normal vector, so the dot product of the two vectors should be 0. Therefore, the equation of the plane having normal vector n and passing through the point P_o is given as,

$$n.(P - P_o) = 0$$

This dot product of the normal to the plane and a vector on the plane becomes the equation of the plane. By evaluating the dot product we get,

$$(a, d, c), ((x - x_o), (y - y_o), (z - z_o)) = 0$$

or

$$a(x - x_o) + b(y - y_o) + c(z - z_o) = 0$$

or

$$ax + by + cz - (ax_o + by_o + cz_o) = 0$$

If we consider

$$(ax_o + by_o + cz_o) = d$$

we get,

$$ax + by + cz = d$$

as the equation of the plane.

Thus, once the normal vector and the point through which the plane passes are known it is possible to determine the equation of the plane.

The intercept of the plane of on the x axis is $-d/a$, on the y axis is $-d/b$ and on the z axis i $-d/c$. The perpendicular distance of the plane form the origin is $d\sqrt{a^2+b^2+c^2}$.

Let us consider an example of a plane that passes through the point (1, 2, 3) and is orthogona to the vector (4, 5, 6).

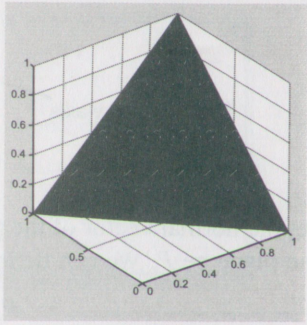
So,

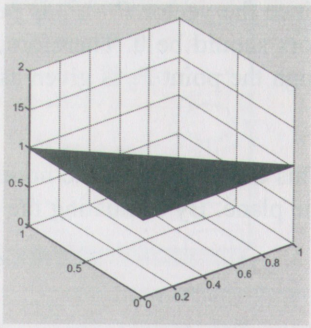
$$P(x, y, z)\ P_o(1, 2, 3).\ (4, 5, 6) = 0$$

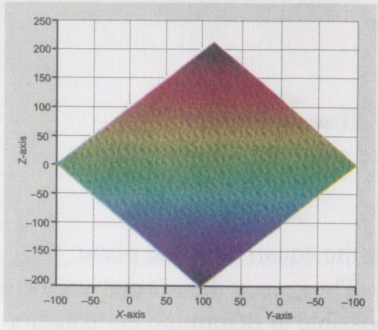
$$4(x-1) + 5(y-2) + 6(z-3) = 0$$

This is the equation to the plane.

MATLAB Program	Output

`pA = [1 1 1];`
`pB = [0 1 0];`
`pC = [1 0 0];`
`plane = cross(pA-pB, pA-pC)`
`points=[pA' pB' pC'];`
`fill3(points(1,:),points(2,:),points(3,:),'r')`
`grid on`
	

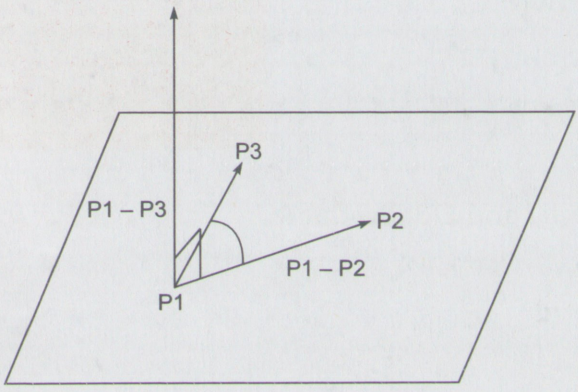
`>> pt1 = [0 1 0];`
`>> pt2 = [1 0 0];`
`>> pt3 = [1 1 1];`
`>> fill3(pt1,pt2,pt3,'b')`
`>> grid on`
	

`x=linspace(-100,100,100);`
`[X,Y] = meshgrid(x);`
`a=1;`
`b=1;`
`c=1;`
`d=1;`
`Z=(d - a * X - b * Y)/c;`
`surf(X,Y,Z)`
`xlabel('x-axis');`
`ylabel('y-axis');`
`zlabel('z-axis');`
	

.14.1 Finding the Normal to the Plane

f three points through which a plane passes are known it is possible to know the normal to a lane through these three points.

$$\text{Normal } (n) = (p1 - p2) \times (p2 - p3)$$
$$n = (p2 - P1) \times X\ (P3 - P1)$$

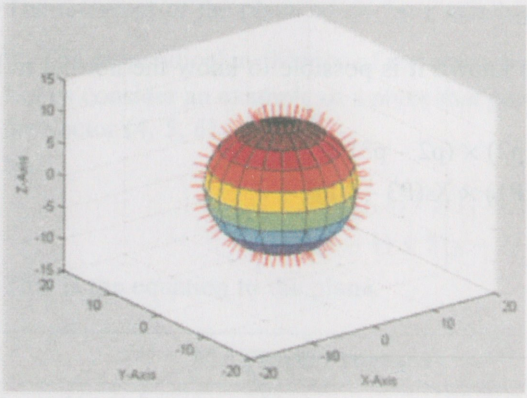


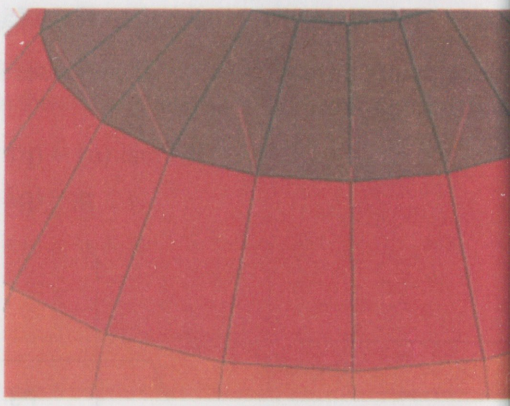
Normal to a plane

```
t1 = [0 1 0];
t2 = [1 0 0];
t3 = [1 1 1];
> normaltotheplane = cross(pt1-pt2, pt1-pt3)
ormaltotheplane =
    -1    -1     1
```

.14.2 Surface Normal (Surfnorm)

```
u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
1=10*cos(v).*sin(u);
1=10*sin(v).*sin(u);
1=10*cos(u);
urf(a1,b1,c1);
label('X-Axis')
label('Y-Axis')
label('Z-Axis')
old on
urfnorm(a1,b1,c1)
```



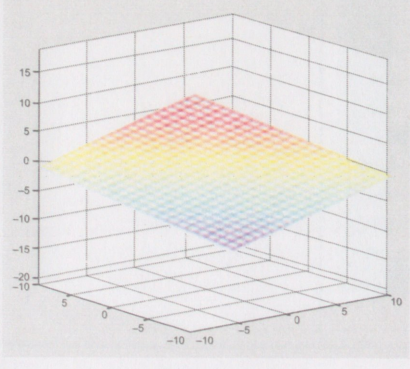


4.14.3 Cross Product and Dot Product

```
>>va = [1.2, 2.2, 3.1]; vb = [–0.7, 0.4, 0.2];
>> normal = cross (va,vb)
normal =
        –0.8000   –2.4100   2.0200
>>dotproduct = dot (va,vb)
dotproduct =
          0.6600
```

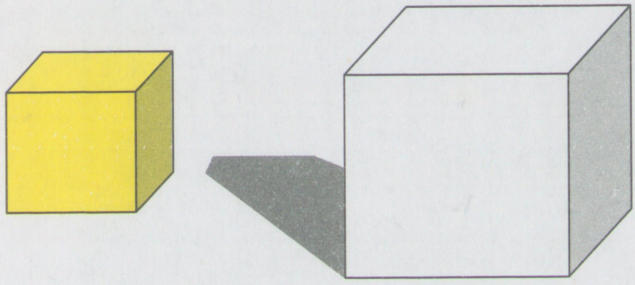
4.14.4 Plotting a Plane Using Mesh

```
>> x=–10:10; y=–10:10;
[X,Y]=meshgrid (x,y);
Z=(–1–X–Y);
mesh (X,Y,Z)
axis tight
grid on
```



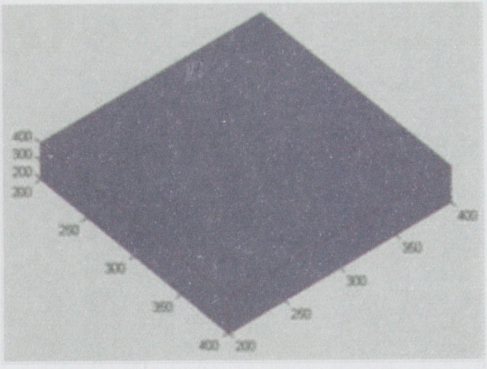
.15 CUBE

A cube is a 3-dimensional figure having all its edges of equal length and all angles equal to 90°. A cube is a 3-dimensional solid object bounded by six square faces and three faces meet at each vertex. The six square faces meet each other at right angles. It has eight vertices and twelve edges. The cube is the platonic solid and is also called the regular hexahedron. The cube is also a square parallelepiped, an equilateral cuboid and a right rhombohedron.



The MATLAB program which creates a cube is given below.

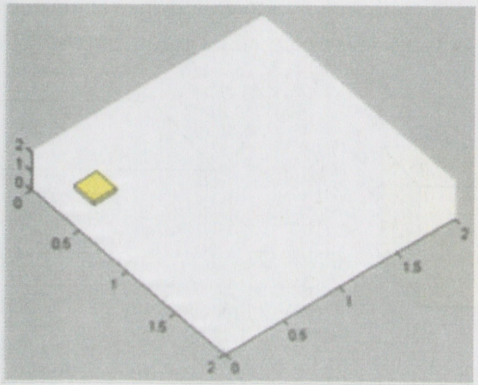
```
ertices = 200*[1 1 1; 1 2 1; 2 2 1; 2 1 1 ;1 1 2;1 2 2; 2 2 2;2 1 2];
aces = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
atch('Faces',faces,'Vertices',vertices,'FaceColor','b');
aterial dull;
iew(50,80);
```



The dimension of the cube can be changed.

```
vertices = 0.2*[1 1 1; 1 2 1; 2 2 1; 2 1 1 ;1 1 2;1 2 2; 2 2 2;2 1 2];
faces = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
```

```
patch('Faces',faces,'Vertices',vertices,'FaceColor','y');
material dull;
view(50,80);
axis([0 2 0 2 0 2]);
```



5
CHAPTER

Image Processing Using MATLAB

*"**Charles Babbage** (26th December 1791-18th October 1871) was a mathematician, philosopher and also a mechanical engineer. He is the creator of programmable computer. He is considered the father of computing. He invented the first mechanical computer. The original machine created by Babbage is now in the London Science Museum. Babbage on July 14th 1822 presented in a paper entitled "Note on the application of machinery to the computation of astronomical and mathematical tables" to the Royal Astronomical Society. He called this the "difference machine." The machine was able to calculate polynomials by using a numerical method known as the differences method. In 1823 the Society approved the idea presented by Charles Babbage and the government granted him £1500 to build the machine he designed."*

5.1 DIGITAL IMAGE

According to the English dictionary an image is a representation of the external form of a person or thing in art. An image is a visual representation of something. The word image in the area of computer science or information technology is a picture that has been created or copied and stored in electronic form and is also called a digital image.



A digital image

5.1.1 Types of Images

There are essentially 5 types of images in MATLAB.

(i) Binary Image

The image format stores an image in the form of a matrix. In this type of format a pixel can have colour black and white only. 0 is assigned for black and 1 for white. A binary image is an $M \times N$ logical matrix where the pixel values are 1 (true) or 0 (false).

(ii) RGB Image

A truecolor RGB or truecolor red-blue-green (RGB) image is represented by a 3-D m × n matrix. Each pixel has red, green and blue components along the z-axis with values in the range [0, 1]. The R,G B components of pixel (m, n) are i (m, n, 1), i (m, n, 2) and i (m, n, 3) respectively

(iii) Grayscale Image

In digital images a grayscale is an image in which the value of each pixel in the image is a single sample, that is, it carries the information about intensity. These images are also called black and white images and are composed of shades of gray, varying from completely black (0) to completely white (1). A grayscale image of size $M \times N$ wide is represented as a matrix of double data type which represents the pixel grayscale intensity in the scale of 0 to 1 where 0 is black and 1 is white.

(iv) Indexed

Indexed or paletted images are represented with an indexed matrix of $M \times N$. The colormap holds all the colors that have been used in the image and the index matrix shows the pixels referring to the colors in the colormap. If for example the 30th color is red cmap (30, :) =[1,0, 0] then the image=(m,n)=30 is a red colored pixel.

(v) Unit8

This type of image uses less memory. In case of some operations computation is faster than the double type of data.

5.1.2 Contrast

It could be considered the relative difference between lighter regions and dark regions of an image. Contrast is the difference in luminance and color which makes the object distinguishable. Contrast is determined by the difference in colour or brightness of the object compared to other objects within the same field of view. The variation in the image with increasing contrast is shown in the figure below.



Original image +20 % Contrast +40 % Contrast

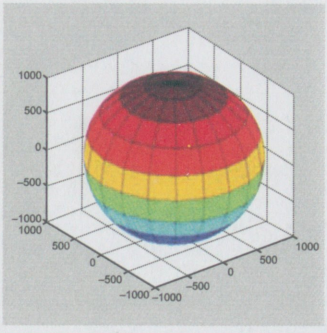
Contrast is the difference between the color or shading of the printed region and the background on which it has been printed.

5.1.3 Brightness

When an object radiates or reflects light it is said to be bright. Brightness is the luminance of an object and the luminance of an object decides if it is bright or not. Brightness can be considered the intensity of the energy output of a light source. It is often expressed as the total energy or the value or the amplitude at the wavelength where the intensity is maximum. Pure white has the maximum brightness and black which absorbs all the light has the minimum brightness.

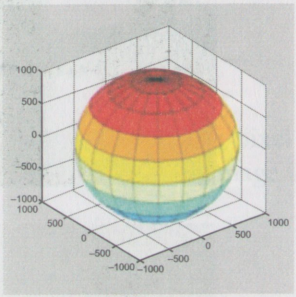
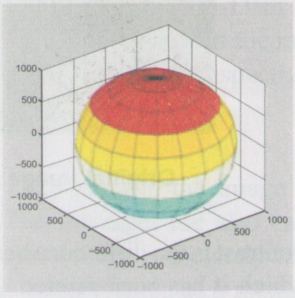
One method by which we can brighten an image in MATLAB is,

Program	Output

for i=1:100
[u,v] =
meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
a1=10*i*cos(v).*sin(u);
b1=10*i*sin(v).*sin(u);
c1=10*i*cos(u);
surf(a1,b1,c1);
end
	

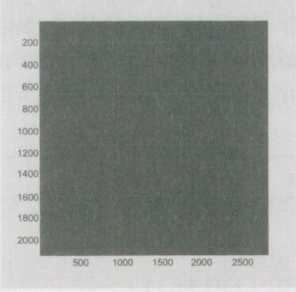
Contd.

Contd.

`>> a=0.7;` `>> brighten(a)`	
`>> a=0.9;` `>> brighten(a)`	

One can observe how the colors are changing from darker colors like brown and red to lighte and brighter colors yellow and white.

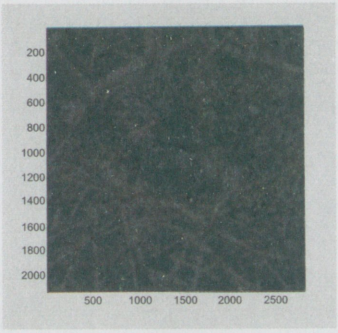
Program	Output	Comments

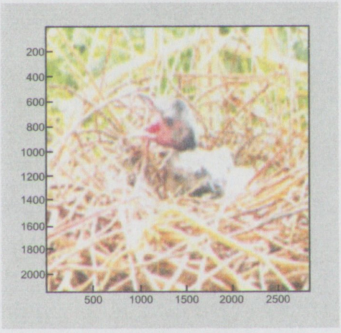
`cd d:\pictures`
`r=imread('bird.jpg');`
`image(r*-0.2);`
		Image is completely dark

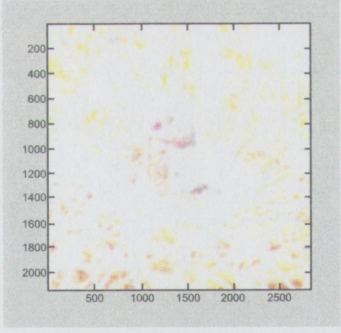
`cd d:\pictures`
`r=imread('bird.jpg');`
`image(r);`
		Actual image

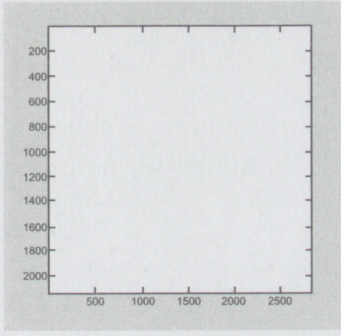
Contd

Contd.

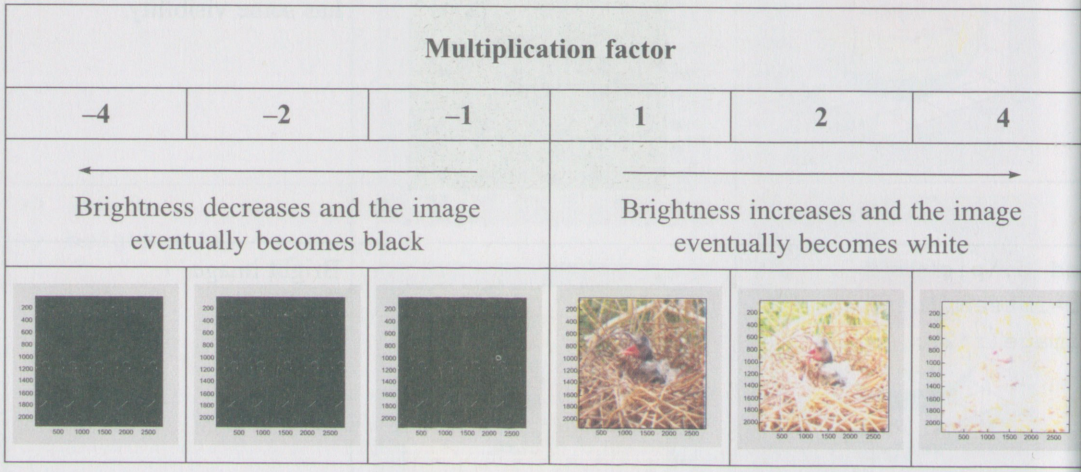
`cd d:\pictures`
`r=imread('bird.jpg');`
`image(r*0.2);`
		Image is dark but as compared to image (r*–0.2) the command image (r*0.2) has some visibility.

`cd d:\pictures`
`r=imread('bird.jpg');`
`image(r*2);`
		Bright image

`cd d:\pictures`
`r=imread('bird.jpg');`
`image(r*5);`
		Very bright image

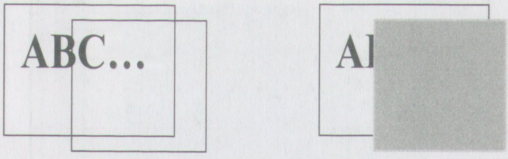
`cd d:\pictures`
`r=imread('bird.jpg');`
`image(r*10);`
		Completely bright image. Nothing is visible.

Multiplication of the image array by a positive constant having a large magnitude results in brighter image and a very large positive multiplication factor leads to a completely white image On the other hand multiplying the image array by a constant that is small or negative leads t a darker image.



5.1.4 Transparency

Transparency is the physical property of a body to allow light to pass through it without being scattered. Transparent materials appear clear. A transparent medium not only allows the transpor of light but also leads to image formation.

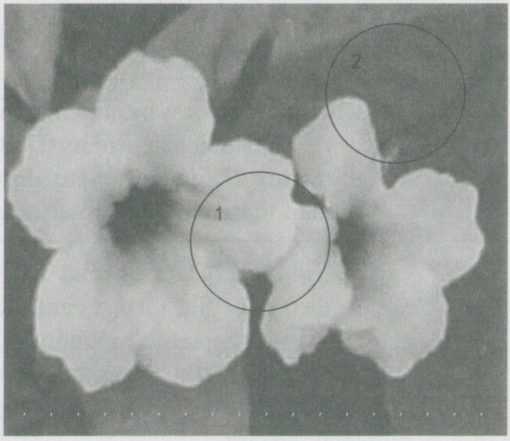


(a) (b)

Here in the above figure (a) the layer above the sheet with alphabets ABC... written in it is 100 % transparent. On the other hand in figure (b) the grey layer is completely opaque or is 0 % transparent.

5.1.5 Depth of Field

In a photograph depth of field is the length between the nearest and the farthest objects that have acceptable sharp focus.



The flowers in circle 1 are in focus whereas the leaves in the circle 2 are out of focus.

Depth of field or DOF is the distance between the nearest and the farthest objects in a scene hat are in focus and gives a sharp image. The sharpness gradually decreases outside this length. Outside the DOF the images is not sharp.

.1.6 Color Specification

There are 3 ways by which one can specify color in a MATLAB graphics. They are:

(i) RGB Triple
(ii) Short name
(iii) Long name

There are eight predefined colors in MATLAB. The RGB (red-green-blue) triple is a three-lement row vector whose elements specify the intensities of the red, green and blue components f the color. The intensities lie in the range of 0 to 1.

RGB Value	Short Name	Long Name
[1 1 0]	y	yellow
[1 0 1]	m	magenta
[0 1 1]	c	cyan
[1 0 0]	r	red
[0 1 0]	g	green
[0 0 1]	b	blue
[1 1 1]	w	white
[0 0 0]	k	black

The details of few image formats supported by MATLAB are:

BMP	BMP stands for a bitmap image file. It is also known as device independent bitmap (DIB) or bitmap. It is capable of storing 2D images of any dimension and resolution. A bitmap image file can color or monochromatic
HDF	Hierarchical Data Format or HDF is the name given to the set of file formats and libraries designed to store and organize large amounts of data. It was originally developed at the National Center for Supercomputing Applications and is presently supported by the non-profit HDF Group.
JPEG	JPEG stands for the Joint Photographic Expert Group. JPEG can give upto 10:1 compression with any perceptible loss in the image quality. The extension for this format of digital images is .jpg
PCX	PCX stands for Personal Computer Exchange. It is an image file format. It was initially the native file format for PC Paintbrush and had become one of the first widely accepted imaging standards in the DOS operating system. Files in the PCX format are commonly stored palette-indexed images which range from 2 or 4 colors to 16 and 256 colors. This format is also extended for true color or 24-bit image.
TIFF	TIFF or the tagged image file format is a format for storing raster graphics images. This is one of the most popular formats among artists and publishers. Armature as well as professional photographers use this format very commonly. The format was developed by a company called Aldus for use in desktop publishing. Since 2009 this format is under the control of Adobe Systems.

A number of other sophisticated image formats have been developed such as GIF, JPEG and PNG.

MATLAB is capable of reading and writing images using the imread and imwrite commands. It can display the images using the image and imshow commands. MATLAB supports a large number of file formats. imformats command can be used to see the list of image formats that are supported by the version of MATLAB installed in the computer.

>> imformats

EXT	ISA	INFO	READ	WRITE	ALPHA	DESCRIPTION
bmp	isbmp	imbmpinfo	readbmp	writebmp	0	Windows Bitmap (BMP)
cur	iscur	imcurinfo	readcur		1	Windows Cursor resources (CUR)
fts	fits	isfits	imfitsinfo	readfits	0	Flexible Image Transport System (FITS)
gif	isgif	imgifinfo	readgif	writegif	0	Graphics Interchange Format (GIF)
hdf	ishdf	imhdfinfo	readhdf	writehdf	0	Hierarchical Data Format (HDF)
ico	isico	imicoinfo	readico		1	Windows Icon resources (ICO)
j2c j2k	isjp2	imjp2info	readjp2	writej2c	0	JPEG 2000 (raw codestream)
jp2	isjp2	imjp2info	readjp2	writejp2	0	JPEG 2000 (Part 1)
jpf	jpx	isjp2	imjp2info	readjp2	0	JPEG 2000 (Part 2)
jpg jpeg	isjpg	imjpginfo	readjpg	writejpg	0	Joint Photographic Experts Group (JPEG)
pbm	ispbm	impnminfo	readpnm	writepnm	0	Portable Bitmap (PBM)
pcx	ispcx	impcxinfo	readpcx	writepcx	0	Windows Paintbrush (PCX)
pgm	ispgm	impnminfo	readpnm	writepnm	0	Portable Graymap (PGM)
png	ispng	impnginfo	readpng	writepng	1	Portable Network Graphics (PNG)
pnm	ispnm	impnminfo	readpnm	writepnm	0	Portable Any Map (PNM)
ppm	isppm	impnminfo	readpnm	writepnm	0	Portable Pixmap (PPM)
ras	isras	imrasinfo	readras	writeras	1	Sun Raster (RAS)
tif tiff	istif	imtifinfo	readtif	writetif	0	Tagged Image File Format (TIFF)
xwd	isxwd	imxwdinfo	readxwd	writexwd	0	X Window Dump (XWD)

5.1.7 Pixel

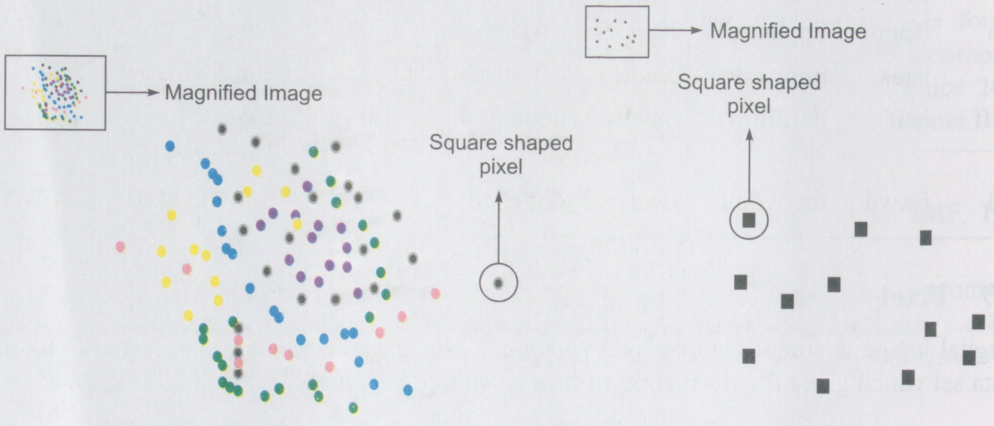
A digital image is composed of pixels which can be considered small dots. A digital image is a data set which gives the instruction of how to colour each pixel or dot. A digital image could

have a size of 512 by 512 pixels. Usually, the number pixels is in the form 2n. An image o size m × n pixel is composed of m pixels in the vertical direction and n pixels in the horizonta direction. A 512 × 512 image has a total number of 262144 pixels. This is a very large numbe and in order to store this information a very large memory size is required. As a result we nee to compress this image and reduce the size of the data. This can be done using techniques lik Fourier analysis and Wavelet Analysis.

The word pixel was first used in 1965 by Frederic C Billingsley of JPL, Jet Propulsio Laboratory. He published two papers in 1965 where he used the term pixel. He used the term pixe to describe elements of video images from space probes to moon and mars. Frederic C Billingsle spent most of his career working on digital image processing. Pixel is the short form of pictur element. In a digital image a pixel or pel as it is sometimes known is the smallest element in digital image display device. A display monitor displays a digital image by dividing the scree into thousands and millions of pixels. The pixels are arranged in rows and columns. Each pixe has its unique address which is determined by its physical coordinates. It is hard to distinguis individual pixels in the monitor as they are very close to each other. The number of bits used t represent each pixel determines how many colours or shades of gray can be displayed. A 8 bit color mode in a color monitor uses 8 bits for each pixel so it is possible to display 28 = 25 different colors or shades of gray. On color monitors each pixel is composed of a red, blue an green dots, all converging at a point. The quality of a display depends on the number of pixels i can display and the bits used to represent each pixel. A true color system uses 24 bits per pixe and this makes it possible to display more than 16 million different colours.

A digital image has all its information in the form of digits stored in an array. Everything i this digital image is discrete. A digital image is a large array of dots. Each dot has a numbe associated with it. The numbers, usually integers, have the information relating to the brightnes of the dots. All the dots make up the entire image. Each dot is called a pixel. A pixel neigbourhood is determined by the array size we select around a pixel.

In paint when the pencil is used to color single pixels and then the pixels are zoomed to several hundred time we can notice that the pixels are square in dimension and it is these tiny square that make a complete image.



The actual pixels in paint. The paint images have been resized.



If we observe carefully we can easily see the square shaped pixels in the image when magnified 800 times.

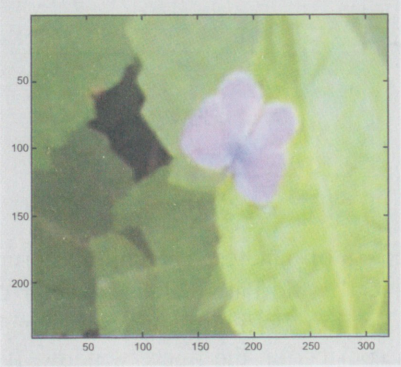
5.1.8 Megapixel

The term megapixel is used to denote the size of a digital image. Megapixel stands for one million pixels (10^6 pixels). As pixels are usually square and form a grid, a 1-megapixel camera will produce an image about 1200 pixels wide by 900 pixels long. Digital images with more megapixels are larger in size and so can a lot of space for their storage. The resolution of digital cameras is often measured in terms of megapixels.

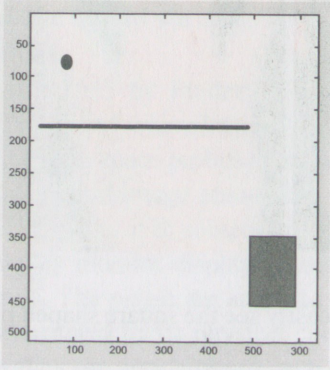
5.2 image (c)

image(C) displays matrix C as an image. Each element of the matrix C specifies the color of a pixel in the image. The different ways to read an image and display it in a figure window is shown below.

```
>> a=imread('flower.jpg','jpg');
>> image(a)
```



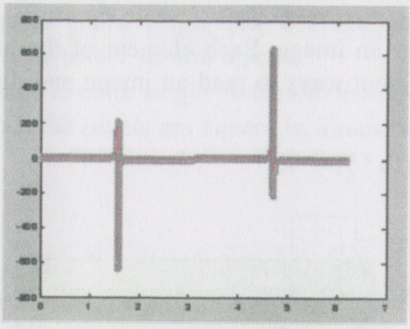
```
>> a=imread('01.bmp','bmp');
>> image(a)
>> axis square
```



5.3 CREATE A SHADOW TO A PLOT

First a line is plotted. A handle is set to the line.

```
>> theta=linspace(0,2*pi,1000);
>> h11=plot(theta,tan(theta),'r');
>> h21=line(theta,tan(theta),'LineWidth',8,'Color',[0.6 0.6 0.6]);
>> set(gca,'Children',[h11 h21]);
```



A shadow plot

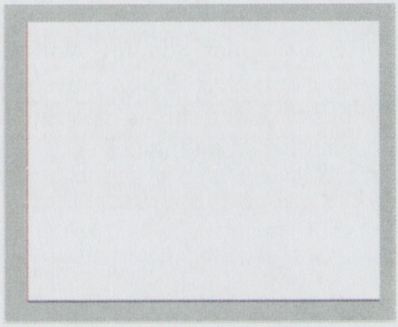
5.4 CHILDREN

Children of a figure are the vectors of handles. The children property of a given object are the handles of the objects. The object is the Parent. It is a vector containing of all the axes and user-interface objects displayed within the figure. One can change the order of the handles and thus change the stacking of the objects on the figure window. It should be noted here that the handles are not hidden.

gca gives the current axis handle.

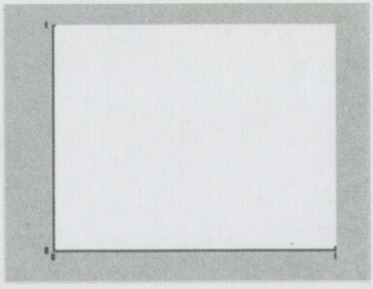
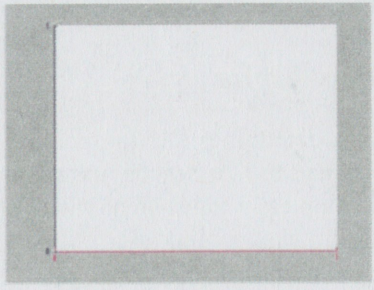
hl=gca returns the handle to the current axis for the current figure. In case multiple axes are reated hl is the handle to the last axes created or clicked with the mouse. In case an axis is not xisting then gca creates an axis and returns it handle.

```
-> figure
-> set(gca,'xcolor','b','ycolor','r','xtick',[],'ytick',[])
```



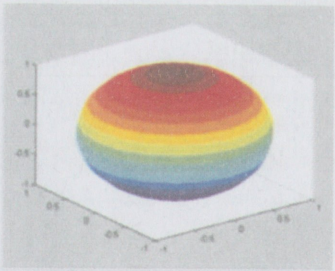
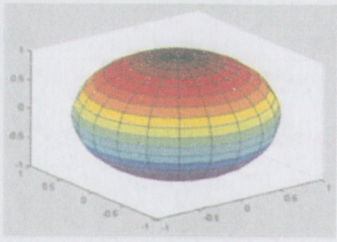
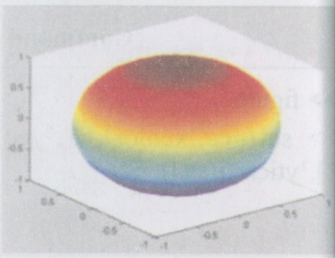
Axis created using gca

Few more examples are illustrated below.

Command	Output
>> figure >> set(gca,'xcolor','k','ycolor','k','xtick',[0 1],'ytick',[0 1])	
>> figure >> set(gca,'xcolor','r','ycolor','b','xtick',[0 1],'ytick',[0 1])	

5.5 SHADING

The function shading is used to set color shading properties. The various shading options availab in MATLAB are:

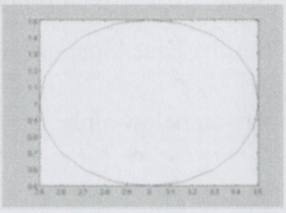
Flat	Faceted	Interp
In the flat shading option each mesh line segment and face has a constant colour determined by the color value at the endpoint of the segment or the corner of the face that has the smallest index or indices.	Here the flat shading with superimposed black mesh lines is done. This is also the default shading mode.	As the name suggest th interp shading option varie the color in each line segmen and face by interpolating th colormap index or true colo value across the line or face.
>> sphere(20) >> shading flat	>> shading faceted	>> shading interp
		

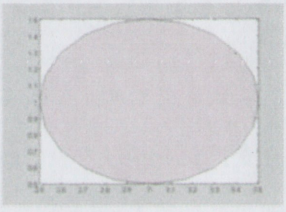
Beside this shading(axes_handle,...) applies the shading type to the objects in the axe specified by axes_handle, instead of the current axes.

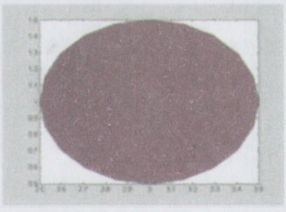
5.6 FaceAlpha

FaceAlpha determines the transparency of the face a patch in the current axis. It takes a valu between 1 and 0. This sets the transparency of the faces of the object. 1 by default is opaque an 0 is for completely transparent which also means invisible.

Program	Output	Comments

`>> x = 3.0;`
`y = 1.0;`
`r = 0.5;`
`x1 = r*sin(-pi:0.1*pi:pi) + x;`
`y1 = r*cos(-pi:0.1*pi:pi) + y;`
`cl=[0.5 0 0.5];`
`ill(x1, y1, cl, 'FaceAlpha', 0);`
		Here although the color is set as cl=[0.5 0 0.5] Since faceAlpha value is 0 it is completely transparent.

`x = 3.0;`
`y = 1.0;`
`r = 0.5;`
`x1 = r*sin(-pi:0.1*pi:pi) + x;`
`y1 = r*cos(-pi:0.1*pi:pi) + y;`
`cl=[0.5 0 0.5];`
`ill(x1, y1, cl, 'FaceAlpha',`
`).2);`
		Here although the color is set as cl=[0.5 0 0.5] and faceAlpha value is 0.2

`>> x = 3.0;`
`y = 1.0;`
`r = 0.5;`
`x1 = r*sin(-pi:0.1*pi:pi) + x;`
`y1 = r*cos(-pi:0.1*pi:pi) + y;`
`cl=[0.5 0 0.5];`
`ill(x1, y1, cl, 'FaceAlpha', 1);`
		Here the faceAlpha value is set at 1 making the object completely opaque

alpha sets the transparency properties for objects in current axes.

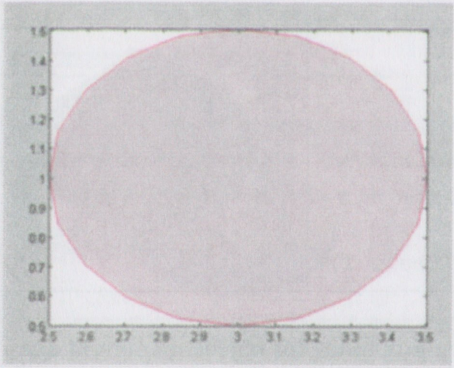
alpha(FaceAlpha) sets the FaceAlpha property of all image, patch and surface objects aceAlpha can be set to,

Option	Property	Output
scalar	Set the FaceAlpha property to the specified value	
'flat'	Set the FaceAlpha property to flat	
'interp'	Set the FaceAlpha property to interp.	

'texture'	Set the FaceAlpha property to texture.	
'opaque'	Set the FaceAlpha property to 1.	Opaque
'clear'	Set the FaceAlpha property to 0.	Transparent

For example in the program given below alpha has been set at 0.2 which means that the obje should be transparent.

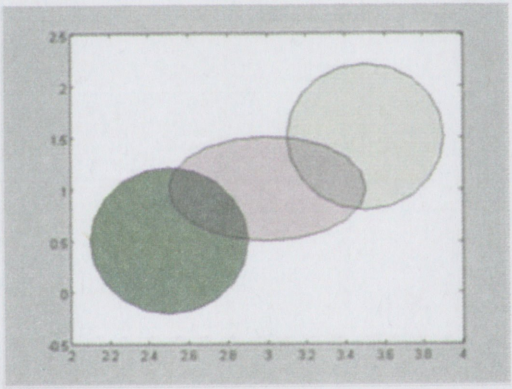
```
x = 3.0;
y = 1.0;
r = 0.5;
x1 = r*sin(-pi:0.1*pi:pi) + x;
y1 = r*cos(-pi:0.1*pi:pi) + y;
cl=[0.5 0 0.5];
fill(x1, y1, cl,'edgecolor','red');
alpha(0.2)
```



One can observe the change in transparency when FaceAlpha is varied.

```
x = 3.0;
y = 1.0;
r = 0.5;
x1 = r*sin(-pi:0.1*pi:pi) + x;
y1 = r*cos(-pi:0.1*pi:pi) + y;
cl=[0.5 0 0.5];
fill(x1, y1, cl, 'FaceAlpha', 0.2);
hold on
xa = 3.5;
ya = 1.5;
```

```
b = 0.4*sin(-pi:0.1*pi:pi) + xa;
b = 0.7*cos(-pi:0.1*pi:pi) + ya;
la = [0.2 0.5 0.2];
ll(xb, yb, cla, 'FaceAlpha', 0.2);
old on
a = 2.5;
a = 0.5;
b = 0.4*sin(-pi:0.1*pi:pi) + xa;
b = 0.7*cos(-pi:0.1*pi:pi) + ya;
la = [0.2 0.5 0.2];
ll(xb, yb, cla, 'FaceAlpha', 0.7);
```



5.7 GIVING COLOR AND TRANSPARENCY TO EDGES OF A FIGURE

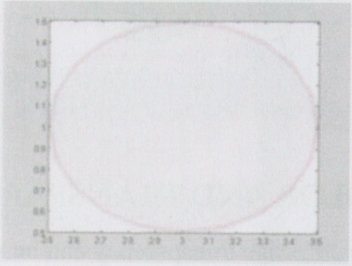
The edge color, line width, edge transparency can be set using the functions edgecolor, linewidth and edgeAlpha.

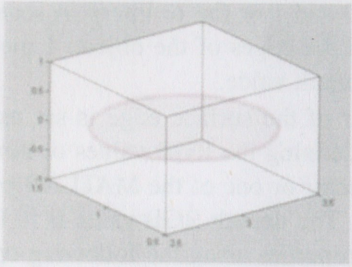
Transparency of the surface edges is set using the edgeAlpha function. EdgeAlpha takes a scalar value between 0 and 1 or flat or interp. A scalar value between 0 and 1 could be set for the transparency of all the edges of the object. 1 means fully opaque and 0 means completely transparent. 1 is the default value.

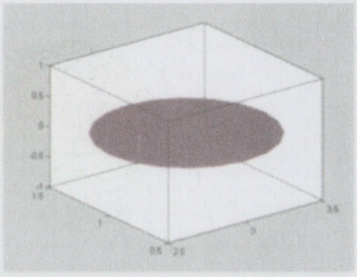
Edge color or color of the surface edge is set using the edgecolor function. Color for the edge could be set by defining the RGB values or none, flat and interp could also be chosen. A three-element RGB vector or one of the MATLAB predefined names, specifying a single color for edges can be used. The default RGB value is [0 0 0] which is black color. Besides this one can also set the line properties using the following options:

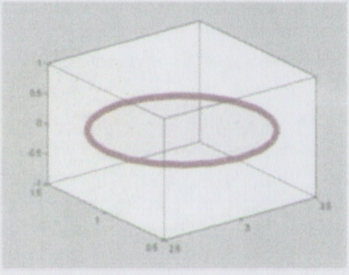
Option	Explanation	Output
'Color'	Sets the line color	[0 0 1] (blue) is the default, three element RGB vector or string
'LineStyle'	Sets the line style	'-' (default) or '--' or ':' or '-.' or 'none
'LineWidth'	Sets the line width	0.5 (default) or any other scalar value
'Marker'	Sets the marker	'none' (default) or string
'MarkerEdgeColor'	Sets the marker edge color	'auto' (default) or 'none' or three element RGB vector or string
'MarkerFaceColor'	Sets the marker face color	'none' (default) or 'auto'
'MarkerSize'	Sets the marker size.	6 (default) or any other scalar value

Few other examples are given below.

x = 3.0;
y = 1.0;
r = 0.5;
x1 = r*sin(–pi:0.1*pi:pi) + x;
y1 = r*cos(–pi:0.1*pi:pi) + y;
cl=[0.5 0 0.5];
cl2=[0.8 0 0.8];
fill(x1, y1, cl,'edgecolor',cl2,'linewidth', 10,'edgealpha',0.1,'facealpha',0.1);
		Highly transparent

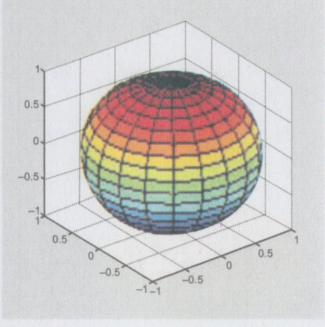
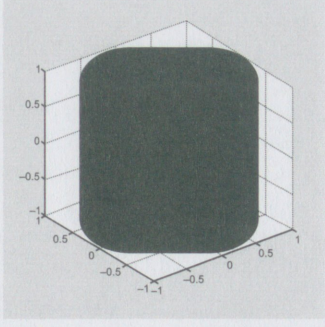
>> view(3)		Shows clearly that both the edge and the patch are transparent

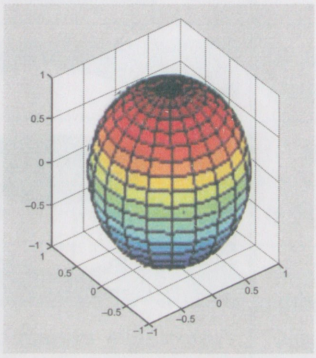
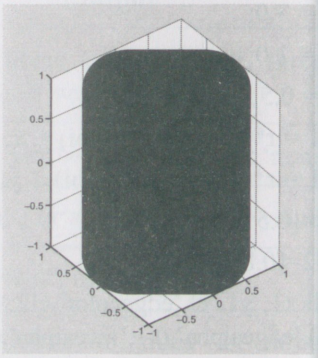
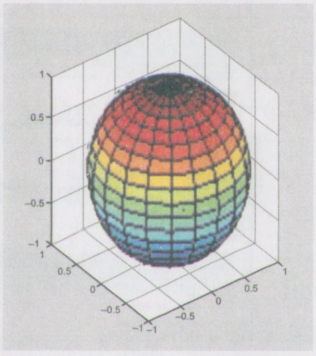
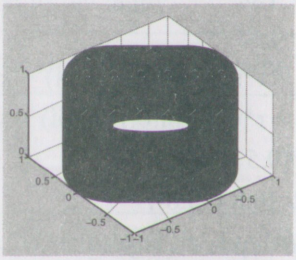
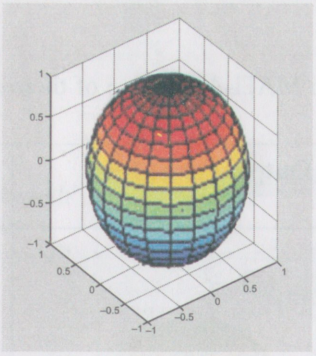
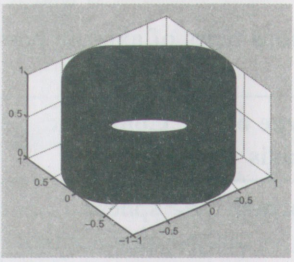
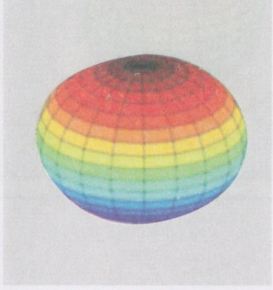
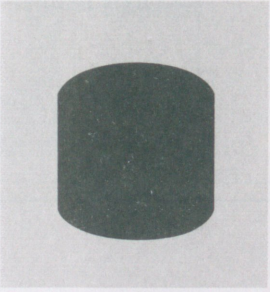
x = 3.0;
y = 1.0;
r = 0.5;
x1 = r*sin(-pi:0.1*pi:pi) + x;
y1 = r*cos(-pi:0.1*pi:pi) + y;
cl=[0.5 0 0.5];
cl2=[0.8 0 0.8];
fill(x1, y1, cl,'edgecolor',cl2,'linewidth',
10,'edgealpha',0.1,'facealpha',0.9);
		Here alpha=0.9 and the object is opaque.

x = 3.0;
y = 1.0;
r = 0.5;
x1 = r*sin(–pi:0.1*pi:pi) + x;
y1 = r*cos(–pi:0.1*pi:pi) + y;
cl=[0.5 0 0.5];
cl2=[0.8 0 0.8];
fill(x1, y1, cl,'edgecolor',cl2,'linewidth',
10,'edgealpha',0.9,'facealpha',0.1);
		The object is very transparent as alpha=0.1

.8 TYPES OF AXIS

arious types of axis are possible in MATLAB. Some of them are listed below.

Type of axis	Command	Output for Sphere	Output for Cylinder
Normal	>> sphere >> axis normal >> cylinder >> axis normal		

Square	>> sphere >> axis square >> cylinder >> axis square		
Equal	>> sphere >> axis equal >> cylinder >> axis equal		
Image	>> sphere >> axis image >> cylinder >> axis image		
off	>> sphere >> axis off >> >> cylinder >> axis off		

Axis can also be deleted in images. For example,

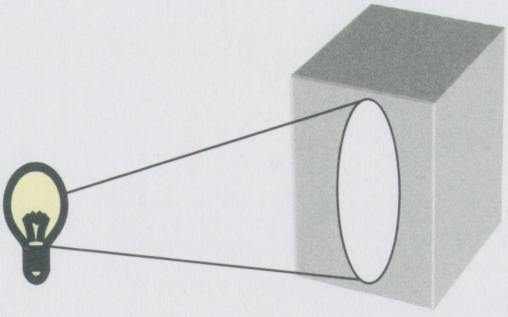
```
d D:\pictures
=imread('bird.jpg','jpg');
mage(a)
xis off
```



5.9 LIGHTING AND MATERIAL

5.9.1 Light

Light can be used to create a light source. Light creates a light object in the current axes. This light created affects only patch and surface objects.



Lighting source is a bulb and the material is a metallic piece. The lighting depends both on the source and the material.

The syntax for light is,

light('Property1',value1, 'Property2',value2...)

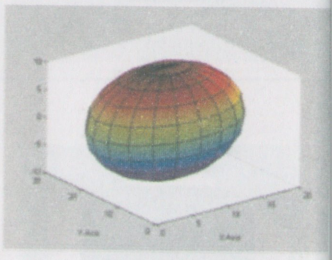
handle = light(...)

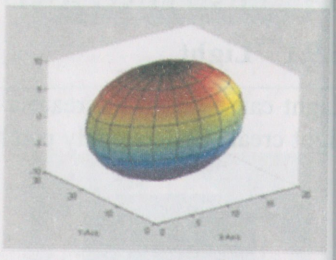
It is possible to create a light object having specified values for the properties mentioned. A handle to the light object can also be created using the command,

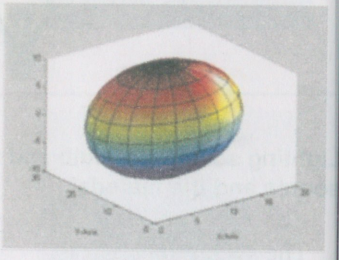
handle = light(...)

For example,

MATLAB Program	Output

`[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a=10+10*cos(v).*sin(u);`
`b=10*sqrt(3)+10*sin(v).*sin(u);`
`c=0+10*cos(u);`
`h= surf(a,b,c);`
`xlabel('X-Axis')`
`ylabel('Y-Axis')`
`set(h,'FaceLighting','flat','FaceColor','interp','AmbientStrength',0.7)`
`light('Position',[1 1 1],'Style','infinite');`
	

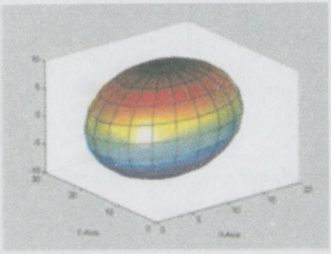
`[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a=10+10*cos(v).*sin(u);`
`b=10*sqrt(3)+10*sin(v).*sin(u);`
`c=0+10*cos(u);`
`h= surf(a,b,c);`
`xlabel('X-Axis')`
`ylabel('Y-Axis')`
`set(h,'FaceLighting','gouraud','FaceColor','interp','AmbientStrength',0.7)`
`light('Position',[1 0 0],'Style','infinite');`
	

`[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a=10+10*cos(v).*sin(u);`
`b=10*sqrt(3)+10*sin(v).*sin(u);`
`c=0+10*cos(u);`
`h= surf(a,b,c);`
`xlabel('X-Axis')`
`ylabel('Y-Axis')`
`set(h,'FaceLighting','phong','FaceColor','interp','AmbientStrength',0.7)`
`light('Position',[1 0 0],'Style','infinite');`
	

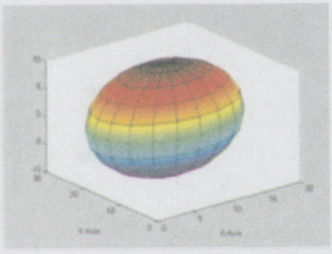
Contd.

Contd.

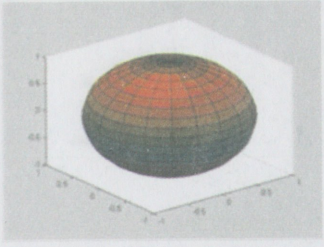
```
[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
a=10+10*cos(v).*sin(u);
b=10*sqrt(3)+10*sin(v).*sin(u);
c=0+10*cos(u);
h= surf(a,b,c);
xlabel('X-Axis')
ylabel('Y-Axis')
set(h,'FaceLighting','phong','FaceColor',
'interp','AmbientStrength',0.7)
light('Position',[1  0  0],'Style','local',
'visible','on');
```



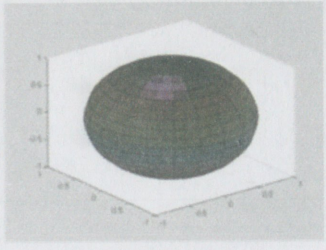
```
[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
a=10+10*cos(v).*sin(u);
b=10*sqrt(3)+10*sin(v).*sin(u);
c=0+10*cos(u);
h= surf(a,b,c);
xlabel('X-Axis')
ylabel('Y-Axis')
set(h,'FaceLighting','phong','FaceColor',
'interp','AmbientStrength',0.7)
light('Position',[1  0  0],'Style','local',
'visible','off');
```



```
sphere
light('color','r','style','local','position',
[0 0 10])
```

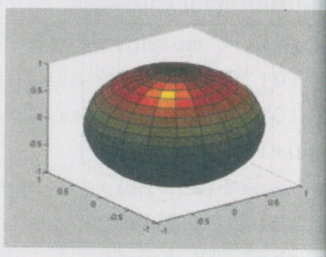


```
sphere
light('color','b','style','local','position',
[0 0 10])
```

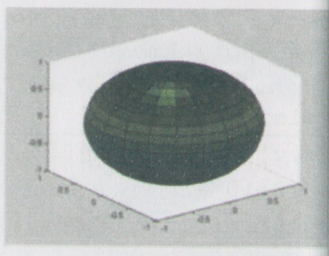


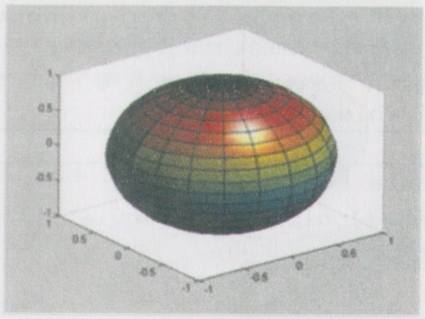
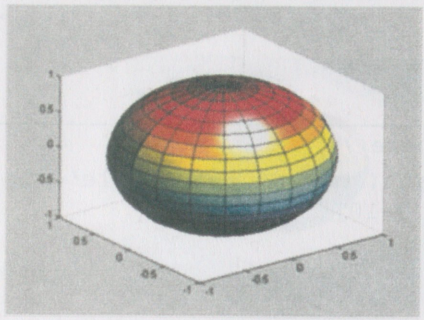
Contd.

```
sphere
light('color','y','style','local','position',
[0 0 10])
```



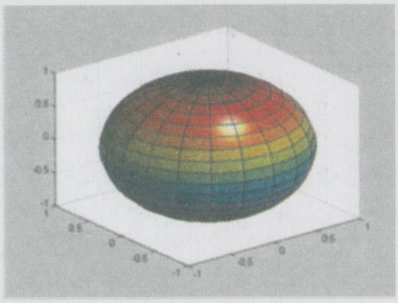
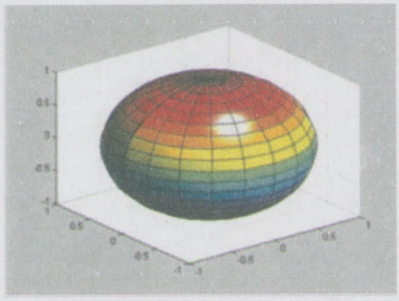
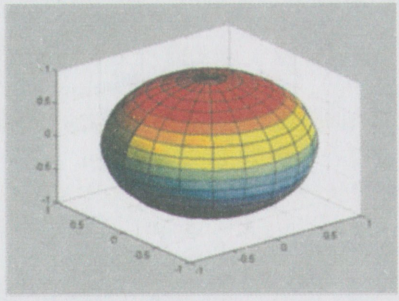
```
sphere
light('color','g','style','local','position',
[0 0 10])
```



The variation in light angle and material	Output
>> sphere >> camlight >> lighting phong	
>> sphere >> camlight >> lighting phong >> lightangle (0, 50)	

Contd

Contd.

sphere camlight lighting phong lightangle(0, 50) material metal	
sphere camlight lighting phong lightangle(0, 50) material shiny	
sphere camlight lighting phong lightangle(0, 50) material dull	

.9.2 Material

he choice of the material in MATLAB decides the reflectance properties of the surfaces nd patches. The material command sets the lighting characteristics of surfaces and patch. bjects. The material command sets the AmbientStrength, DiffuseStrength, SpecularStrength, pecularExponent, and Specular Color Reflectance properties of all the surface and patch objects. here must be a visible light object in the axes for lighting to be enabled. The syntax for using ıe material function is,

1. material shiny
2. material dull
3. material metal

Besides the choice of shiny dull and metal we can also choose a material specifically designed y us. The syntax for this is,

1. material([ka kd ks])
2. material([ka kd ks n])
3. material([ka kd ks n sc])

Here ka stands for ambient, diffuse and specular strength of the object. The choice of defau gives us default material. n stands for the specular exponent and sc stands for the specular col reflectance of the object. The material default option gives us the default material.

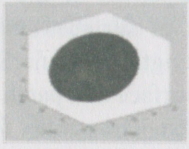
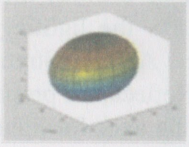
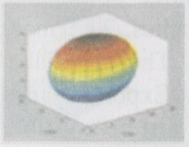
In MATLAB material is used for determining the reflectance properties of surfaces a patches.

For example, here a sphere has been created along with a light source using the MATLA program given below. The material of the sphere is now changed to find the effect of the lig on the object.

```
[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
a=10+10*cos(v).*sin(u);
b=10*sqrt(3)+10*sin(v).*sin(u);
c=0+10*cos(u);
h= surf(a,b,c);
xlabel('X-Axis')
ylabel('Y-Axis')
set(h,'FaceLighting','phong','FaceColor','interp','AmbientStren
th',0.7)
light('Position',[1 0 0],'Style','local','visible','on');
```

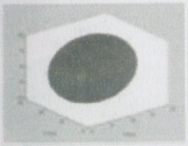
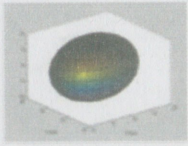
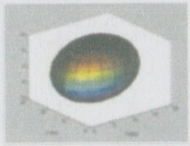
In MATLAB material sets the lighting characteristics of a material. The material of the sphe is changed using the various options available for material. Before we know the various optio available for material in MATLAB we must know the following terms:

Specular reflection—Specular reflection is mirror-like reflection of light from a surface object. In specular reflection light from a single incoming direction is reflected back in a sing outgoing direction. The angle of incidence is equal to the angle of reflection. The incider reflected and normal directions are coplanar. Ks is the coefficient of specular reflection. Th specular color reflectance property of the patch or the surface controls the color.

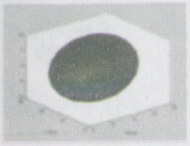
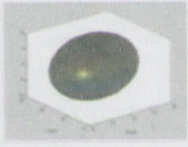
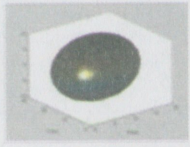
material([0 0.2 0.2])	material([0.5 0.2 0.2])	material([1 0.2 0.2])
		

As Ks is increased from 0 to 1 the sphere looks different.

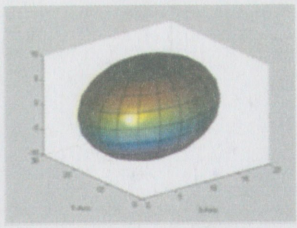
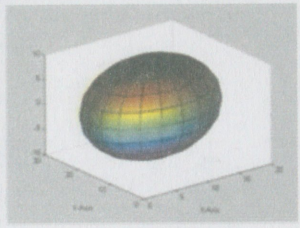
Ambient reflection—The intensity of ambient light is uniform at every point. The ambie reflection coefficient Ka, $0 \leq Ka \leq 1$, determines the reflected fraction of ambient light. La is th ambient component of light source and the ambient intensity is $Ia = Ka \times La$. In the examp given below Ka is increased gradually for 0 to 1.

material([0.2 0 0.2])	material([0.2 0.5 0.2])	material([0.2 1 0.2])
		

Diffuse reflection—Diffuse reflection is the reflection of light from a surface such that an incident ray is reflected at many angles instead of a single angle as in the case of specular reflection. Many surfaces are known to reflect light diffusely. Several surfaces also show a mixture of diffuse and specular reflection. Kd is the coefficient of diffuse reflection. In the example given below Kd is increased gradually from 0 to 1.

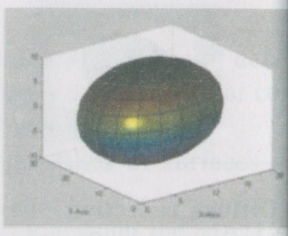
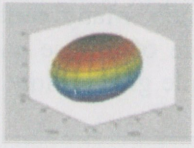
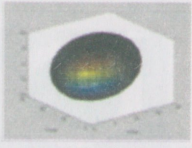
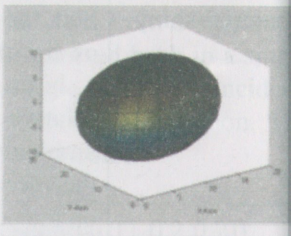
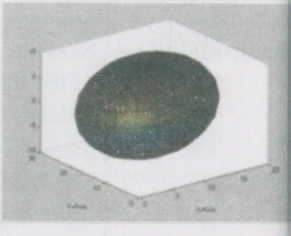
material([0.2 0.2 0])	material([0.2 0.2 0.5])	material([0.2 0.2 1])
		

The various options for materials are listed below.

Material	Property	Command	Output
material shiny	Sets the reflectance properties of the material so that it has a high specular reflectance relative to the diffuse and ambient light. The colour of the specular light depends only on the color of the light source.	material shiny	
material dull	Sets the reflectance properties of the material so that the object reflects more diffuse light and has no specular highlights, but the color of the reflected light depend only on the light source.	material dull	

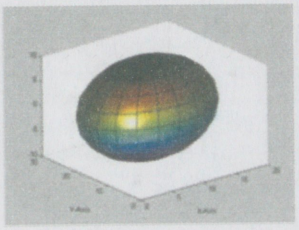
Contd.

Contd.

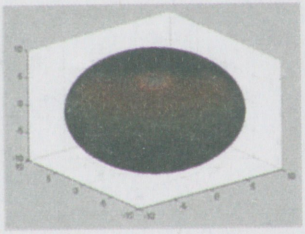
material metal	Sets the reflectance properties of the material so that the object has a very high specular reflectance and the color of the reflected light depends both on the light source and the color of the object.	material metal	
material ([ka kd ks])	Sets the ambient, diffuse and specular refection coefficient of the object.	material ([0.7 0.2 0.1])	
		material ([0.1 0.7 0.2])	
		material ([0.1 0.2 0.7])	
material ([ka kd ks n])	Sets the ambient, diffuse and specular reflection coefficient and specular exponent of the objects.	material ([0.2 0.2 0.2 4])	
material ([ka kd ks n sc])	Sets the ambient, diffuse, specular reflection coefficeint and specular exponent and specular color reflectance of the objects.	material ([0.2 0.2 0.2 4 10])	

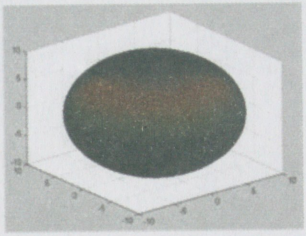
Conte

Contd.

material default	Sets the ambient/diffuse/ specular reflectance stregth, specular exponent, and specular color reflectance of the objects to their defaults values.	material default	

A few other examples are listed below.

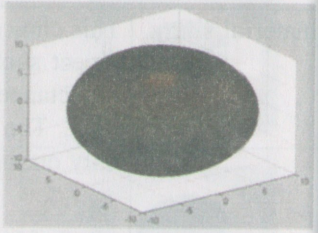
material shiny	This choice sets the reflectance properties of the object such that the object has a high specular reflectance relative to the diffuse and ambient light. The color of the specular light depends only on the color of the light source.
	`[u,v] = meshgrid((0:0.01:1)*pi, (0:0.01:2)*pi);`
	`a1=10*cos(v).*sin(u);`
	`b1=10*sin(v).*sin(u);`
	`c1=10*cos(u);`
	`h=surf(a1,b1,c1);`
	`set(h,'FaceLighting','phong','FaceColor','interp','AmbientStrength',0.8)`
	`light('Position',[0 0 20],'Style','infinite');`
	`material shiny`
		

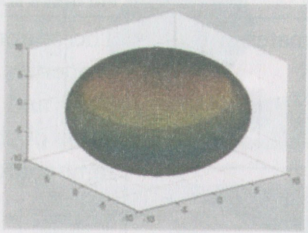
material dull	The option dull for material sets the reflectance properties so that the object reflects more diffuse light and has no specular highlights, but the color of the reflected light depends only on the light source.
	`[u,v] = meshgrid((0:0.01:1)*pi,(0:0.01:2)*pi);`
	`a1=10*cos(v).*sin(u);`
	`b1=10*sin(v).*sin(u);`
	`c1=10*cos(u);`
	`h=surf(a1,b1,c1);`
	`set(h,'FaceLighting','phong','FaceColor','interp','AmbientStrength',0.8)`
		

Contd.

Contd.

	`light('Position',[0 0 20],'Style','infinite');` `material dull`	

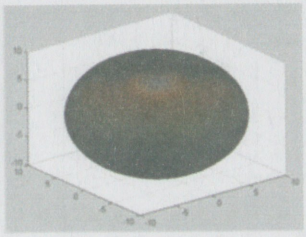
material metal	The material metal option sets the reflectance properties so that the object has a very high specular reflectance, very low ambient and diffuse reflectance, and the color of the reflected light depends on both the color of the light source and the color of the object.
	`[u,v] = meshgrid((0:0.01:1)*pi,(0:0.01:2)*pi);`
	`a1=10*cos(v).*sin(u);`
	`b1=10*sin(v).*sin(u);`
	`c1=10*cos(u);`
	`h=surf(a1,b1,c1);`
	`set(h,'FaceLighting','phong','FaceColor','interp','AmbientStrength',0.8)`
	`light('Position',[0 0 20],'Style','infinite');`
	`material metal`
		

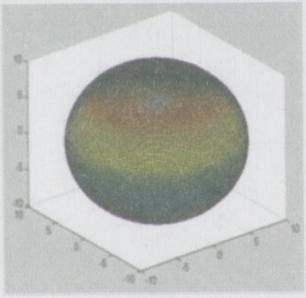
material([ka kd ks n])	The material([ka kd ks n]) option sets the ambient/diffuse/specular strength and specular exponent of the objects.
	`[u,v] = meshgrid((0:0.01:1)*pi,(0:0.01:2)*pi);`
	`a1=10*cos(v).*sin(u);`
	`b1=10*sin(v).*sin(u);`
	`c1=10*cos(u);`
	`h=surf(a1,b1,c1);`
	`set(h,'FaceLighting','phong','FaceColor','interp','AmbientStrength',0.1)`
	`light('Position',[0 0 20],'Style','infinite');`
	`material ([ 0.6 0.3 0.5 0.2])`
		

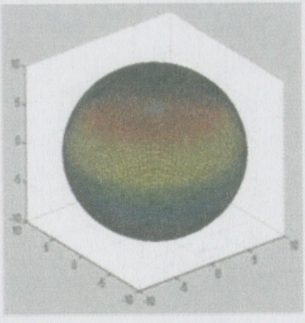
material default	The default option for material sets the ambient/diffuse/specular strength, specular exponent, and specular color reflectance of the objects to their defaults.	

Contd.

Contd.

	`[u,v] = meshgrid((0:0.01:1)*pi,(0:0.01:2)*pi);`
	`a1=10*cos(v).*sin(u);`
	`b1=10*sin(v).*sin(u);`
	`c1=10*cos(u);`
	`h=surf(a1,b1,c1);`
	`set(h,'FaceLighting','phong','FaceColor','interp','AmbientStrength',0.1)`
	`light('Position',[0 0 20],'Style','infinite');`
	`material default`
		

material([ka kd ks])	The material([ka kd ks]) option sets the ambient/diffuse/specular strength of the objects.
	`[u,v] = meshgrid((0:0.01:1)*pi,(0:0.01:2)*pi);`
	`a1=10*cos(v).*sin(u);`
	`b1=10*sin(v).*sin(u);`
	`c1=10*cos(u);`
	`h=surf(a1,b1,c1);`
	`set(h,'FaceLighting','phong','FaceColor','interp','AmbientStrength',0.1)`
	`light('Position',[0 0 20],'Style','infinite');`
	`material([0.7 2 0.9])`
		

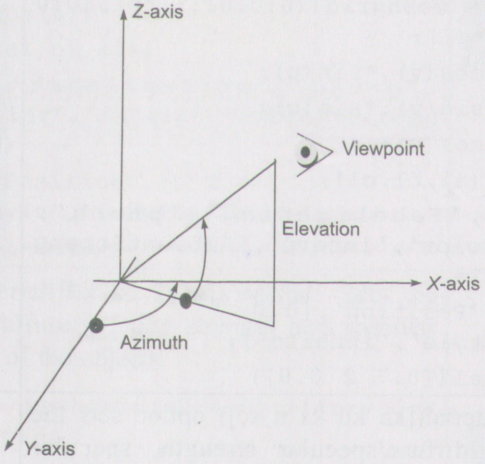
material([ka kd ks n sc])	The material([ka kd ks n sc]) option sets the ambient/diffuse/specular strength, specular exponent, and specular color reflectance of the objects.
	`[u,v] = meshgrid((0:0.01:1)*pi,(0:0.01:2)*pi);`
	`a1=10*cos(v).*sin(u);`
	`b1=10*sin(v).*sin(u);`
	`c1=10*cos(u);`
	`h=surf(a1,b1,c1);`
	`set(h,'FaceLighting','phong','FaceColor','interp','AmbientStrength',0.1)`
	`light('Position',[0 0 20],'Style','infinite');`
	`material([0.7 2 0.9 20 5])`
		

Contd.

It is thus possible to set the specular, diffuse and ambient reflection coefficient of a surfac or patch and thus affect the way they look when light source is applied.

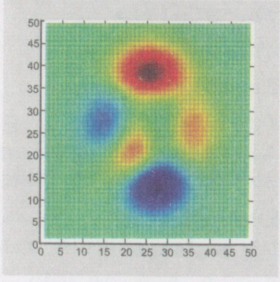
5.10 VIEW

MATLAB allows us to set the viewpoint to control the orientations of the graphics displayed i the axis. One can set the orientation of the graphics using the view command. The view comman specifies the viewpoint by fixing the azimuth and the elevation with respect to the origin. Azimut is the polar angle in the *x-y* plane. A vector from the origin to the point of interest is projecte perpendicularly onto a reference plane. The angle between the projected vector and the referenc vector on the reference plane is called the azimuth. The counter-clockwise direction of rotation i considered the positive azimuth angle. Elevation is the angle above or below the *x-y* plane. Th angle of elevation is the angle above the horizontal that an observer must look to see the object Elevation angles for objects above the horizon can range from 0° (objects on the horizon) up to 90 (objects at the zenith). The counter-clockwise direction of rotation in the *z-x* plane is considere the positive elevation angle. The figure below explains the azimuth and elevation angles.



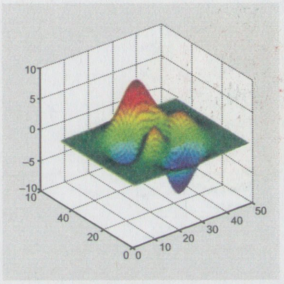
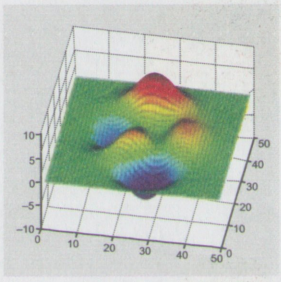
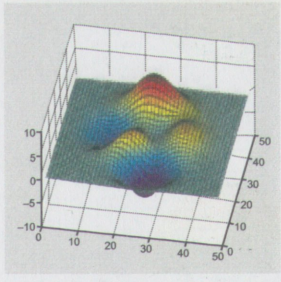
Azimuth and elevation angles

The various syntaxes available for view are:

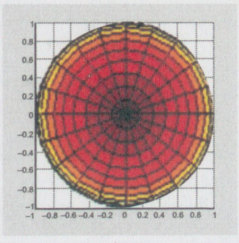
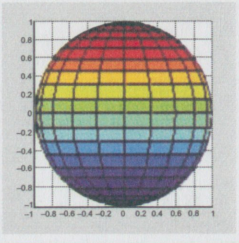
Setting the viewpoint using view	Output	Comments
>> surf(peaks) >> view(2)		view(2) sets the default 2-dimensional view having az = 0, el = 90.

Contd.

Contd.

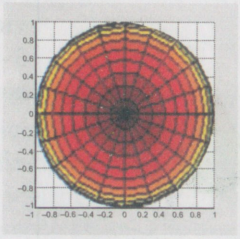
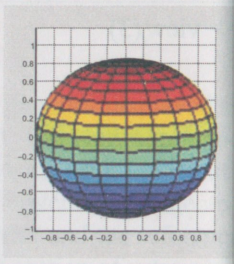
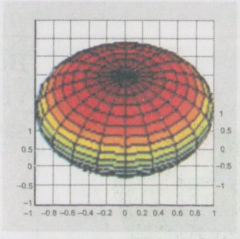
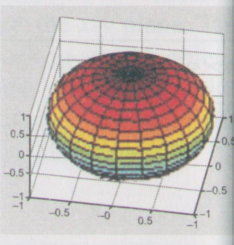
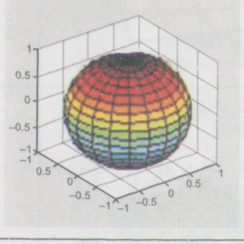
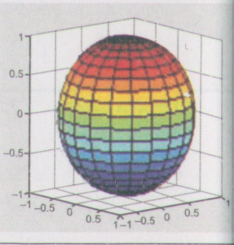
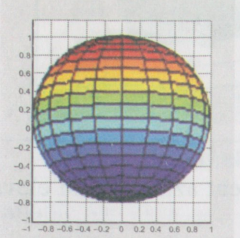
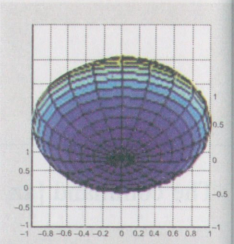
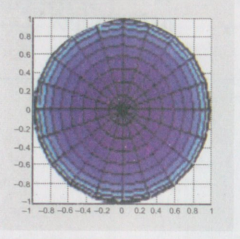
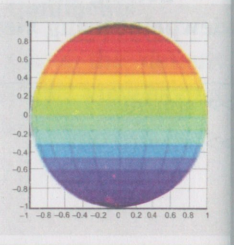
>> view(3)		view(3) sets the default 3-dimensional view having az = –37.5, el = 30.
>> view(10,50) or >> view([10,50])		Both view(10, 50) or view(([10, 50]) give the same result.
		

The command [az, el] = view returns the current azimuth and elevation. For example,

Setting the viewpoint using view	Output	Setting the viewpoint using view	Output
>> sphere >> view(0,90)		view(0,0)	

Contd.

Contd.

>> sphere >> view(0,60)		>> sphere >> view(90,90)	
>> sphere >> view(0,10)		>> view(10,50)	
>> sphere		>> view(50,10)	
>> sphere >> view(0,–10)		>> sphere >> view(0,–60)	
>> sphere >> view(0,–90)		>> sphere >> view(90,0)	

The command [az,el] = view returns the current azimuth and elevation. For example,

```
>> [az,el]=view
az =
```

37.5000

l =

30

T = view returns the current 4-by-4 transformation matrix.

>surf(peaks)

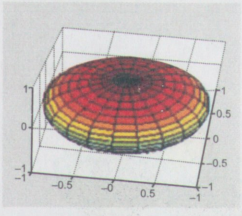
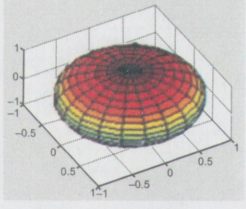
iew(2)

> T=view

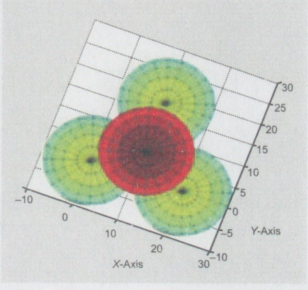
' =

1.0000	0	0	–0.5000
0	1.0000	0	–0.5000
0	0	–1.0000	9.1603
0	0	0	1.0000

In the example given below the viewpoint has been set at various combinations of azimuth nd elevation angles.

>> sphere >> view(10,60)	>> sphere >> view(60,60)
	

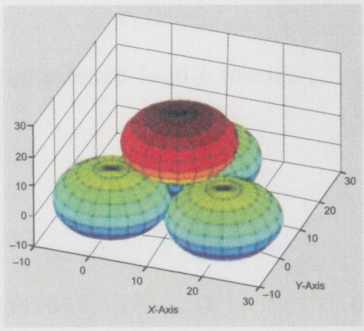
MATLAB Code	View	View angle

u,v] =
eshgrid((0:0.1:1)*pi,(0:0.1:2)
pi);
1=10*cos(v).*sin(u);
1=10*sin(v).*sin(u);
c1=10*cos(u);
surf(a1,b1,c1);
hold on
		view(20, 90)

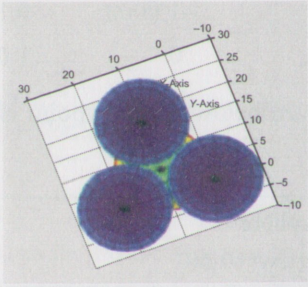
Contd.

Contd.

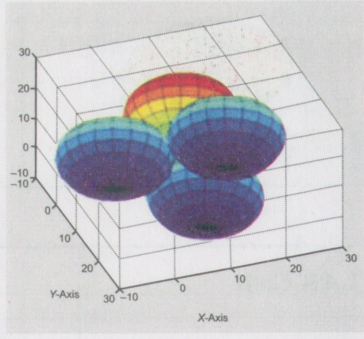
```
[u,v] =
meshgrid((0:0.1:1)*pi,(0:0
.1:2)*pi);
a2=20+10*cos(v).*sin(u);
b2=0+10*sin(v).*sin(u);
c2=0+10*cos(u);
surf(a2,b2,c2);
hold on
[u,v] =
meshgrid((0:0.1:1)*pi,(0:0
.1:2)*pi);
a3=10+10*cos(v).*sin(u);
b3=10*sqrt(3)+10*sin(v).*s
in(u);
c3=0+10*cos(u);
surf(a3,b3,c3);
xlabel('X-Axis')
ylabel('Y-Axis')
hold on
[u,v] = meshgrid((0:0.1:1)
*pi,(0:0.1:2)*pi);
a4=10+10*cos(v).*sin(u);
b4=(10/sqrt(3))+10*sin(v).
*sin(u);
c4=(2*sqrt(2)/sqrt(3))*10+
10*cos(u);
surf(a4,b4,c4);
xlabel('X-Axis')
ylabel('Y-Axis')
```



As it is



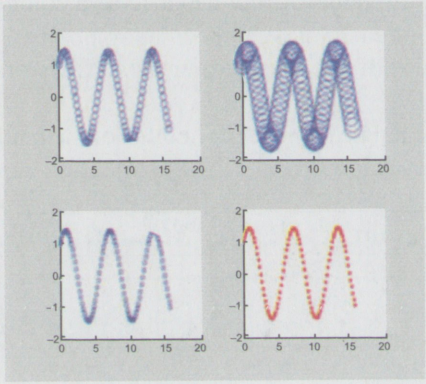
view(20, –90)



view(20, –45)

5.11 CREATING PLOTS USING MARKERS FILLED WITH COLORS AND HAVING VARYING SIZES

A 2-dimensional scatter plot can be plotted using the scatter function. The various syntax available for scatter plot are:

Command	Output	Explanation
= linspace(0,5*pi,100); = sin(x)+cos(x); ubplot(2,2,1) catter(x,y) ubplot(2,2,2) catter(x,y,300) ubplot(2,2,3) catter(x,y,10,'s') ubplot(2,2,4) catter(x,y,10,'s','r','filled')		subplot(2,2,1) scatter(X,Y) subplot(2,2,2) scatter(X,Y,size) subplot(2,2,3) scatter(X,Y,size, 'markertype') subplot(2,2,4) scatter(X,Y,size, 'markertype', 'colour','filled')

The various marker types and colors available are listed below. 'filled' fills the marker with the lor specified. It is also possible to get a handle to the line objects created by scatter plot using,

= scatter(...)

The various types of lines, markers and colors available in MATLAB are as follows:

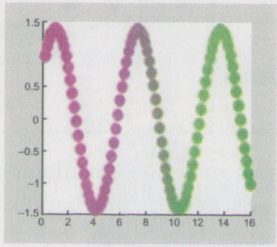
Line	Explanation	Markers	Explanation	Colour	Explanation
-	solid line (default)	+	plus	r	Red
--	dashed line	o	circle	g	Green
:	dotted line	x	cross	b	Blue
-.	dash-dot line	.	poitn	c	Cyan
		*	Star or asterix	m	Magenta
		d	Diamond	y	Yellow
		s	Square	k	Black
		^/v	Upward/ downward pointing triangle	w	White
		h	Hexagon, Six pointed star		
		p	Pentagon, five pointed star		
		>/<	Right/left pointing triangle		

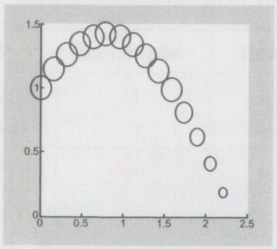
In the command scatter(X,Y, 'Size,'marker','Color'), Color determines the color of ea marker. When Color is a vector the same length as x and y, the values in Color are linea mapped to the colors in the current colormap. If Color is a length(X)-by-3 matrix, it specifies t colours of the markers as RGB values. Color can also be a color string form the list given abo

Size determines the area of each marker (specified in points^2). Size can be a vector the sa length as x and y or a scalar. If Size is a scalar, MATLAB draws all the markers having t same size.

Examples using matrix of length (X)-by-3 for colors of the markers and a matrix for the vario sizes of the markers is given below.

MATLAB Program	Output	Explanation

`x = linspace(0,5*pi,100);`
`y = sin(x)+cos(x);`
`colors = zeros(100,3);`
`r = linspace(1,0,100);`
`g = linspace(0,1,100);`
`b = linspace(1,0,100);`
`for i = 1:100`
`colors(i,:) = [r(i) g(i) b(i)];`
`end`
`scatter(x,y,300,colors,'filled');`
		colors(i,:) = [r(g(i) b(i)];
		creates a length(x) by-3 array of colour which specifie the colours of th markers as RG values.

`x = linspace(0,5*pi,100);`
`y = sin(x)+cos(x);`
`sizes = 1000*y;`
`scatter(x,y,sizes);`
		sizes = 1000*y;
		creates an array o sizes for each marke

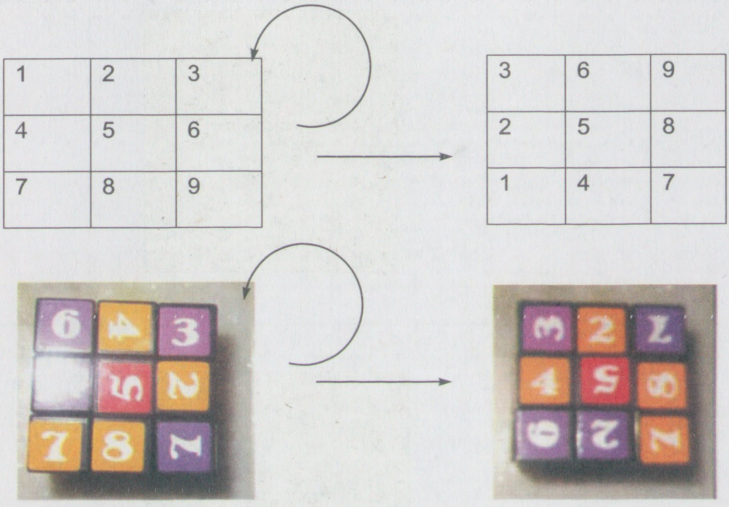
5.12 ROTATION OF AN IMAGE

A matrix can be rotated through 90° using the rot90 function in MATLAB. The rot90 (a) functi rotates a matrix by 90° in the counterclockwise direction. The function rot (a,k) rotates the matr a counterclockwise direction by k*90° where k is an integer.

The two syntaxes available for rot90 are as follows.

Syntax	Result
rot90(a)	Rotates an array through 90° in counterclockwise direction In case of multidimensional arrays the plane formed by firs and second dimensions is rotated through 90°.
rot90(a,k)	Rotates the matrix a through k × 90° in counterclockwis direction. Here k is an integer.

The rot90 function rotates a matrix by 90° in the counterclockwise direction. This has been ustatred below.

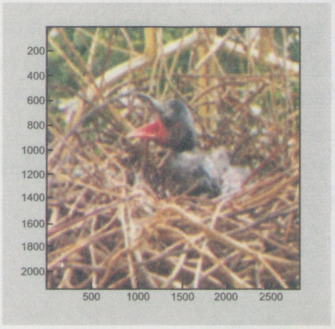


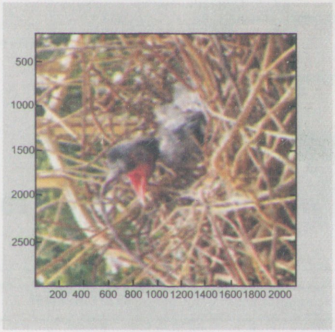
Rotation by 90°

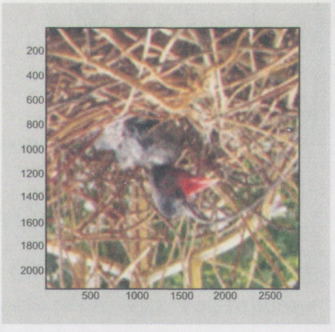
Array	Rotation	Output
> a=[1 2 3;4 5 6;7 8 9] = 1 2 3 4 5 6 7 8 9	>> rot90(a)	ans = 3 6 9 2 5 8 1 4 7
	>> rot90(a,2)	ans = 9 8 7 6 5 4 3 2 1
	>> rot90(a,4)	ans = 1 2 3 4 5 6 7 8 9

As a digital image is an array it is possible to rotate this array also. This will result in a rotated age.

Program	Output	Explanation

`>> cd d:\pictures`
`i=imread('bird.jpg');`
`>> figure,image(i)`
		Original image

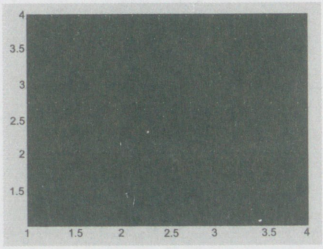
`>>`
`cd d:\pictures`
`i=imread('bird.jpg');`
`figure,image(i);`
`j(:,:,1)=rot90(i(:,:,1),1);`
`j(:,:,2)=rot90(i(:,:,2),1);`
`j(:,:,3)=rot90(i(:,:,3),1);`
`figure,image(j);`
`>>`
		Rotated once by 90 in counterclockwise direction

`>> cd d:\pictures`
`i=imread('bird.jpg');`
`figure,image(i);`
`j(:,:,1)=rot90(i(:,:,1),1);`
`j(:,:,2)=rot90(i(:,:,2),1);`
`j(:,:,3)=rot90(i(:,:,3),1);`
`hold on`
`k(:,:,1)=rot90(j(:,:,1),1);`
`k(:,:,2)=rot90(j(:,:,2),1);`
`k(:,:,3)=rot90(j(:,:,3),1);`
`image(k)`
		Rotated twice by 90 in counterclockwise direction

5.13 pcolor

A pcolor or pseudocolor plot is a rectangular array of cells. The color of each cell is determin by A.The shading is faceted by default. Each cell is filled with a single color. The last row a column of the matrix A is not used. The minimum and maximum elements of array A are assign the first and last colors in the colormap.

>> pcolor(hadamard(4))



> hadamard(4)

ns =

```
1   1   1   1
1  -1   1  -1
1   1  -1  -1
1  -1  -1   1
```

A Hadamard matrix is a square matrix whose entries are either +1 or −1 and whose rows are nutually orthogonal. It is named after the French mathematician Jacques Hadamard. Any two ifferent rows in a Hadamard matrix represent two perpendicular vectors this, in turn, means that n every two different rows have matching values in exactly half of the columns and mismatched alues in half of the remaining columns. The order of a Hadamard matrix must be 1, 2, or a nultiple of 4. A Hadamard matrix is a type of square matrix having elements as 1 and –1. In a Iadamard matrix, considering any two columns or rows next to each other gives half the adjacent ells similar sign and the other half having a different sign. The $n \times n$ Hadamard matrix (H_n) must ave $n(n-1)/2$ squares –1 and $n(n-1)/2$ squares 1.

For example,

> hadamard(4)

ns =

```
1   1   1   1
1  -1   1  -1
1   1  -1  -1
1  -1  -1   1
```

Has $4(4-1)/2 = 6$ squares with –1 and $4(4+1)/2 = 10$ squares with +1.

> hadamard(8)

ns =

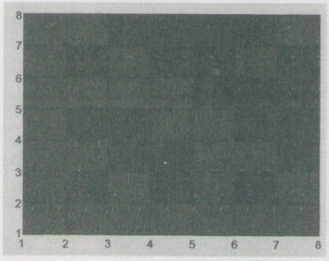
```
1   1   1   1   1   1   1   1
1  -1   1  -1   1  -1   1  -1
1   1  -1  -1   1   1  -1  -1
1  -1  -1   1   1  -1  -1   1
```

Contd.

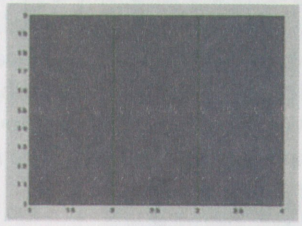
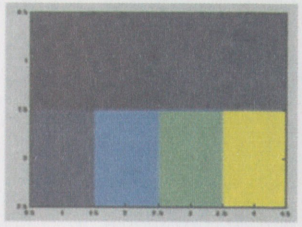
Contd.

```
     1     1     1     1    -1    -1    -1    -1
     1    -1     1    -1    -1     1    -1     1
     1     1    -1    -1    -1    -1     1     1
     1    -1    -1     1    -1     1     1    -1
```

```
>> pcolor(hadamard(8))
```



It should be noted that the pcolor does not show the last column and row of the matrix.

>> A=[1 2 3 4;10 20 30 40]; >> pcolor(A) >>		<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>10</td><td>20</td><td>30</td><td>40</td></tr></table>
>> A=[1 2 3 4;10 20 30 40]; >> image(A) >>		This is not the case when we use image(A). Here the number of cells is 8.

5.14 READING AN IMAGE FROM AN URL

URL stands for uniform resource locator. It is also sometimes known as the web addres particularly when it is used with http. It is a specific character string which constitutes th reference to a resource. In most web browsers the url is displayed on top in the address bar.

For example the URL of NIT Rourkela is.

http://www.nitrkl.ac.in

The MathWorks website URL is,

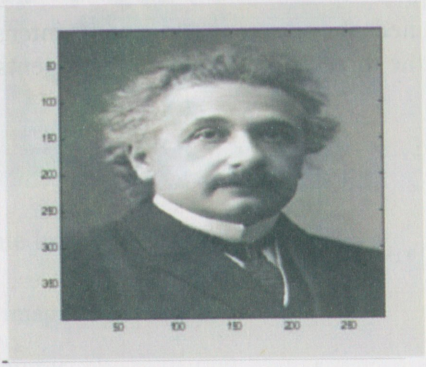
ttp://www.mathworks.com/index.html?s_tid=gn_loc_drop

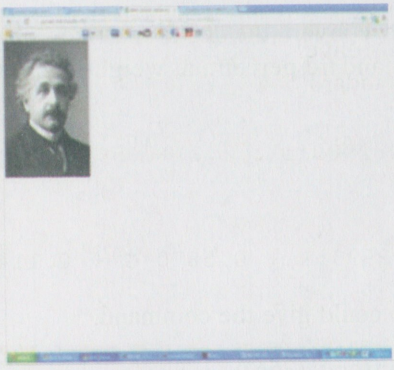
URL in the browser



It is possible to upload an image form an url in the igure window.

```
rl='http://upload.wikimedia.org/wikipedia/commons/5/50/Albert_Einstein_%28Nobel%29.
ng';
> i=imread(url);
> image(i)
```





5.15 CHANGING THE BACKGROUND COLOR USING figure('color',c)

```
>> figure('color','r')
>> cd d:\pictures
=imread('bird.jpg');
mage(i)
axis off
```



5.16 CONVERT A 8-BIT RGB IMAGE TO A GRAYSCALE IMAGE

8-bit color is a true direct color system. The platter can store 4, 16 or 256 distinct colo These limits are imposed by the architecture of the display. 256 values can fit in a sing 8-bit byte. 8-bit color graphics is a method in which an image file is stored such that ea pixel of the image is represented by a 8-bit byte. Thus, the maximum number of colors th can be displayed is $2^8 = 256$. In computer one byte has 8 bits and can store $2^8 = 256$ valu ranging from 0 to 255.

In order to convert a 8-bit RGB image first, separate the color indexed image into its differe RGB components. This can be done using the equations given below:

```
>> cd d:\pictures
>> i=imread('aadil.jpg','jpg');
>> image(i)
>> R=i(:,:,1);
>> G=i(:,:,2);
>> B=i(:,:,3);
```

Next we convert the RGB matrices obtained above into a gray-scale intensity imag using the standard perceptual weightings for the three different color components red, gree and blue.

```
>> j = 0.2989*i(:,:,1)+0.5870*i(:,:,2)+0.1140*i(:,:,3);
```

Or

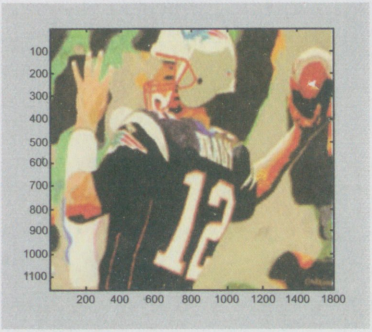
```
>>j = 0.2990*R + 0.5870*G + 0.1140*B;
```

Then we could give the command,

```
>> colormap(gray (256))
image(j)
```

The entire program is given below.

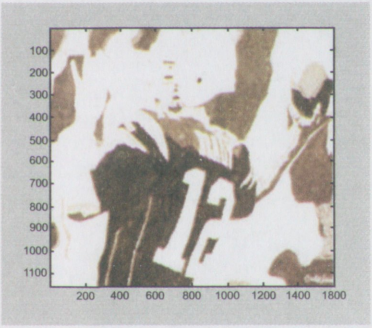
```
>> cd d:\pictures
>> i=imread('aadil.jpg','jpg');
>> image(i)
>> j = 0.2989*i(:,:,1)+0.5870*i(:,:,2)+0.1140*i(:,:,3);
>> colormap(gray(256))
image(j)
```



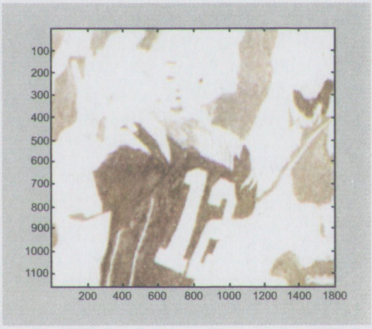
Original Truecolor image RGB



Grayscale image with 256 shades of gray



Grayscale image with 100 shades of gray



Grayscale image with 50 shades of gray

First an image is uploaded.

```
> cd d:\pictures
> i=imread('fog.jpg');
> image(i)
```

Then we find out the monochrome luminescence by combining the RGB values according o the NTSC standard. This applies the coefficients related to the eye's sensitivity to TGB olors.

```
> j = 0.2989*i(:,:,1)+0.5870*i(:,:,2)+0.1140*i(:,:,3);
> colormap(gray(256))
> image(j)
```

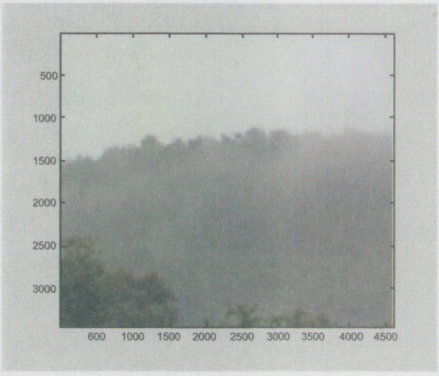


image (i)

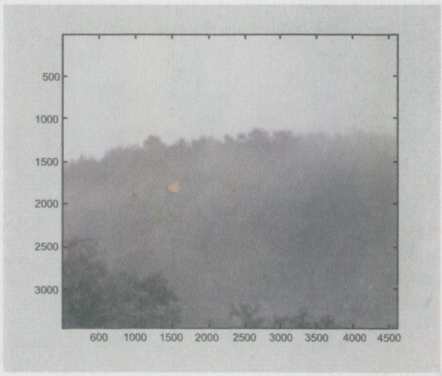


image (ii)

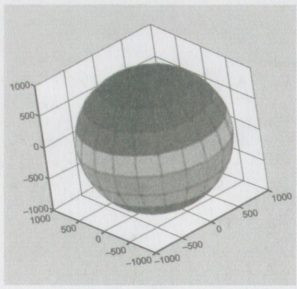
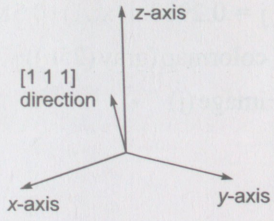
It should be noted here that the value in the j matrix lie within 0 and 256.

```
>> min(j(:))
ans =
    16
>> max(j(:))
ans =
    213
```

5.17 camup

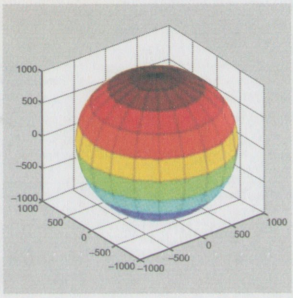
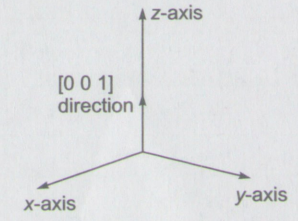
camup returns the camera up vector setting in the current axes. The camera up vector specifie the direction that is oriented up in the scene.

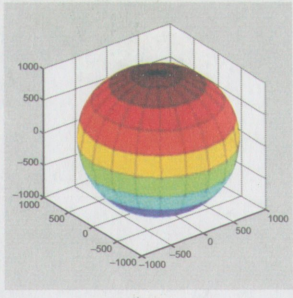
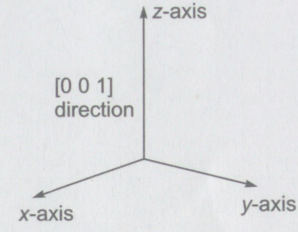
MATLAB Program	Output	Explanation

`camup([vector])`
`for i=1:100`
`[u,v] =`
`meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a1=10*i*cos(v).*sin(u);`
`b1=10*i*sin(v).*sin(u);`
`c1=10*i*cos(u);`
`surf(a1,b1,c1);`
`end`
`camup([1 1 1])`
		The camup([vector]) command sets the up vector in the current axes to the specified value. The up vector is specified as *x*, *y*, and *z* components in vector [*x y z*]. In this example the up vector in the current axis is as shown below.
		 

Contd

Contd.

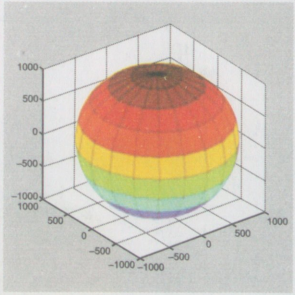
camup('mode')
`for i=1:100`
`[u,v] =`
`meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a1=10*i*cos(v).*sin(u);`
`b1=10*i*sin(v).*sin(u);`
`c1=10*i*cos(u);`
`surf(a1,b1,c1);`
`end`
`camup('mode')`
	ans =
	auto
	 
		The camup('mode') returns the current value of the camera up vector mode, which can either be auto (default) or manual.auto is the default camera up vector mode.
		In this example the up vector in the current axis is [0 0 1] which is the auto mode. This is also the default mode. This direction is shown below.
		 

camup('auto')
`for i=1:100`
`[u,v] =`
`meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a1=10*i*cos(v).*sin(u);`
`b1=10*i*sin(v).*sin(u);`
`c1=10*i*cos(u);`
`surf(a1,b1,c1);`
`end`
`camup('auto')`
		The camup('auto') sets the camera up vector mode to auto. In auto mode, [0 1 0] is the up vector of for 2-D views. This means the y-axis points up. For 3-D views, the up vector is [0 0 1], which implies that the *z*-axis points up. The up vector [0 0 1] is the auto mode. This has been shown below.
		 

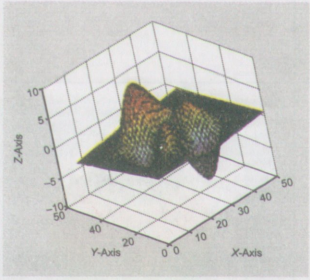
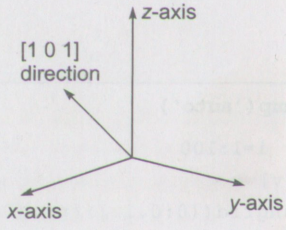
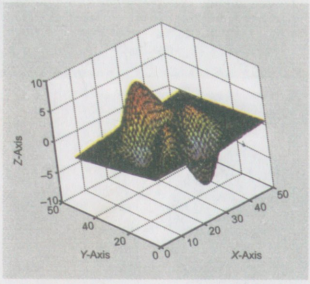
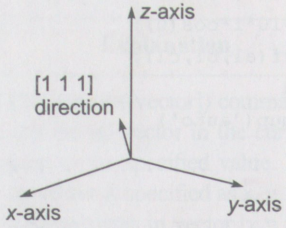
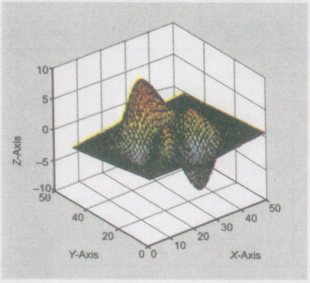
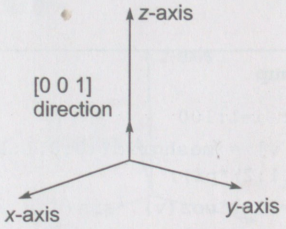
camup
`for i=1:100`
`[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a1=10*i*cos(v).*sin(u);`
`b1=10*i*sin(v).*sin(u);`
	ans =
	0 0 1
	>>
		The camup returns the camera up vector setting in the current axes. The camera up vector specifies the direction that is oriented up in the scene.

Contd.

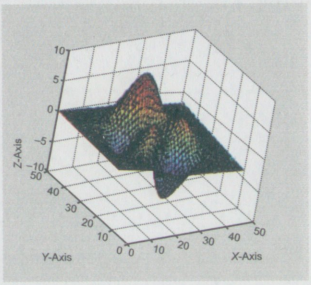
```
c1=10*i*cos(u);
surf(a1,b1,c1);
end
>> camup
```

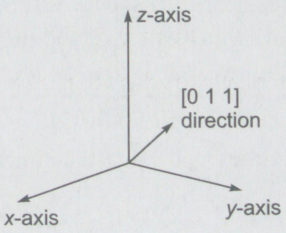


A few more examples using the camup function are given below.

MATLAB Commands	Output	The up vector specified as *x*, *y*, and *z* components in vector [x y z] is shown below
>> surf(peaks) >> xlabel('X-Axis'); >> ylabel('Y-Axis'); >> zlabel('Z-Axis'); >> camup([1 0 1])		
>> camup([1 1 1])		
>> camup([0 0 1])		

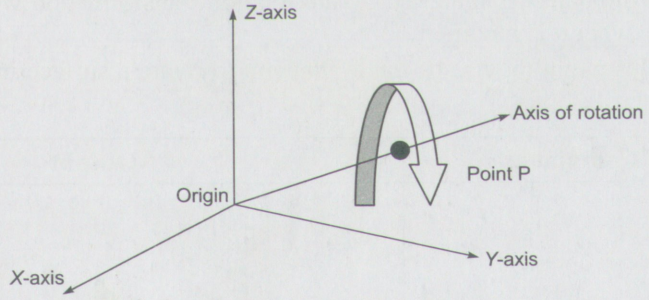
```
>> camup([0 1 1])
```





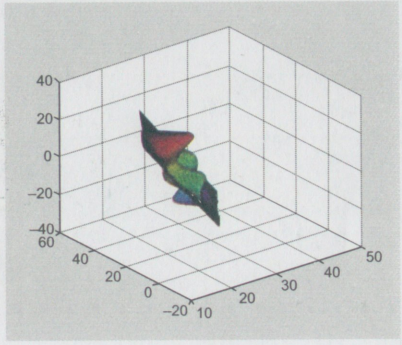
5.18 rotate

The function rotate in MATLAB rotates a graphics object in three-dimensional space about a specified direction according to the right-hand rule. It should be noted that the graphics object which we want to rotate must be a child of the same axes. The axis of rotation is defined by an origin and a point P relative to the origin. P can be expressed in the spherical coordinates using the [theta phi] angles, or in Cartesian coordinate.



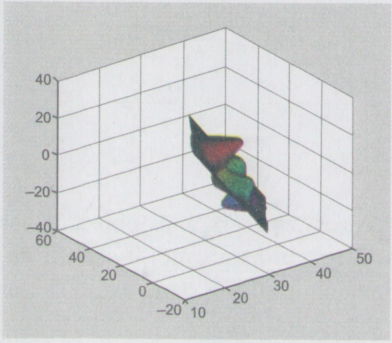
rotate(h, direction, angle) rotates the graphics object h by angle degrees about an axis of rotation the direction of which is a two- or three-element vector that described in conjunction with the origin. For example,

```
>> h=surf(peaks(50));
>> rotate(h,[1 1 1],90)
```



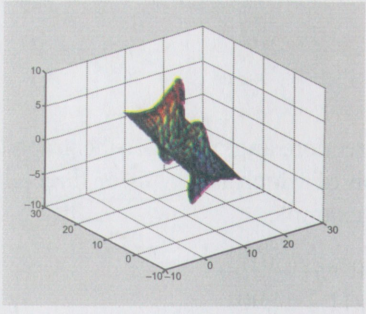
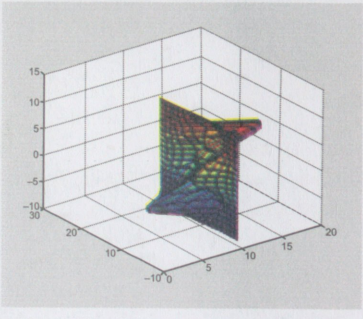
rotate(...,origin) specifies the origin of the axis of rotation as a three-element vector. The defaul origin is the center of the plot box.

```
>> h = surf(peaks(50));
>> center = [50 50 50];
>> rotate(h,[1 1 1],90,center)
```



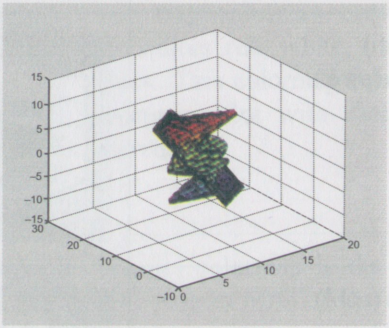
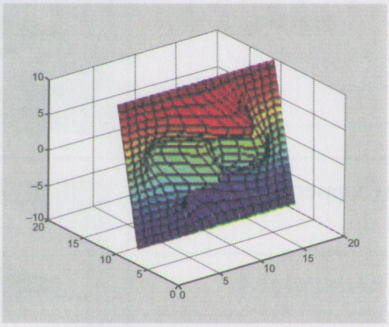
Here the data of the object is modified by the rotation transformation whereas in view and rotate3d only the viewpoint is modified.

Few examples illustrating the use of rotate function to rotate a surface in three-dimensional space is given below.

MATLAB Commands	Output
>> ima=peaks(20); >> sr=surf(ima); >> sr=surf(ima); >> dir=[1 1 1]; >> rotate(sr,dir,30) >>	
>> rotate(sr,dir,90)	

Contd

Contd.

rotate(sr,dir,80)	
>> ima=peaks(20); >> sr=surf(ima); >> dir=[1 0 0]; >> rotate(sr,dir,80) >>	

5.19 Colormaps

In MATLAB colormaps are 3 by m arrays consisting of double precision floating point numbers. The numbers are in the range of 0 to 1. MATLAB color maps cannot have integer values. True color images of class double have data values which are floating point numbers in the range of 0 to 1. In true color image of class unit8 the data values are integer numbers in the range 0 to 255 and for true color images of class unit16 the data lies in the range of 0 to 65535. A .jpg format file is loaded. This image is displayed as a transformed intensity image as a heat map.

A colormap is an m-by-3 matrix of real numbers between lying between 0.0 and 1.0. The i-th row of the colormap matrix defines the i-th color by specifying the intensity of red, green and blue.

For example,

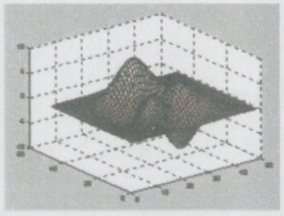
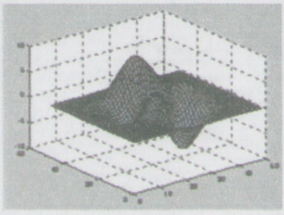
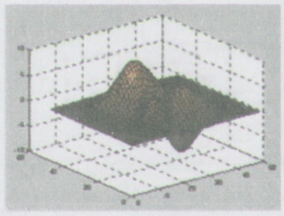
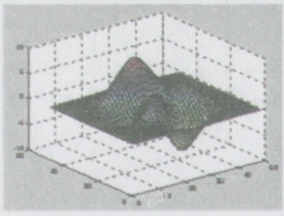
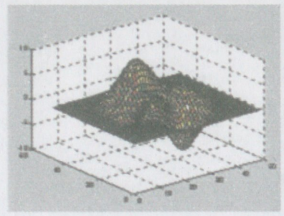
map(i,:) = [r(i) g(i) b(i)].

[0 0 0] would give black and [1 1 1] would give white.

colormap(map) will set the colormap to the matrix map. An error message is shown in case a value is not in the interval 0 to 1.

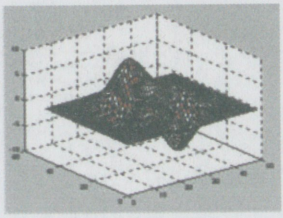
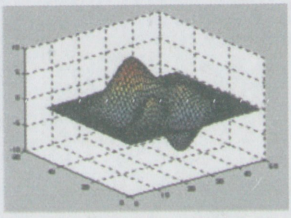
colormap('default') and colormap(hsv) set the current colormap to the default colormap.

map = colormap shows the current colormap being used.

MATLAB Command	Output	Explanation
surf(peaks) colormap(pink)		The pinkcolormap contains pastel shades of pink. The pinkcolormap provides sepia tone colorization of grayscale photographs.
colormap(cool)		The coolcolormap consists of colors that are shades of cyan and magenta. It varies smoothly from cyan to magenta.
colormap(copper)		The coppercolormap varies smoothly from black to bright copper.
colormap(hsv)		The hsv colormap varies the hue component of the hue-saturation-value color model. The colors begin with red, pass through yellow, green, cyan, blue, magenta, and return to red. The colormap is particularly appropriate for displaying periodic functions.
colormap(prism)		The prismcolormap repeats the six colors red, orange, yellow, green, blue, and violet.

Contd

ntd.

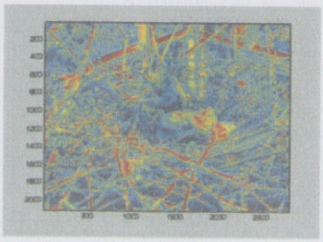
olormap (flag)		The flag colormap consists of the colors red, white, blue, and black. This colormap completely changes color with each index increment.
olormap ('default')		The jet colormap is the default colormap in MATLAB.

A digital image has been shown using various colormaps.

MATLAB Command	Output	Explanation
> cd d:\pictures =imread ('bird.jpg'); > image (i) > cd d:\pictures > i=imread ('bird.jpg'); > imean=mean(i,3); > image (imean)		Original image
> colormap (hot (256))		The hot colormap varies smoothly from black through shades of red, orange, and yellow, to white.

Contd.

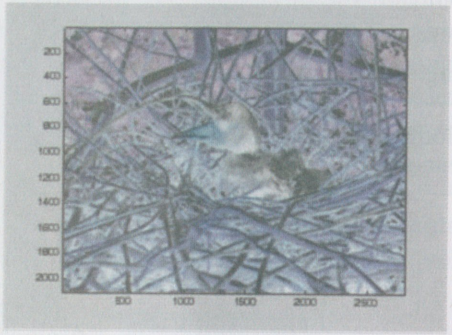
Contd.

>> colormap(copper)		The coppercolormap vari smoothly from black to brig copper.
>> colormap(jet)		The jet colormap range from blue to red, and passe through the colors cya yellow, and orange. It is variation of the hsv colorma The jet colormap is associate with an astrophysical fluid j simulation from the Nationa Center for Supercompute Applications.

It is also possible to get the complement of a colormap. For example:

```
>> cd d:\pictures
>> i=imread('bird.jpg');
>> image(i)
>> j=255-i;
>> image(j)
```





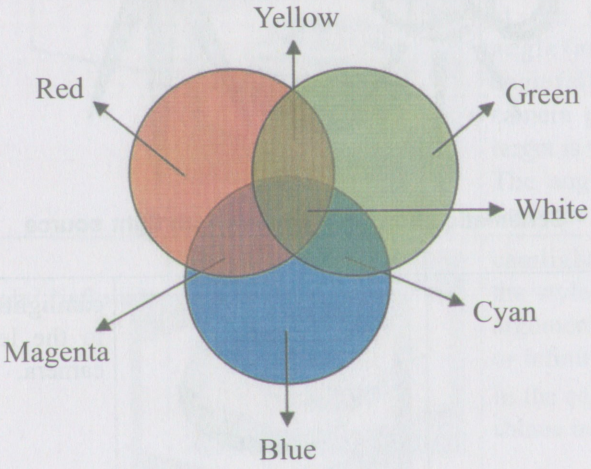
5.20 RGB IMAGE

The RGB colour model is an additive model in which red, green and blue colors are adde together in various intensities to create a wide range of colors. Red, green and blue are th additive primary colors. In the RGB color system all the colors are constructed from the red (R

een (G) and blue (B) colors. For example, each pixel in a LCD monitor displays colors in this anner by combination of red, green and blue light emitting diodes (LEDs).

In order to generate various other colors the red, green and blue LEDs are set to emit light at rious intensities and the combination of the intensities generates a color. The red, green and ue use 8 bits each to store information regarding their intensities. As a result $2^8 = 256$ different tegers can be stored in the 8 bits which have integer values from 0 to 255.

So 256*256*256 = 16777216 possible colors can be generated. When the red pixel is set to it the red LED is turned off and when the red pixel is set to 255, the red LED is turned fully d the red colour is emitted at maximum intensity. Any value in between 0 and 255 them sets e LED to partial light emission.




```
> cd d:\pictures
> i=imread('bird.jpg');
> red = i(:,:,1);
> green = i(:,:,2);
> blue = i(:,:,3);
> j=cat(3,red,blue,green);
> image(j)
```



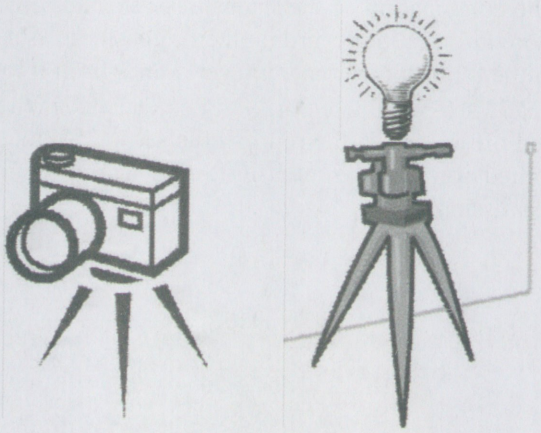
Original image



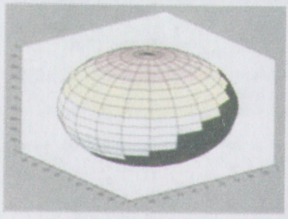
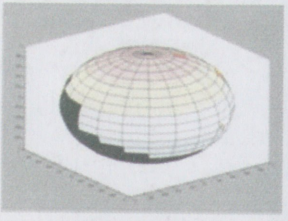
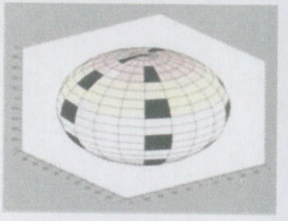
RGB image

5.21 camlight

camlight creates or moves light object in camera coordinates.

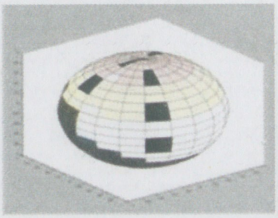
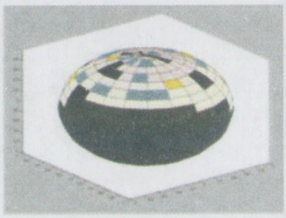
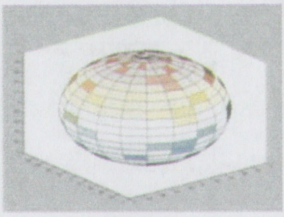
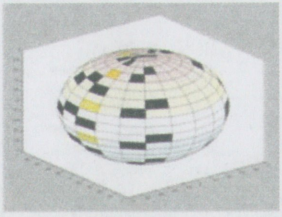


Schematic showing a camera with light source

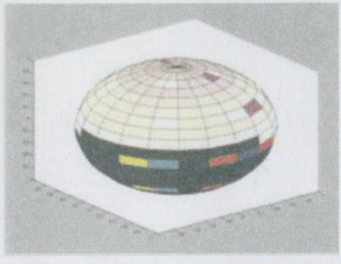
>> sphere >> camlight('left')		camlight('left') creates a lig to the left and up from th camera.
>> sphere >> camlight('right')		camlight('right') creates light to the right and up fro the camera.
>> sphere >> camlight('headlight')		camlight('headlight') create a light at the camera position This option a light source a the camera position and thu providing a headlight tha moves along with the camera

Cont

ontd.

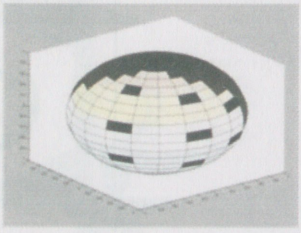
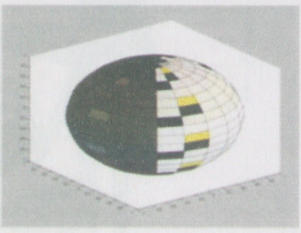
>> sphere >> camlight		camlight with no arguments is the same as camlight('right').
>> sphere >> camlight(50,90)		camlight(az,el) creates a light at the specified azimuthal angle (az) and elevation angle (el) with respect to the camera position. The camera target is the center of rotation. The angles az and el are in degrees.
>> sphere >> camlight('headlight','infinite')		camlight(...,'style') defines the style argument. The style argument can be either local or infinite. In the case of infinite the light shines in parallel rays.
>> sphere >> camlight('headlight','local')		In the case of local the light is a point source that radiates from the location in all directions. Local is also the default style.

A few other examples using camlight function is given below.

>> sphere >> camlight(0,70)	

Contd.

Contd.

>> sphere >> camlight(0,-45)	
>> sphere >> camlight(90,0)	

5.22 INSERT ELLIPSE, ANNOTATION OR DOUBLE ARROW IN A FIGURE

An ellipse can be inserted in an image using the following commands.

```
>> cd d:\Pictures
a=imread('bird.JPG','JPG');
image(a)
annotation('ellipse',...
[0.271769662921348 0.717808439178007 0.152571428571429 0.161904761904768],...
'FaceColor',[0.854901960784314 0.701960784313725 1]);
```



An annotation can be inserted in an image using the following commands.

```
> cd d:\Pictures
=imread('bird.JPG','JPG');
nage(a)
> annotation('rectangle',...
0.242071428571429    0.15952380952381    0.207928571428571
.214285714285716],...
FaceColor',[1 1 0]);
```



A double arrow can be inserted in an image using the following commands.

```
> cd d:\Pictures
=imread('bird.JPG','JPG');
nage(a)
> annotation('doublearrow',[0.259831460674157 0.546348314606742],...
0.383769539078156  0.348697394789579],'Head2Length',20,'Head2Wid
h',20,...
Head1Length',20,...
Head1Width',20,...
Color',[0 1 1]);
```




Various options in insert allow us to insert legend, line, arrow, double arrow, rectangle, ellipse etc.
Double clicking the inserted option opens a window which allows us to modify the added element.

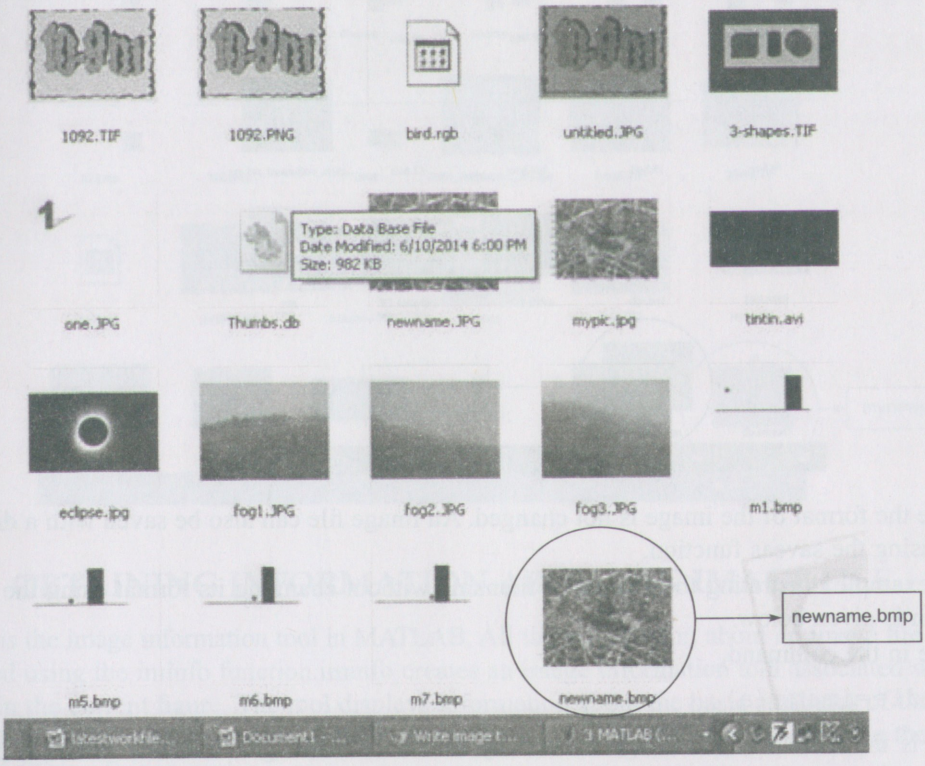

This window allows us to modify the added elements line style, width, colour, head style, head width, head length etc.


Inspector window
When the More Properties... option is selected then the inspector window is opened giving the details about an added elemer

5.23 SAVING AN IMAGE WITH A NEW FILENAME

It is possible to change the filename of an image using the imwrite function.imwrite (A, filename) writes image data A to the file specified by filename, inferring the file format from the extension.imwrite creates a new file in your current folder. For example,

```
>> cd D:\Pictures
a=imread('bird.JPG','JPG');
image(a)
>> imwrite(a,'newname.bmp','bmp')
```



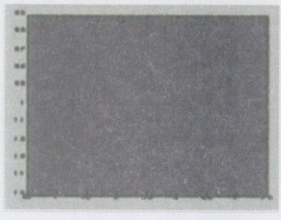
The original bird.jpg file is now saved as newname.bmp. Note that the format of the image is also changed and the new file is located in the same folder.

Another example using imwrite is given below.

```
>> cd D:\Pictures
>> a=imread('bird.JPG','JPG');
>> image(a)
>> imwrite(a,'newname.JPG','JPG')
```

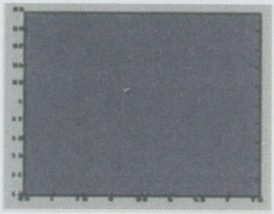
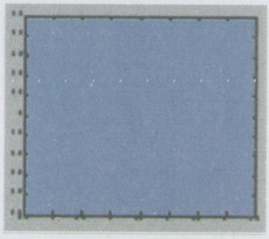
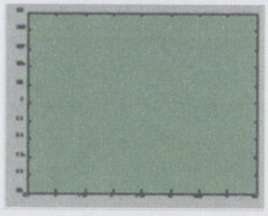
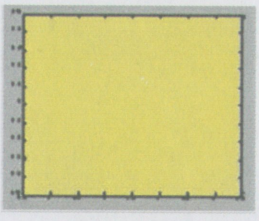
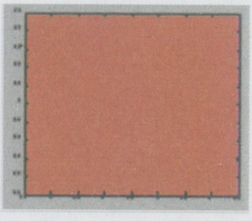
```
        Filename: 'D:\Pictures\bird.JPG'
    FileModDate: '15-Apr-2010 12:16:46'
        FileSize: 2959872
          Format: 'jpg'
    FormatVersion: ''
           Width: 2816
          Height: 2112
        BitDepth: 24
       ColorType: 'truecolor'
    FormatSignature: ''
    NumberOfSamples: 3
       CodingMethod: 'Huffman'
       CodingProcess: 'Sequential'
            Comment: {}
              Make: 'Panasonic '
              Model: 'DMC-FX01 '
        Orientation: 1
        XResolution: 72
        YResolution: 72
    ResolutionUnit: 'Inch'
          Software: 'Ver.1.0 '
          DateTime: '2009:03:13 15:13:45 '
  YCbCrPositioning: 'Co-sited'
DigitalCamera: [1x1 struct]
 UnknownTags: [1x1 struct]
ExifThumbnail: [1x1 struct]
```

5.25 VARIATION OF COLORS WITH THE VARIATION IN THE VALUES IN THE ARRAY

>> a=[0 0 0 0]; >> image(a)		As the values in the array are increased the color changes from blue end  to the red end  .

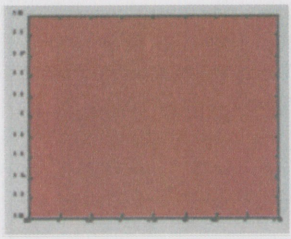
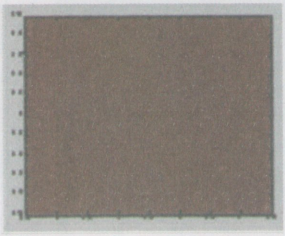
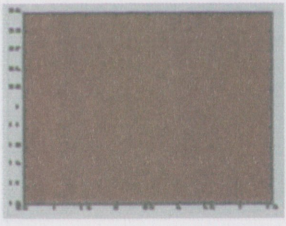
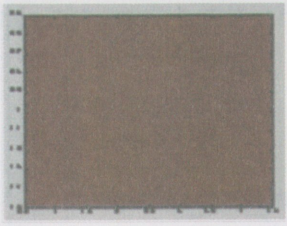
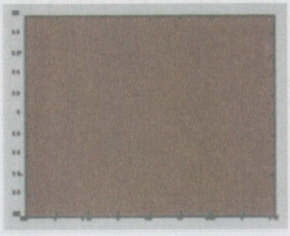
Contd

Contd.

>> a=[10 10 10 10]; >> image(a)		
>> a=[20 20 20 20]; >> image(a)		
>> a=[30 30 30 30]; >> image(a)		
>> a=[40 40 40 40]; >> image(a)		
>> a=[50 50 50 50]; >> image(a)		

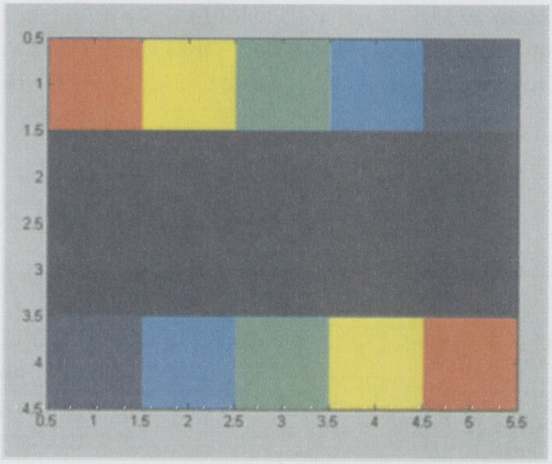
Contd.

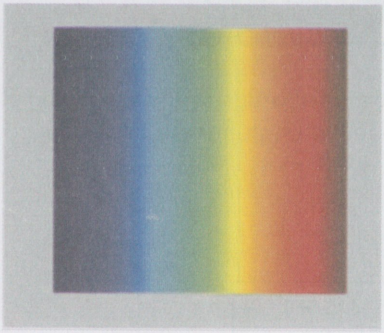
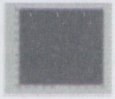
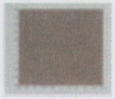
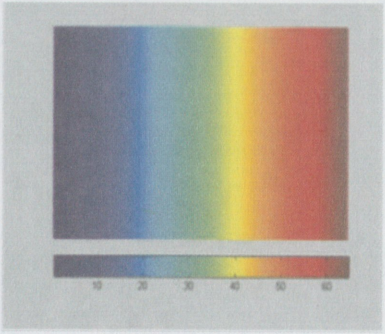
Contd.

>> a=[60 60 60 60]; >> image(a)		
>> a=[70 70 70 70]; >> image(a)		
>> a=[80 80 80 80]; >> image(a)		
>> a=[90 90 90 90]; >> image(a)		
>> a=[100 100 100 100]; >> image(a)		

A range of different colors has been generated in the examples given below.

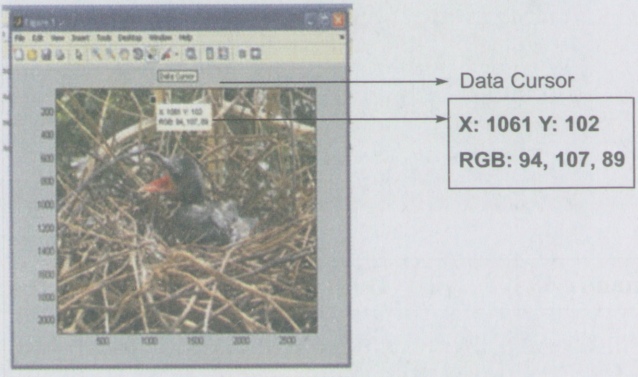
```
>> A=[50 40 30 20 10;0 0 0 0 0;0 0 0 0 0;10 20 30 40 50];
>> image(A)
```



MATLAB Command	Output	Expalanation
>> image(1:64) >> axis off		>> image(1)  >> image(64)  All the shades between these two colors are generated by the command, image(1:64)
>> image(1:64) >> axis off >> colorbar SouthOutside		It is also possible to add a color bar to the image. The various possible locations of colorbar are given below. North: Inside plot box near top South: Inside bottom East: Inside right West: Inside left NorthOutside: Outside plot box near top SouthOutside: Outside bottom EastOutside: Outside right WestOutside: Outside left

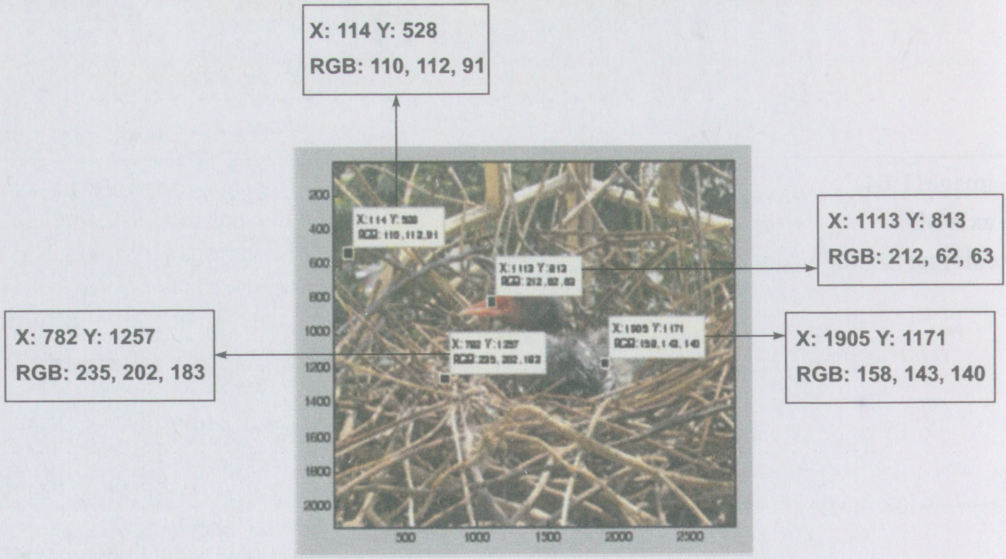
It is also possible to know the RGB value of any point in an image. If the image is loaded i the figure window we can click the Data Cursor option as shown below. After clicking the Da Cursor option we can click on any point in the image to know its RGB value.

```
>> cd d:\pictures
i=imread('bird.jpg');
image(i)
```

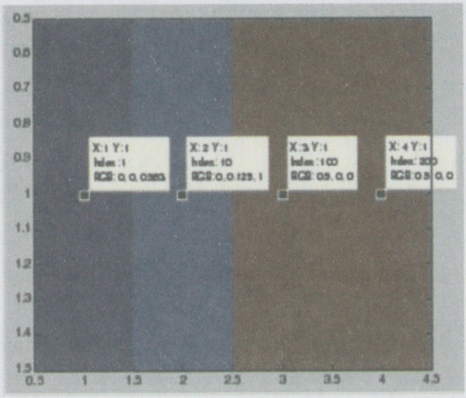


If we press alt and click on the image this will display the RGB values of all the poin otherwise only the RGB value of the latest point will be displayed

```
>> cd d:\pictures
>> i=imread('bird.jpg');
>> image(i)
```




```
>> a=[1 10 100 200];
>> image(a)
```

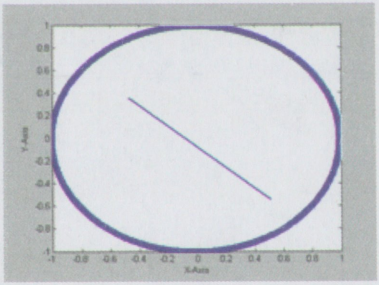


26 ADD LINE AND TEXT TO AN IMAGE

26.1 gline

ne adds line to a plot interactively.

For example,

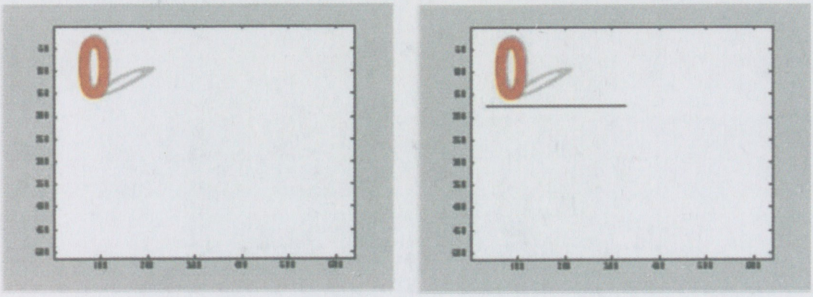


```
1;
eta=0:0.01:2*pi;
r*cos(theta);
r*sin(theta);
ot(x,y,'linewidth',8)
abel('X-Axis')
abel('Y-Axis')
```

>> gline

Here a circle is first drawn. Then a line has been added to the circle using gline. The two ints between which the line is drawn is selected by the user. Another example is given below.

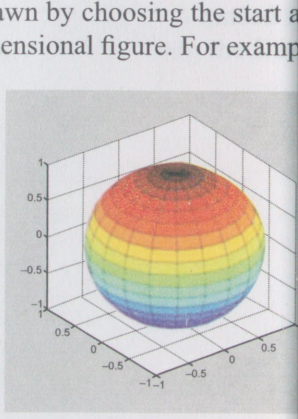
```
cd d:\pictures
i=imread('zero.jpg');
image(i)
gline
```



A line added to an image using gline

Here an image is uploaded in the figure window and a line is drawn by choosing the start a finish points. It should be noted that gline can not be used for 3-dimensional figure. For examp

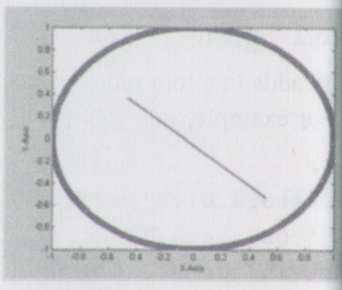
```
>> sphere
>> gline
??? Error using → gline at 40
GLINE works only for 2-D plots.
>>
```



h= gline

returns a handle to the line. For example,

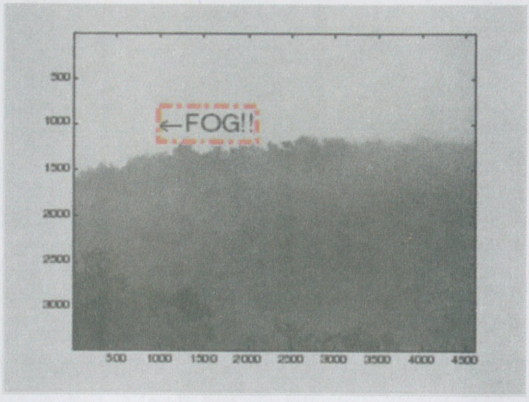
```
r=1;
theta=0:0.01:2*pi;
x=r*cos(theta);
y=r*sin(theta);
plot(x,y,'linewidth',8)
xlabel('X-Axis')
ylabel('Y-Axis')
>> h=gline;
>> set(h,'color','b')
>>
```



5.26.2 text

Text can be added to the image using the text function.

```
>> cd d:\pictures
>> i=imread('fog.jpg');
>> image(i)
>> text(1000,1000,'\leftarrowFOG!!','fontsize', 20,'edgecolor','red','linestyle','- .','linewidth',
>>
```



Text added to an image

26.3 gtext

ext function allows mouse placement of a text string in a two-dimensional view in the current gure window. The location of this text is selected using two cross hairs which can be moved ith the help of the mouse. The syntaxes for gtext are:

ext('string')

gtext('string') displays a string on the figure window when we press the mouse button or a yboard key at the desired point where the cross hairs intersect. The cross hair can be moved ing the mouse and the intersection of the cross hairs can be used to select the location of the xt.

An example is given below,

```
cd d:\pictures
i=imread('flowers2.jpg');
image(i)
gtext('A flower')
```



Cross hairs and text in the image

26.4 h=gtext('string')

gtext('string') will return the handle to a text graphics object after placing the string on the ot at the selected location. Another example illustrating the use of gtewxt is given below. Here itle has also been added to the image. The axis has been deleted.

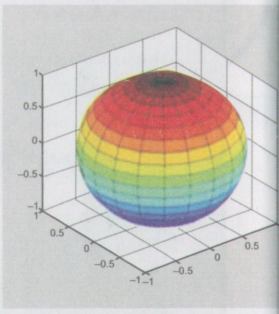
```
cd D:\pictures
a=imread('bird.jpg','jpg');
image(a)
axis off
```

```
>> title('bird');
>> gtext('bird in the nest');
```



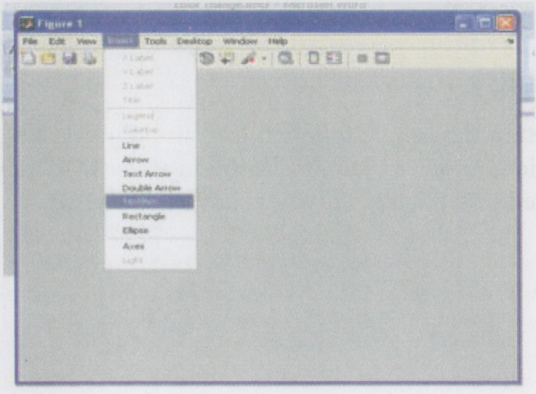
gtext is not possible for 3-dimensional figure. For example, in the sphere shown in the figu window it is not possible to use gtext to add text to the figure as it is a 3-dimensional figure.

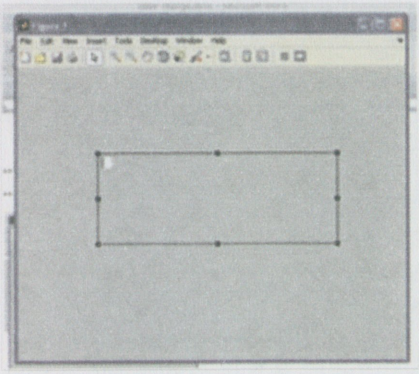
```
>> sphere
>> gtext('sphere')
??? Error using Æ gtext at 39
View must be two-dimensional.
```

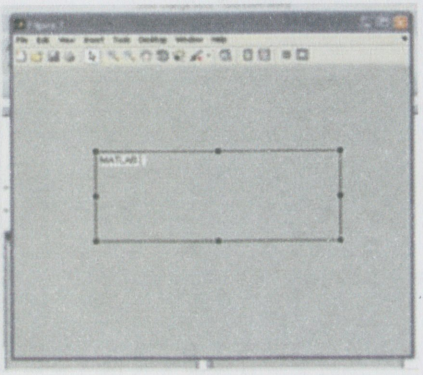


5.26.5 Adding Text to a Figure Using Tool Box

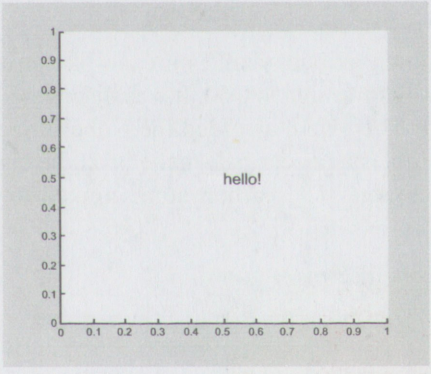
>> figure







> text(0.5,0.5,'hello!','fontsize',16,'color','r')



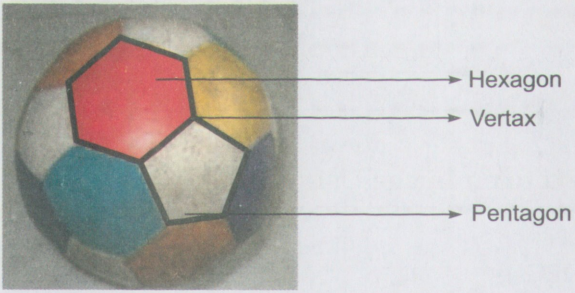
.26.6 Adding Text to an Image Using Annotation

> cd d:\Pictures
=imread('bird.JPG','JPG');
> image(a)
> annotation('textbox',...
).304571428571428 0.757142857142862 0.428571428571429 0.102380952380952],...
String',{'Wow what a wonderful day!'},...
FontWeight','bold',...
FontSize',19,...
FontName','Agency FB',...
EdgeColor',[1 0 0],...
LineWidth',2,...
BackgroundColor',[1 1 0],...
Color',[0 1 1]);



5.27 BUCKYBALL

Buckminsterfullerene also known as buckyball is a cage-like structure having 60 vertices a each vertex has a carbon atom. Its formula is C60. It was discovered in 1985 by Richard Small Harry Kroto, and Robert Curl. They were awarded the Nobel Prize in 1996 for the discovery Buckminsterfullerene. It resembles a soccer ball made of 20 hexagons and 12 pentagons. The is a bond along each polygon edge. The carbon atoms linked to three other carbon atoms covalent bonds.



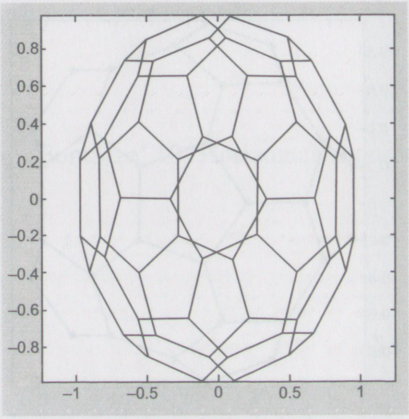
A soccer ball is similar to the buckyball made of 20 hexagons and 12 pentagons

In MATLAB, the plot of a buckyball or a geodesic dome can be created using the buc function. One can obtain the coordinates of the buckyball graph using,

```
[B,V] = bucky
```

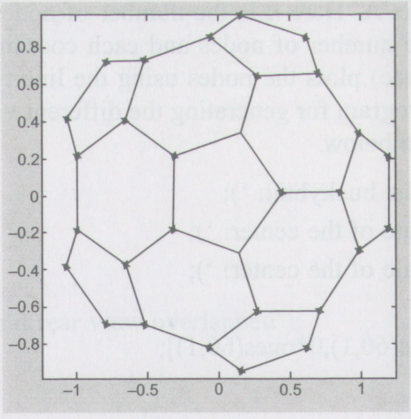
The command lines given below generate a complete buckyball.

```
[B,V] = bucky;
gplot(B,V)
axis equal
```



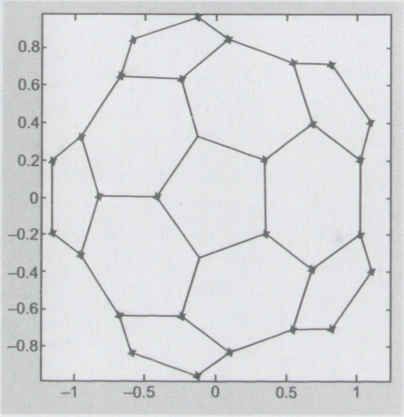
On the other hand the command lines given below will generate 30 points in the buckyball. ere we can plot half of a buckyball, placing asterisks at each node.

```
= 1:30;
B,XY] = bucky;
plot(B(k,k),XY(k,:),'-*')
xis square
```



This plot is half the buckyball as k = 1:30. In order to plot the other half of the buckyball e use k = 31:60. Here we can plot the other half of the buckyball, using red lines and placing terisks at each node.

```
= 31:60;
B,XY] = bucky;
plot(B(k,k),XY(k,:),'r-*')
xis square
```



A graph is a set of nodes with specified connections between them. gplot plots nodes a links to the representing adjacency matrix. The gplot function graphs a set of coordinates usi an adjacency matrix. The various syntaxes for gplot are:

```
gplot(A,Coordinates)
gplot(A,Coordinates,LineSpec)
```

gplot(A,Coordinates) plots a graph of the nodes defined in the coordinates vector accordi to the *n*-by-*n* adjacency matrix A. Here *n* is the number of nodes. The coordinates matrix is *n*-by-2 matrix, where *n* is the number of nodes and each coordinate pair represents one no gplot(A, Coordinates, LineSpec) plots the nodes using the line type, marker symbol and co specified by LineSpec. The program for generating the different views of the buckyball showi all the vertices has been given below.

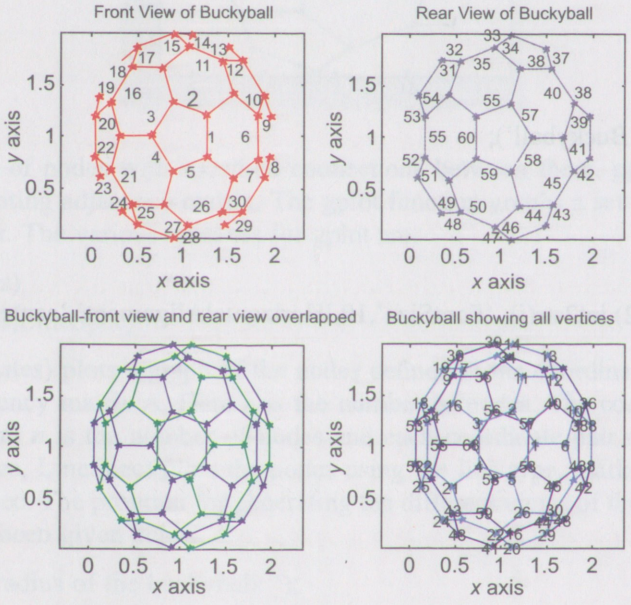
```
r=input('Enter the radius of the buckyball: ');
x=input('Enter the x coordinate of the center: ');
y=input('Enter the y coordinate of the center: ');
[B,V] = bucky;
V=r*V+ [x*ones(60,1),y*ones(60,1),0*ones(60,1)];
gplot(B,V);
xlabel('x axis');
ylabel('y axis');
H = zeros(60,60);
k = 1:30;
H(k,k) = B(k,k);
subplot(2,2,1)
gplot(B-H,V,'r-*');
axis equal
title('Front View of Buckyball');
```

```
label('x axis');
label('y axis');

or j = 1:30
   text(V(j,1),V(j,2),int2str(j), 'FontSize',10,'HorizontalAlignment','center');
nd

ubplot(2,2,2)


utput
plot(H,V,'b-*');
xis equal;
tle('Rear View of Buckyball');
label('x axis');
label('y axis');

or j = 31:60
   text(V(j,1),V(j,2),int2str(j), 'FontSize',10,'HorizontalAlignment','center');
nd

ubplot(2,2,3)

plot(B-H,V,'b-*');
old on
plot(H,V,'g-*');
label('x axis');
label('y axis');
xis equal
tle('Buckyball-fornt view and rear view overlapped');

ubplot(2,2,4)

xis equal
plot(B-H,V,'b-*');
old on
plot(H,V,'c-*');
xis equal
tle('Buckyball showing all vertices');
label('x axis');
label('y axis');
old off
xis equal
```

```
for j = 1:60
    text(V(j,1),V(j,2),int2str(j), 'FontSize',10,'HorizontalAlignment','center');
end
```

Output

```
Enter the radius of the buckyball: 1
Enter the x coordinate of the center: 1
Enter the y coordinate of the center: 1
```



6

CHAPTER

Animation, Sound and Movies Using MATLAB

"Controlling complexity is the essence of computer programming."

– Brian Kernighan

.1 ANIMATION

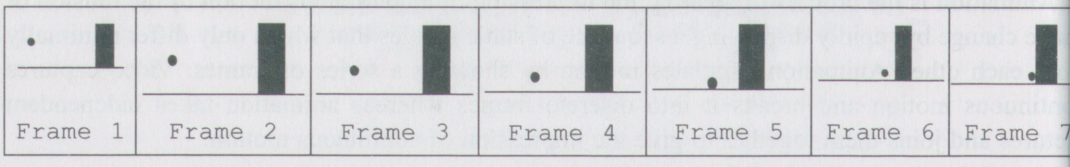
ccording to the English dictionary animation is a way of making a movie using a series of rawings, computer graphics or photographs of objects that are slightly different form one nother and that when viewed quickly one after another created the appearance of movement.

Animation is the process of creating the impression of a continuous motion or the illusion of hape change by rapidly displaying a sequence of static images that which only differ minimally rom each other. Animation simulates motion by showing a series of frames. Video captures ontinuous motion and breaks it into discrete frames whereas animation takes independent ictures and joins them together to give the impression of continuous motion.

Here 7 frames are involved in this animation.

```
dd:\pictures

=imread('m1.bmp');
mage(i);
(1)=getframe;

=imread('m2.bmp');
mage(j);
(2)=getframe;
```

```
k=imread('m3.bmp');
image(k);
f(3)=getframe;

l=imread('m4.bmp');
image(l);
f(4)=getframe;

m=imread('m5.bmp');
image(m);
f(5)=getframe;

n=imread('m6.bmp');
image(n);
f(6)=getframe;

o=imread('m7.bmp');
image(o);
f(7)=getframe;

movie(f,10,12)
```

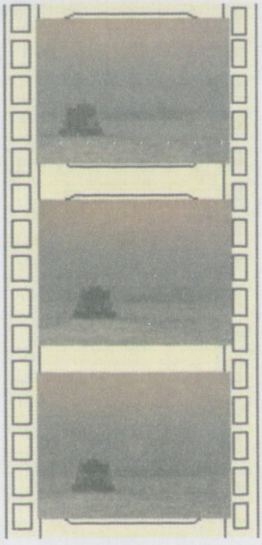


When these 7 frames are shown one after another it gives us the impression that a ball hi the wall. Each of these frames differs only minimally from the previous but the net effect motion of the ball and finally the ball hits the wall. Here the frames per second (FPS) is 12 an the number of the time the clip is played is 10.

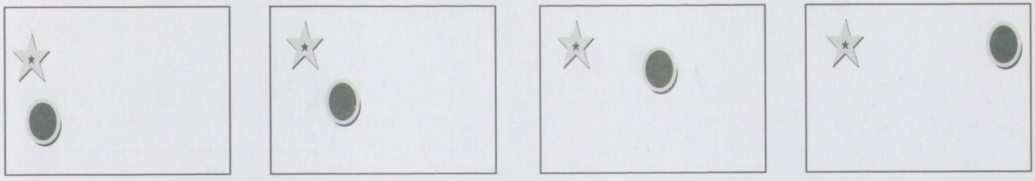
Here in the three frames given below everything in the picture changes although it is the sam location of the hills. These are shots from a movie.

Here in these frames only the red ball moves with respect to the star and everything remain unmoved. This gives the impression that the red ball is going closer to the star. These are sho from an animation.





Fog and Sunset



Three different ways by which we can animate plots in MATLAB

The mode to be selected depends on the speed in which a movie can be generated, low cost and how accurately an objective is met.

1. In the first method a number of different pictures are saved and then played back as a movie. A movie is not in real time and it is simply a playbacks frames captured previously. This method is suited where each frame is complex and is not possible to redraw them rapidly. Although one can create a movie frame in advance prior to making a movies file so that the original drawing time does not cause any hindrance during playback. These frames are then added one after another to make an entire movie.

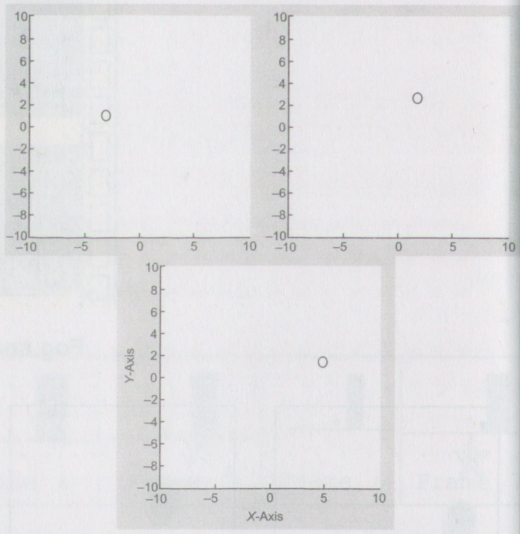
2. A figure on the screen is continuously redrawn with incremental changes in each redraw figure. In this technique figures are drawn, erased and then redrawn. This is done usin the various drawing modes supported by MATLAB graphics.
3. The *X*-data, *Y*-data and *Z*-data are redefined and the plot object properties are change which are, in turn, linked to data sources. Updating the properties is done using th refreshdata calls. In this method the plots to are updated in data and plots are redrawn.

In MATLAB animation can also be done by moving a marker from a coordinate point t another on the same figure window. Everything else in the figure window remains unmove except the markers that have been chosen to change their positions with time.

6.1.1 Animation Showing a Ball Moving Along an Elliptical Path

MATLAB Program	Output

```
t=linspace(0,20*pi,10000);
a=5;
b=2;
x=a*cos(t);
y=b*sin(t);
h=line(x(1),y(1),'marker','o',
'color','b','markersize',10,
'erase','xor');
axis('square');
axis([-10 10 -10 10])
for k=1:length(t)
   set(h,'xdata',x(k),'ydata',y(k));
   drawnow
end
xlabel('X-Axis')
ylabel('Y-Axis')
```



6.1.2 Animation of Two Balls Moving Simultaneously

MATLAB Program

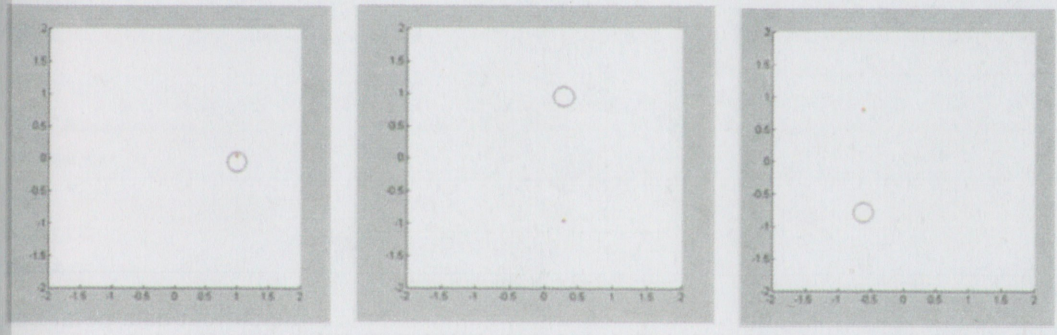
```
t=linspace(0,20*pi,10000);
x=cos(t);y=sin(t);
h1=line(x(1),y(1),'marker','o','color','b','markersize',20,'erase','xor');
h2=line(x(1),y(1),'marker','.','color','r','markersize', 10,
'erase','xor');
```

Contd

Contd.

```
axis('square');
axis([-2 2 -2 2])
for k=1:length(t)
    set(h1,'xdata',x(k),'ydata',y(k));
    set(h2,'xdata',x(length(t)-k),'ydata',y(length(t)-k));
    drawnow
end
xlabel('X-Axis')
ylabel('Y-Axis')
```

Output showing the position of the 2 beads (red and blue) at three different moments



.2 drawnow

he `drawnow` command updates the figure window and their children. The `drawnow` function is sed in animation loops to update the figure window during the execution of the function and to pdate graphical user interfaces. The `drawnow` function flushes the event queue and updates the gure window. The `drawnow` function forces objects to be plotted instantaneously. An example emonstrating the drawnow command is given below.

Program

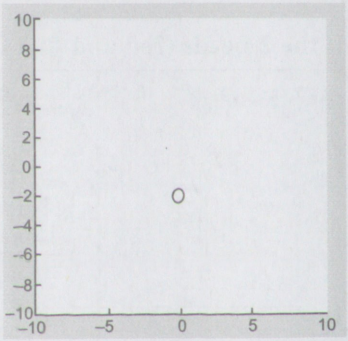
```
t=linspace(0,20*pi,10000);
a=1;
b=2;
x=a*cos(t);
y=b*sin(t);
h=line(x(1),y(1),'marker','o','color','b','markersize',10,'erase','xor');
```

Contd.

```
axis('square');
axis([-10 10 -10 10])
for k=1:length(t)
set(h,'xdata',x(k),'ydata',y(k));
 drawnow
end
xlabel('X-Axis')
ylabel('Y-Axis')
```

The drawnow command updates the figure window. The `drawnow` function flushes the event queue and updates the figure window

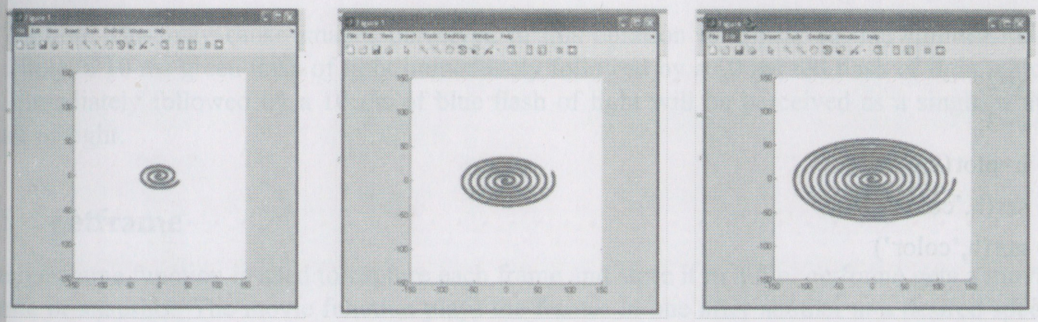
Output



As can be seen from this figure the drawnow command plots the current data point on Here a loop is used to calculate the new data point and the command drawnow plots t new position.

Another example of animation is given below. Here a bead moves in an elliptical spiral.

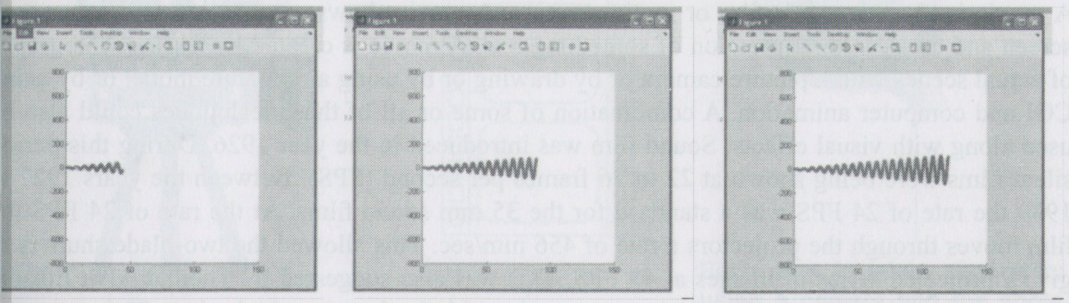
```
t=linspace(0,20*pi,10000);
a=2;
b=1;
x=a*t.*cos(t);
y=b*t.*sin(t);
h=line(x(1),y(1),'marker','o','color','r','markersize',4,'erase','none');
axis('square');
axis([-150 150 -150 150])
for k=1:length(t)
    set(h,'xdata',x(k),'ydata',y(k));
    drawnow
end
```



hree shots showing how the elliptical spiral having equation x=a*t.*cos(t) and y=b*t.*sin(t) aries as t varies

Another example which shows how the function y=x.*sin(x) looks as x changes is given elow.

```
=linspace(0,40*pi,10000);
=x.*sin(x);
=line(x(1),y(1),'marker','o','color','r','markersize',2,'erase','none');
xis('square');
xis([0 150 -800 800])
or k=1:length(x)
  set(h,'xdata',x(k),'ydata',y(k));
  drawnow
nd
```



nree shots showing how the variation of the function y=x.*sin(x) varies with the variation of x

.3 handle

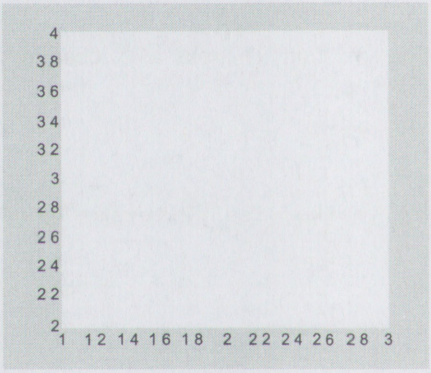
MATLAB handle is used to call functions indirectly. When MATLAB creates a graphics object, assigns an identifier to the object. This identifier is called a handle. One can use this handle access the properties of the object using the set and get functions. The example given below eates a plot and returns a handle to an object in h:

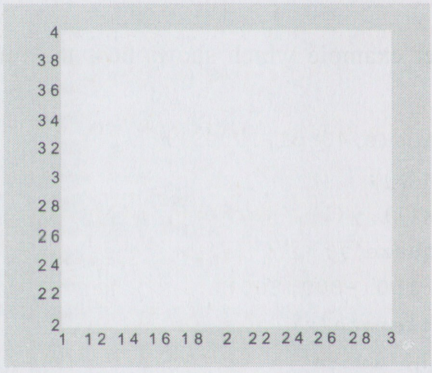
For example,

```
>> x=2;
>> y=3;
>> h=plot(x,y,'ro');
>> set(h,'color','b');
>> get(h,'color')
```

ans =

0 0 1





6.4 MAKING MOVIES WITH MATLAB

A movie is also known as film or motion picture. A movie shows a series of still images on screen and creates the impression of something real. A movie is created by taking photograph of actual scenes with a picture camera or by drawing or by using a miniature model or by usin CGI and computer animation. A combination of some or all of these techniques could also b used along with visual effects. Sound film was introduced in the year 1926. During this perio silent films were being shown at 22 to 26 frames per second (FPS). Between the years 1927 t 1930 the rate of 24 FPS was a standard for the 35 mm sound films. At the rate of 24 FPS th film moves through the projectors a rate of 456 mm/sec. This allowed the two-blade shutters t give a projected series of images at 48 FPS. This was also suggested by Thomas Alva Edisor Many modern-day 35 mm film projectors use three-blade shutters which gives 72 images pe second. Here in this case each frame is flashed on the screen three times.

The human eye and the brain can process and perceive only 10 to 12 images per secon When observing a lighted display we begin to notice a brief moment of darkness if it is 1 milliseconds (ms) or longer. Observers can recall a specific image in an unbroken series o different images when each of these images lasts for as little as 13 milliseconds. When given very short single-millisecond visual stimulus human beings report duration of between 100 m and 400 ms due to persistence of vision in the visual cortex. Persistence of vision may creat an illusion of continuity and thus allowing a sequence of still images to give an impressio

f motion. This may cause images perceived in this duration to appear as one stimulus. For xample, a 10 ms green flash of light immediately followed by a 10 ms red flash of light which immediately followed by a 10 ms of blue flash of light will be perceived as a single white ash of light.

.5 getframe

he getframe function is used to capture each frame and store it in a file. getframe gets a movie ame or snapshot. The movie function plays the frames in one after another at a desired speed r a desired number of times a movie is used to playback an array of movie frames. Information out the function getframe can be obtained using the command,

> help getframe

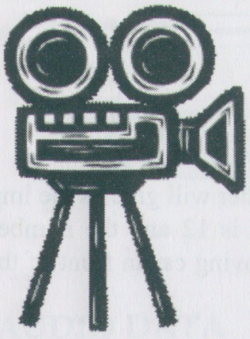
The function getframe returns a frame. The frame is one shot of the current axes or figure.

F = getframe gets a frame from the current axes and

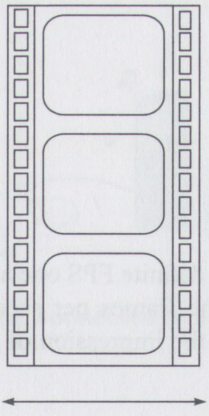
F = getframe(h) gets a frame from the figure or axes identified by handle h.

In an animation an object in the frame is made to move from one point to the other. An object the shot is moved and this gives the feeling of something moving.

In an movies the entire frame is changed one after another. Each shot is captured separately d the shots are shown one after the other.



A movie camera



35 mm



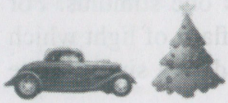
Each shot is taken separately using a camera and the shots are shown at a specific rate

The syntax of the function movies is,

ovie(M) plays the movie in matrix M once

ovie(M, n) plays the movie n times

ovie(M, n, fps) plays the movie at fps frames per second and plays it n number of time



Frame 1



Frame 2



Frame 3



Frame 4

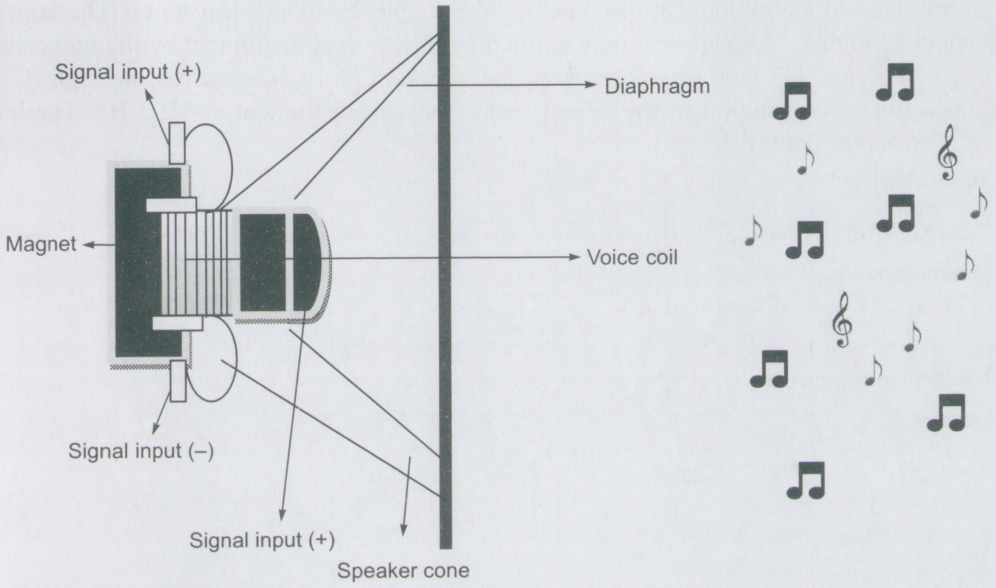
These three frames when shown one after the other give us the impression that the car moving in front of the Christmas tree.

```
cd d:\pictures

i=imread('m1.bmp');
image(i);
    f(1)=getframe;

j=imread('m2.bmp');
image(j);
f(2)=getframe;

k=imread('m3.bmp');
image(k);
    f(3)=getframe;

l=imread('m4.bmp');
image(l);
     f(4)=getframe;

movie(f,10,12)
```

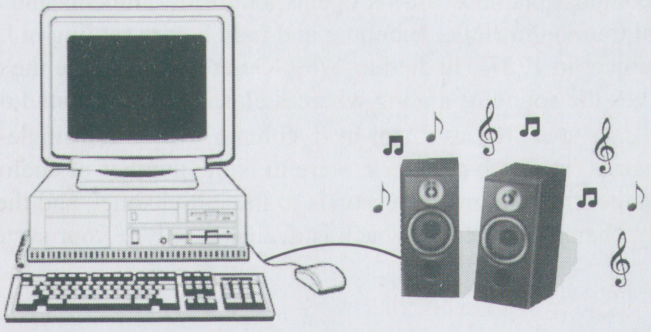
The three frames when played at a definite FPS one after the other will give us the impressi of a movie of the mountain. Here the frames per second (FPS) is 12 and the number of t time the clip is played is 10. It gives the impression of a slow moving car in front of the nice decorated Christmas tree.

6.6 SOUND

Sound is a vibration which propagates in the form of a mechanical wave of pressu and displacement, through a medium. Sound wave cannot propagate in vacuum. It needs medium to carry the sound wave from one point to another. In most cases this medium is air b it could be other materials like as liquid or solid. Sound can move through water or steel.The is an original source of the wave. This is a vibrating object capable of disturbing the medium which it propagates. The wave could be created by the vibrating cords in a musical instrume or the vocal chords of a person, the vibrating lines of a tuning fork or the vibrating diaphrag of a speaker.



A Speaker



A computer connected to a speaker

5.7 AUDIO DATA

Audio data is specified as an m-by-1 column vector for single-channel or mono audio. An n-by-2 matrix is for stereo playback. Here m is the number of audio samples. If an audio data is in m-by-2 matrix, then the first column corresponds to the left channel, and the second column corresponds to the right channel. It should be noted that stereo playback is available only if our system supports it.

sound (y) sends audio signal y to the speaker at the default sample rate of 8192 hertz. hertz or Hz is the unit of frequency in SI system and is equivalent to cycles per second. It is defined as the number of cycles per second of a periodic phenomenon. It is used to define a sine wave,

which are used in audio applications, such as the frequency of musical tones. The larger t frequency the higher is the musical note emitted by the musical instrument by the movement the string. The low 'C' note originates from the vibration of 16 Hz whereas the high 'C' no originates from the vibration of the strings in the musical instrument at 8192 Hz. The huma ear can hear up to 20000 Hz.

For example,

```
>> load handel.mat;
>> sound(y)
```

Or

```
>> load gong.mat;
sound(y);
```

Or

```
>> load chirp.mat;
sound(y, Fs);
```

Plays the file handel at the sample rate of 8192 Hz. The handel file is an excerpt fro Handel's "Hallelujah Chorus". George Frideric Handel was born in 1685. He was a Germa born British Baroque composer famous for his operas, oratorios, anthems and organ concerto Handel received critical training in Halle, Hamburg and Italy before settling in London (1712) a a naturalized British subject in 1727. He died in 1759. Gong and chirp are the other music fil in MATLAB. Gong gives the sound of a gong whereas chirp gives the sound of chirping bird

Her y is the audio data specified as an m-by-1 column vector for single-channel (mon audio, or an m-by-2 matrix for stereo playback. Here m is the number of audio samples. If y an m-by-2 matrix, then the first column corresponds to the left channel, and the second colum corresponds to the right channel. Stereo playback is available only if your system supports it.

```
>> load handel.mat;
sound(y, 2*Fs);
```

or,

```
>> load gong.mat;
sound(y, 2*Fs);
```

sound(y, Fs) plays the original audio signal y at sample rate 2 times of that of Fs. Fs is th recorder sample rate. This implies that the frequency now is 16384 Hz. The sound becomes ver shrill and could sound like noise.

Fs is the sample rate in hertz. It is a positive number between 80 and 1000000.8192 is th default value of Fs for an audio data.

```
>> load handel.mat;
>> sound(y, Fs, nBits)
```

This command uses nBits bits per sample for audio signal y. MATLAB plays the audio file with a bit depth of 16 bits per sample but the system that is being used should support this. nBits or bit depth of sample values is specified as an integer. Its value depends on the audio hardware installed. It can take values 8 (default), 16 or 24. Most platforms support bit depths of 8 bits or 16 bits. In order to set nBits value to 8,

```
>> load gong.mat;
nBits = 8;
sound(y, Fs, nBits);
```

7

CHAPTER

Numerical Methods Using MATLAB

"Google was founded by Larry Page and Sergey Brin.. The word "Google" is derived from the word "googo which stands for 10^{100}. This term was coined by 9 year old nephew, Milton Sirotta, of American Mathemat cian, Edward Kasner, in the year 1938. Kasner popularized the use of the term in his book Mathematics ar Imagination in 1940. A googol although no specific significance, but it can be used to compare with other larg quantities. A googol is approximately 70!.

$$70! = 1\ 1978571669\ 9698917960\ 7278372168\ 9098736458\ 9381425464\ 2585755536$$
$$2864628009\ 5827898453\ 1968000000\ 0000000000$$

$$1\ googol = 10^{100} = 2^{(\log_2 10)100}$$

$$As\ \log_2 10 = 3.321928094887362347870319429489 4$$

$$2^{(3.321928094887362347870319429489 4)100} = 2^{3321.928094887362347870319429489 4}$$

7.1 NUMERICAL INTEGRATION

Numerical integration is the approximate computation of an integral using numerical methods There are a large number of numerical methods available for numerical integration. Thes methods can be used to find the approximate value of a definite integral from values of th integrand when exact mathematical integration is not available. In this section the Trapezoida rule and the Simpson's rule have been explained.

7.1.1 Trapezoidal Rule

The trapezoidal rule is based on the Newton-Cotes formula. The trapezoidal rule is a techniqu for approximation of the definite integral, $\int_a^b f(x)\,dx$. This rule approximates the region under th plot of the function $f(x)$ as a trapezoid and calculates its area.

$$\int_a^b f(x)\,dx \approx (b-a)\left[\frac{f(a)+f(b)}{2}\right]$$

The trapezoidal rule approximates the integer and $(f(x))$ as an nth order polynomial.

If $f(x)$ is considered an nth order polynomial then could be written as,

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots\ldots\ldots + a_{n-1}x^{n-1} + a_nx^n$$

The integral of the function is then approximated by the integral of the nth order polynomial.

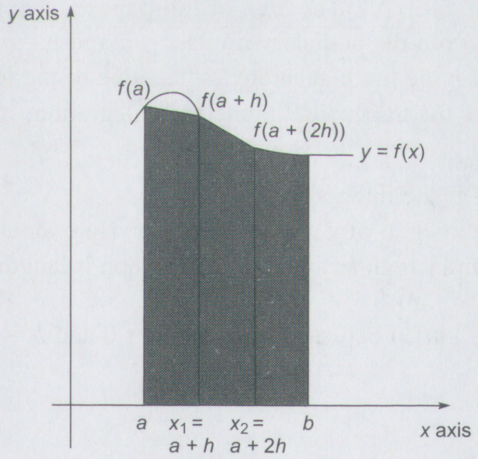
$$I = \int_a^b f(x)\,dx$$

The trapezoidal rule assumes $n = 1$. As $n = 1$ the polynomial is linear of the form and the area nder a linear polynomial of the form $f(x) = a_0 + a_1x$ is considered.

The trapezoidal rule is a method for approximating a definite integral $\int_a^b f(x)\,dx$ considering near approximations of f. The trapezoids are drawn as shown below. The bases of the trapezoids e vertical lines.

In order to use the trapezoidal rule the following steps need to be followed:

1. Divide the interval of integration (a, b) into n sub-intervals of equal width.
2. The area of the trapezoid is estimated by adding the areas of all the trapezoids. The width of each trapezoid is $\Delta x = (b - a)/n$. The larger the value of n the more accurate the value of the integral.



The Trapezoidal Rule

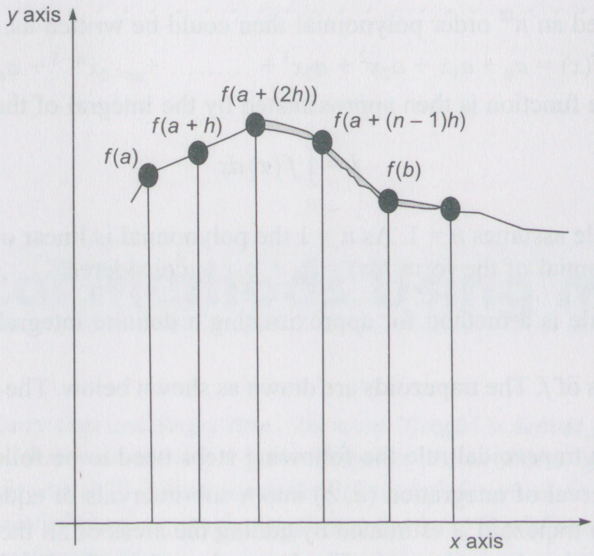
Using the trapezoidal rule we get,

$$A = \frac{h}{2}\left(f(a) + 2f(a+h) + 2f(a+2h) + \ldots + 2f(a+(n-1)h) + f(b)\right)$$

$$= \frac{(b-a)}{2n}\left(f(a) + 2f(a+h) + 2f(a+2h) + \ldots + 2f(a+(n-1)h) + f(b)\right)$$

$$= \frac{(b-a)}{n}\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + k\frac{(b-a)}{n}\right)\right)$$

$$\text{error} = \int_a^b f(x)\,dx - \frac{(b-a)}{n}\left[\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + k\frac{b-a}{n}\right)\right]$$



Area under the curve y = f(x) within the boundary limits x = a and y = b

The width of each interval is Δx. The area of the trapezoids approximately equals the a under the curve $y = f(x)$ within the boundary limits $x = a$ and $x = b$.

The higher the value of n the more accurate is the value of the integral.

The function trapz gives the trapezoidal numerical integration.

$$i = \text{trapz}(y)$$

If y is a vector trapz(y) is the integral of y.

i = trapz(x, y) gives the integral of y with respect to x. Here inputs x and y could be compl

If x is a column vector and y is an array whose dimension is length(x) then trapz(x, y) opera across this dimension.

Here the function is $y = \sin(x)$ between the limits $a = 0$ and $b = \pi$.

```
a=0;
b=pi;
n=100;
h = (b-a)/n;
ig=0;
for i = 1:n-1
ig = ig+2*sin(i*h);
end
ig = ig +sin(b);
ig = ig * h/2;
disp(ig)
```

```
1.9998
```

==========

Here the function is $y = x$ between the limits $a = 0$ and $b = 2$

```
=0;
=2;
=100;
 = (b-a)/n;
g=0;
or i = 1:n-1
g = ig+2*(i*h);
nd
g = ig +(b);
g = ig * h/2;
isp(ig)
```

.1.2 Simpson's 1/3 Rule for Numerical Integration

he Simpson's 1/3 rule for numerical integration is used to estimate the value of a definite ıtegral. It works by creating an even number of intervals and fitting a parabola in each pair of ıtervals. Simpson's rule provides the exact result for a quadratic function or parabola.

$$\int_a^b f(x)\,dx = \frac{h}{3}\left[(y_0 + y_n) + 2(y_2 + y_4 + y_6 + \ldots) + 4(y_1 + y_3 + y_5 + \ldots)\right]$$

here a and b are the lower and upper limits of integration.

The Simpson's 1/3 rule for finding out the integral of a function is very easy to put it into program and find the integral of functions. In Simpsons's 1/3 rule for multiple segment the ıterval can be divided into n segments where n is even.

Therefore,

$$h = \frac{l0 - \ln}{n}$$

here ln is the upper limit and $l0$ the lower limit of integration. Therefore, we need to find,

$$\text{Integral} = \int_a^b f(x)\,dx$$

Simpson's 1/3rd when applied,

$$\text{ıtegral} = \int_a^b f(x)\,dx = \int_{l0}^{l2} f(x)\,dx + \int_{l2}^{l4} f(x)\,dx + \int_{l4}^{l6} f(x)\,dx + \ldots\ldots\ldots\ldots + \int_{l(n-2)}^{\ln} f(x)\,dx$$

$$= (l2 - l0)\left[\frac{f(x0) + 4f(x1) + f(x2)}{6}\right] + (l4 - l2)\left[\frac{f(x2) + 4f(x3) + f(x4)}{6}\right] + \ldots\ldots\ldots\ldots$$

$$+ (\ln - l(n-2))\left[\frac{f(x(n-2) + 4f(x(n-1) + f(xn)}{6}\right]$$

Now $l2 - l0 = 2h$

Therefore,

$$\text{Integral} = \left(\frac{h}{3}\right)[f(x0)] + 4(f(x2)) + f(x3) + f(x5) + \ldots + f(x(n-1))) + 2(f(x2)) + f(x4) + f(x$$

$$+ \ldots + f(x(n-2)))\, f(xn)$$

Therefore the integral is,

$$\text{Integral} = \left(\frac{h}{3}\right)\left[f(x0) + 4\sum_{1}^{n-1} f(xi))) + 2\sum_{2}^{n-2} f(xi) + f(xn) \right]$$

Comparison Simpson's 1/3 Rule and Trapezoidal Rule	**Output**	**Actual Integration**

`a=input('enter a:')`
`b=input('enter b:')`
`n=input('enter number of intervals:')`
`h=(b-a)/n;`
 	
`sum1=0;`
`for(i=1:2:n - 1)`
`xval= a + i*h;`
`sum1=sum1+xval*xval;`
`end`
 	
`sum2=0;`
`for(j=2:2:n-1)`
`xval= a + j*h;`
`sum2=sum2+xval*xval;`
`end`
 	
`I=(h/3)*[(a*a + b*b)+4*sum1+2*sum2];`
`disp('The integral is equal to:')`
`I`
	>> simp
	enter a:0
	a =
	0
	enter b:2
	b =
	2
	enter number of intervals:100
	n =
	100
	The integral is equal to:
	I =
	2.6667
		$\int_0^2 x^2 dx = \frac{x^3}{3}\Big\|_0^2 = 8/3 - 0 = 2.6666\ldots66$
		It has been found that in general, t Simpson's 1/3rd rule can be used compute an integral more efficient and accurately than the trapezoid rule. This is because it is a highe order polynomial and is more like to provide a better fit for the functio

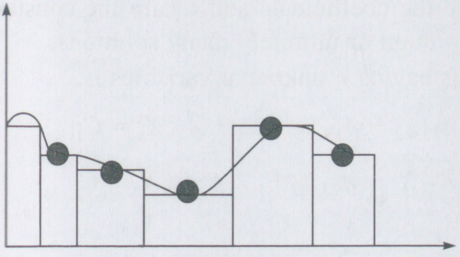
ontd.

```
a=input('enter a:')
b=input('enter b:')
n=input('enter number of
  intervals:')
h=(b-a)/n;
t=0;
sum=0.0;
for i=(i+1):n-1
    x=a+i*h;
sum=sum+2*(x*x);
i=i+1;
end
t=(h/2)*(sum+(a*a)+(b*b));
```

```
>> int
enter a:0
a =
     0
enter b:2
b =
     2
enter number of
intervals:100
n =
   100
t =
   2.6668
```

A large class of quadrature rules can be derived by constructing interpolating functions which e easy to integrate. The interpolating functions are typically polynomials. The simplest method to let the interpolating function be a constant function. This is a polynomial of degree 0. This nction passes through the point $\left(\left(\frac{a+b}{2}\right), f\left(\frac{a+b}{2}\right)\right)$. This is called the midpoint rule or the ctangular rule.

$$\int_a^b f(x)\,dx \approx (b-a) f\left(\frac{a+b}{2}\right)$$



Rectangular Rule

MATLAB Program	Output

x1=input('enter the value of lower limit:')
x2=input('enter the value of upper limit:')
n=input('enter the value of n:')
h=(x2-x1)/n;
x=linspace(x1,x2,(n+1));
f=(x.*x);
i=1;
sum=0;
for i=2:n
sum=sum+(2.*f(i));
i=i+1;
end
int=(h/2)*(sum+f(1)+f(n+1));
disp('the value of the integral is:')
int
	enter the value of lower limit:0
	x1 =
	0
	enter the value of upper limit:2
	x2 =
	2
	enter the value of n:100
	n =
	100
	the value of the integral is:
	int =
	2.6668

7.2 SYSTEM OF LINEAR EQUATIONS

A system of linear equations occurs when two or more equations govern a process together. linear equation is an equation where the terms are either a constant or constant multiplied b the first power of the variables like x, y, z etc. A system of linear equations is a collection linear equations having the same variables. A solution of a linear system of equations satisfi each equation. In a system of linear equations a set of linear equations act simultaneously. finding the solution to the system of linear equations all the equations have to be considere simultaneously and the solution to the system of linear equations satisfy all the equatio simultaneously.

A system of m linear equations each having n unknown variables (X_1, X_2, X_3 ....., X_n) considered here. Here A_{ij} are the coefficients and C_i are the constant terms. This system ma have no solution, a unique solution or infinitely many solutions.

The system of m equations having n unknown variables is,

$$A_{11}X_1 + A_{12}X_2 + A_{13}X_3 + \ldots + A_{1n}X_n = C_1 \quad \ldots(1$$

$$A_{21}X_1 + A_{22}X_2 + A_{23}X_3 + \ldots + A_{2n}X_n = C_2 \quad \ldots(2$$

$$\ldots$$

$$A_{i1}X_1 + A_{i2}X_2 + A_{i3}X_3 + \ldots + A_{ij}X_j + \ldots + A_{in}X_n = C_i \quad \ldots(3$$

$$\ldots$$

$$A_{m1}X_1 + A_{m2}X_2 + A_{m3}X_3 + \ldots + A_{mn}X_n = C_m \quad \ldots(4$$

The coefficient matrix is,

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ A_{i1} & A_{i2} & A_{i3} & \dots & A_{in} \\ \dots & \dots & \dots & \dots & \dots \\ A_{m1} & A_{m2} & A_{m3} & \dots & A_{mn} \end{pmatrix}$$

The matrix of the constant terms is,

$$C = \begin{pmatrix} C_1 \\ C_2 \\ . \\ . \\ C_i \\ . \\ . \\ C_m \end{pmatrix}$$

The coefficient of the n unknown variables is,

$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ . \\ . \\ . \\ X_n \end{pmatrix}$$

Let us consider a system of linear equations,

$$2x + 3y = 5 \quad \dots(1)$$

$$x + y = 2 \quad \dots(2)$$

If X is the determinant of the unknown variables and C is the determinant of the constant ms then,

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 1 \end{bmatrix} C = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

Here X which is the matrix of the unknown variables needs to be determined. If we multiply h side of the equation by A^{-1}, we get

$$A^{-1}(A\,X) = A^{-1}(C)$$

or

$$(A^{-1}A)(X) = A^{-1}(C)$$

or

$$IX = A^{-1}C \qquad (AA^{-1} = I, \text{ where } I \text{ is the identity matrix})$$

An identity matrix is a matrix which has ones on the main diagonal of the matrix and zer elsewhere. A 3 by 3 identity matrix is therefore,

```
>> eye(2)

ans =

     1     0
     0     1
```

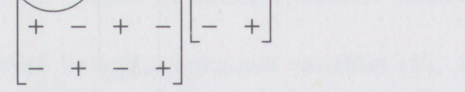
$X = A^{-1}C$ ($IX = X$, as any matrix multiplied by the identity matrix will give the same matrix

Finding the inverse of the matrix A and multiplying the matrix with B we can find the soluti matrix, C, of the system of linear equations. Here we have to find A^{-1} in order to find X. The are various ways by which the inverse of a matrix can be found. One of the methods is t adjoint method.

The inverse of a matrix is given by,

$$A^{-1} = \frac{\text{Adj}(A)}{\det(A)} \quad \text{or} \quad A^{-1} = \frac{1}{\det(A)} \qquad (\text{cofactor matrix of } A)^T$$

adj (A) is short the for adjoint of A. The inverse of the matrix is found by using the adjoint matr method. We find the adjoint matrix by replacing each element in the matrix with its cofactor a applying a + or – sign as shown



The cofactor of an element in a matrix is the value obtained by evaluating the determina formed by the elements not in that particular row or column and then finding the transpose the resulting matrix.

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} \frac{5}{2} \end{bmatrix}$$

Determinant of A or Det (A) = 2 – 3 = –1

Therefore, the cofactor matrix of $A = \begin{bmatrix} 1 & -1 \\ -3 & 2 \end{bmatrix}$

The adjoint of A of Adj (A) is the transpose of the cofactor matrix.

$$\text{Adj}(A) = \begin{bmatrix} 1 & -3 \\ -1 & 2 \end{bmatrix}$$

$$A^{-1} = \frac{\text{Adj}(A)}{\text{Det}(A)} = \begin{bmatrix} 1 & -3 \\ -1 & 2 \end{bmatrix} / -1 = \begin{bmatrix} -1 & 3 \\ 1 & -2 \end{bmatrix}$$

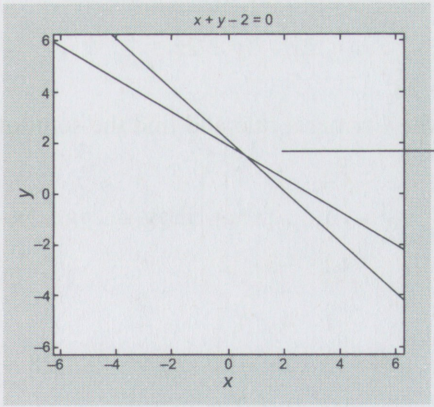
$$A^{-1}C = \begin{bmatrix} -1 & 3 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Therefore the solution to the above system of equations is $X = 1$ and $Y = 1$.

Using '\' operator the solution to the system of linear equations can be found. We can now verify this solution by directly finding out $A^{-1}C$. As $A^{-1}C$ is the same as $A\backslash C$ we can write $X = A^{-1}C$ instead of $X = A\backslash C$ to get the solution.

```
-> A=[2 3;1 1];
-> C=[5;2];
-> A\C
ns =
   1
   1
```

Program	Output

`u=inline('2*x+3*y-5','x','y')`
`u =`
`Inline function:`
`u(x,y) = 2*x+3*y-5`
`>> v=inline('x+y-2','x','y')`
`v =`
`Inline function:`
`v(x,y) = x+y-2`
`>> ezplot(u)`
`hold on`
`ezplot(v)`
	 (1, 1) is the solution to the system of equations

A general solution of a system of linear equations having two unknowns is of the form

$$a\,X + b\,Y = c$$

$$d\,X + e\,Y = f$$

Cramer's rule can also be used for finding the solution of this set of equation. D is th determinant given by,

$$D = \begin{vmatrix} a & b \\ d & e \end{vmatrix}$$

$$D = ae - bd$$

$$X = \begin{vmatrix} c & b \\ f & e \end{vmatrix} / D$$

$$X = (ce - fb)/D$$

and

$$Y = \begin{vmatrix} a & c \\ d & f \end{vmatrix}$$

$$Y = (af - dc)/D$$

The point of intersection of the two lines is the solution to this system of equations. If the tw equations do not converge at a point then there is no unique solution to this system of equation

7.2.1 System of Equations with Three Unknowns

Thus we can find the solution to the 2 by 2 system of equations. Let us consider another exampl of a 2 by 2 system of equations.

$$x + 2y + z = 4$$

$$3x - y - z = 1$$

$$2x - 3y + 2z = 1$$

The determinant $D = -28$

Let us now try to use the Cramer's rule and find the solution.Using Cramer's rule we get

$$x = \begin{bmatrix} 4 & 2 & 1 \\ 1 & -1 & -1 \\ 1 & -3 & 2 \end{bmatrix} / -28 = -28/-28 = 1$$

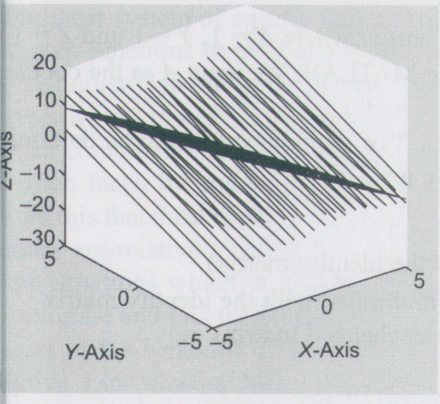
$$y = \begin{bmatrix} 1 & 4 & 1 \\ 3 & 1 & -1 \\ 2 & 1 & 2 \end{bmatrix} / -28 = -28/-28 = 1$$

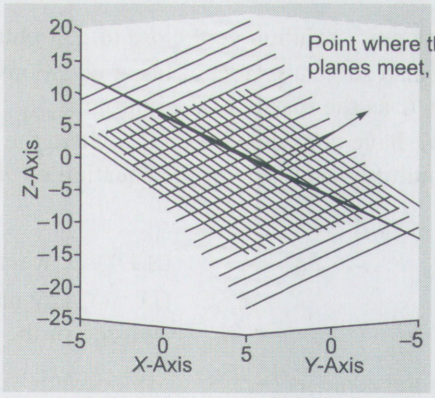
$$z = \begin{bmatrix} 1 & 2 & 4 \\ 3 & -1 & 1 \\ 2 & -3 & 1 \end{bmatrix} / -28 = -28/-28 = 1$$

This problem involves three unknown variables and hence is a three-dimensional problem. ch of the above equation represents a plane. The point where the three planes intersect is the ution to this system of equations. A MATLAB code has been written to plot the three planes hin a specific range of x, y and z.

The three planes when plotted using MATLAB shows how they intersect.

```
linspace(-5,5,20);
linspace(-5,5,20);
,v]=meshgrid(x,y);
=-1-3*u-v;
ot3(u,v,z1,'r')
ld on
=u +2*v-2;
ot3(u,v,z2,'g')
ld on
=0.5-u+1.5*v;
ot3(u,v,z3,'b')
abel('X-Axis')
abel('Y-Axis')
abel('Z-Axis')
```





The three planes $x + 2y + z = 4$, $3x - y - z = 1$ and $2x - 3y + 2z = 1$ intersect at a point P, $0 < 5$, $0 < y < 5$ and $0 < z < 20$, which is the solution to the system of equations.

The two matrices A and C are,

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 3 & -1 & -1 \\ 2 & -3 & 2 \end{bmatrix}$$

$$C = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}$$

The determinant of the coefficient matrix A is,

Here $\det(A)$ or $D = -28$

Cofactor matrix of $A = -5 - 8 - 7$

$$-70 \quad 7$$
$$14 \quad -7$$

$$\text{Adj}(A) = A' = \begin{bmatrix} -5 & -7 & 1 \\ -8 & 0 & 4 \\ -7 & 4 & -7 \end{bmatrix}$$

$$A^{-1} = \frac{\text{Adj}(A)}{\text{Det}(A)} = \begin{bmatrix} -5 & -7 & 1 \\ -8 & 0 & 4 \\ -7 & 4 & -7 \end{bmatrix} / -28 = \begin{bmatrix} 0.1786 & 0.2500 & -0.0357 \\ 0.2857 & 0 & -0.1429 \\ 0.2500 & -0.1429 & 0.2500 \end{bmatrix}$$

In matrix notation the set of equations we can written as,

$$AX = C,$$

$$X = A^{-1}C = \begin{bmatrix} 0.1786 & 0.2500 & -0.0357 \\ 0.2857 & 0 & -0.1429 \\ 0.2500 & -0.1429 & 0.2500 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.9286 \\ 1 \\ 1.1071 \end{bmatrix}$$

Thus, we get a solution very close to that obtained earlier where $X = 1$, $Y = 1$ and $Z = 1$.

We can also solve a system of linear equations using MATLAB. We input A as the coeffic matrix and C as the matrix of the constants.

Here, we have considered $AX = C$, where the matrix X of n unknowns has to be determi So if we multiply each side of the equation by A^{-1}, we get,

$$A^{-1}AX = A^{-1}C$$
$$IX = A^{-1}C \quad (AA^{-1} = I, \text{ where } I \text{ is the identity matrix})$$
$$X = A^{-1}C \quad (IX = X, \text{ any matrix multiplied with the identity matrix will return the same unchanged matrix}).$$

$A^{-1}C$ is the same as the backslash operator'\'. So

$$X = A^{-1}C$$

In MATLAB $A^{-1}C$ can be determined using '\'.

So

$$X = A^{-1}C = A\backslash C.$$

Now let us consider the same set of linear equations,

```
>> A=[1 2 1;3 -1 -1;2 -3 2];
>> C=[4;1;1];
>> X=A\C
```

=

1.0000
1.0000
1.0000

3 SYSTEM OF NON-LINEAR EQUATIONS

uations whose graphs do not form a straight line or is not linear is called a nonlinear equation. a nonlinear equation, the variables are either of degree greater than 1 or less than 1, but it is ver equal to 1. One type of nonlinear equation is a single nonlinear equation having a single riable and the solution is scalar x for which $f(x) = 0$. The other type of nonlinear equation is a stem of n coupled nonlinear equations having n variables, where the solution is a vector x for ich all components of f or all the equations in the system are zero simultaneously, $f(x) = 0$.

A few examples of nonlinear system of equations are:

$= x^2$ ...(1)	$x^2 + y^2 = 2$ ...(1)	$e^x + e^{-x} = 2$ ...(1)	$\ln(1 + x) = 2$ ...(1)
$= y^2$ ...(2)	$x^2/2 + y^2/3 = 1$ ...(1)	$e^{2x} + 1 = 2$ ...(2)	$e^x - 1 = \ln n(x)$ ...(2)

The output of a nonlinear system of equations is not directly proportional to the input. A nonlinear stem compared to a linear system, is a system which does not satisfy the superposition principle. nonlinear system of equations is a set of simultaneous equations in which the unknowns or the known functions in the case of differential equations appear as variables of a polynomial of gree higher than one or in the argument of a function which is not a polynomial of degree one. other words, in a nonlinear system of equations, the equations to be solved cannot be written as inear combination of the unknown variables or functions that appear in it. Nonlinear problems e often faced by engineers, scientists, mathematicians and largely by physicists. The reason hind this that most systems that we come across are inherently nonlinear in nature. As nonlinear stems are relatively difficult to solve, this is why the nonlinear systems are approximated by ear equations which is also known as linearization. Although this technique works but it is t accurate and this technique only works for a certain range of values. It is known that certain pects of the behavior of a nonlinear system appear to be chaotic, unpredictable and counter-uitive. Linearization hides these phenomena such as chaos and singularities. Although chaotic havior may resemble random behavior, it is not absolutely random.

3.1 Solving a System of Nonlinear Equations

nsider a system of n nonlinear equations having n unknowns.

$$f_1(x_1, x_2, \ldots, x_n) = 0$$
$$f_2(x_1, x_2, \ldots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, x_2, \ldots, x_n) = 0$$

It is possible to write the system in a single expression using the notation,

$$f(x) = 0$$

where the vector x contains the independent variables,

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_n \end{bmatrix}$$

and the vector f contains the

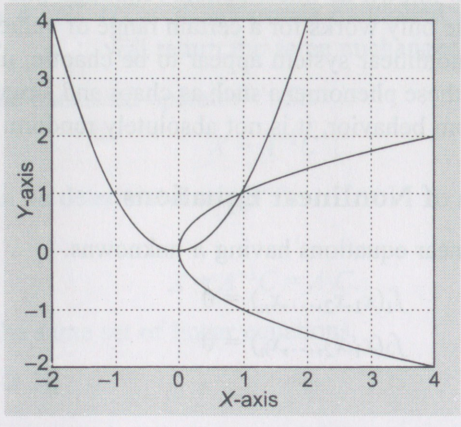
$$f(x) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \\ f_n(x_1, x_2, \ldots, x_n) \end{bmatrix}$$

For example, the set of equations,

$$y = x^2$$
$$x = y^2$$

is a system of nonlinear equations having two variables. The solution to these equations is intersection of the two curves represented by the equations.

Program	Output	Explanation

`x=-2:0.1:2;`
`y=x.*x;`
`plot(x,y)`
`hold on`
`y=-2:0.1:2;`
`x=y.*y;`
`plot(x,y)`
`grid on`
`xlabel('x-axis')`
`ylabel('y-axis')`
		The two curves $y = x^2$ a $x = y^2$ intersect at points (0) and (1, 1). Therefo this is the solution this system of nonline equations.

.4 SOLVING SYSTEM OF EQUATIONS USING MATLAB FUNCTION fsolve

solve is a function used for solving a system of nonlinear equations of several variables. Let us onsider the system of nonlinear equations given above.

$$y = x^2 \quad \ldots(1)$$

$$x = y^2 \quad \ldots(2)$$

The program for solving the above system of equations is,

```
unction nonlinear
i = [2 2];
ptions=optimset('Display','iter');
 = fsolve(@funct,xi,options)
nd
unction expr = funct(x)
xpr = [x(2)-x(1)*x(1);
     x(1)- x(2)*x(2)];
nd
```

Output

Norm of Iteration	First-order Func-count	$f(x)$	Trust-region step	optimality	radius
0	3	8	6	1	
1	6	0.395062	0.942809	0.741	1
2	9	0.0101136	0.377124	0.0806	2.36
3	12	3.09991e-005	0.088735	0.00397	2.36
4	15	4.65692e-010	0.00552436	1.53e-005	2.36
5	18	1.08615e-019	2.15792e-005	2.33e-010	2.36

Equation solved.
fsolve completed because the vector of function values is near zero as measured by the default value of the function tolerance, and the problem appears regular as measured by the gradient.
<stopping criteria details>

x =

1.0000 1.0000

Let us consider another problem.

```
unction nlinear
 x0 = [-2 -2];
```

```
options = optimset('Display','iter');
x = fsolve(@fn,x0,options)
end

function F = fn(x)
F = [x(2) - exp(x(1));
     x(2) - x(1)*x(1)];
end
```

Output

Iteration	Func-count	$f(x)$	Norm of step	First-order optimality	Trust-region radius
0	3	40.5597	1	23.7	1
1	6	11.8618	1	5.17	1
2	9	4.1601	1	2.82	1
3	12	0.449256	0.529848	0.885	1
4	15	0.00210967	0.0262778	0.0577	2.5
5	18	7.89051e-008	0.000161963	0.000351	2.5
6	21	1.46237e-016	0.000161963	1.51e-008	2.5

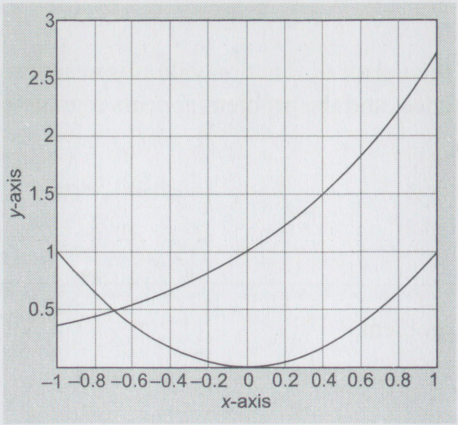
Equation solved.
fsolve completed because the vector of function values is near zero as measured by the default value of the function tolerance, and the problem appears regular as measured by the gradient.

<stopping criteria details>

x =

−0.7035 0.4949

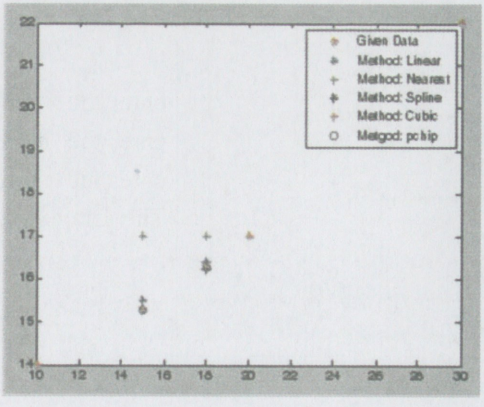
Program	Output	Explanation

`x=-1:0.1:1;`
`y=exp(x);`
`plot(x,y)`
`hold on`
`y=x.*x;`
`plot(x,y)`
`grid on`
`xlabel('x-axis')`
`ylabel('y-axis')`
		As we can see the solution is (−0.7 0.5) which is also the solution found using the fsolve function.

7.5 INTERPOLATION

The function interp1 can be used to interpolate between a given set of data points at one or more than one x values. The default interpolation method is linear interpolation. The other interpolation methods available are nearest neighbour and spline interpolation. Another method of interpolation is to fit a curve through the given data points and then find the values of the curve at the desired points.

```
x = [10 20 30];
y = [14 17 22];
plot(x,y,'r*')
hold on
xdash = [15 18]
ydashln = interp1(x,y,xdash)
ydashn = interp1(x,y,xdash,'nearest')
ydashs = interp1(x,y,xdash,'spline')
ydashc = interp1(x,y,xdash,'cubic')
ydashp = interp1(x,y,xdash,'pchip')
plot(xdash,ydashln,'b*')
hold on
plot(xdash,ydashn,'g+')
hold on
plot(xdash,ydashs,'k*')
hold on
plot(xdash,ydashc,'r+')
hold on
plot(xdash,ydashp,'bo')
legend('Given Data','Method: Linear','Method: Nearest','Method:
Spline','Method: Cubic','Metgod: pchip')
```



Very often empirical data are obtained experimentally and the data points do not conform to any mathematical relationship. When other points are to be estimated from this set of data points then interpolation is the method by which one can find values of such data points.

The MATLAB function 'interp1' in its default mode linearly interpolates.

A more accurate result is possible by considering cubic spline interpolation. This method connects the given data points by a 3rd order polynomials in each interval. The transition between adjacent splines, which are cubic polynomials, is smooth. This is because their lower order derivatives on either side of the data points are equal.

7.5.1 Lagrange Interpolation

A polynomial of degree n can be fit through $(n + 1)$ points. The Lagrange interpolation formula is used d for interpolation an nth degree polynomial $f(x)$ through $(n + 1)$ given data points. Lagrange interpolation formula gives us an nth degree polynomial $f(x)$ approximated through the $(n + 1)$ data points. The Lagrange interpolation formula is,

$$f(x) = \frac{(x - x_1)\ldots(x - x_n)}{(x_0 - x_1)\ldots(x_0 - x_n)} f_0 + \ldots + \frac{(x - x_0)\ldots(x - x_{n-1})}{(x_n - x_0)\ldots(x_n - x_{n-1})} f_n$$

$$= \sum_{i=0}^{n} \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_i)}{(x_i - x_j)} f_i$$

Program	Output

`m=input('enter the number of elements:');`
`for i=1:m`
`k=input('enter the number:');`
`x(i)=k;`
`end`
`disp('The array is:')`
`x`
`for j=1:m`
`k=input('enter the number:');`
`y(j)=k;`
`end`
`disp('The array is:')`
`y`
`n=length(x);`
`xx=input('enter the point where we have to find y:');`
`c=0;`
	enter the number of elements:3
	enter the number:1
	enter the number:2
	enter the number:3
	The array is:
	x =
	1 2 3
	enter the number:1
	enter the number:2
	enter the number:3
	The array is:
	y =
	1 2 3

Contd

Contd.

```
for k=1:n
    a=1;
    b=1;
for l=1:n
if (k~=l)
        a=a*(xx-x(l));
        b=b*(x(k)-x(l));
end
end
        c=c+(a/b)*y(k);
end
disp('The interpolated value of y at
this value of x is:')
c
```

```
enter the point where we have to find y:5
The interpolated value of y at this value
of x is:
c =
     5
```

5.2 Two-Dimensional Interpolation

Two-dimensional interpolation is very similar to the one-dimensional interpolation but in the case of two-dimensional interpolation there are two independent variables and one dependent variable.

The MATLAB function 'interp2' is used for two-dimensional interpolation. It is a two-way linear interpolation taking place in both directions.

The Table below shows the measurements of z at various values of x and y.

For an estimate of z when x and y is given, only four data points are needed.

$x \rightarrow$ $y \downarrow$	0.2	0.4	0.6	0.8	1
2					
4					
6		32	36		
8		41	44		
10					

```
=f(x,y)
1=0.4;
2=0.6;
1=6;
2=8;
```

Contd.

Contd.

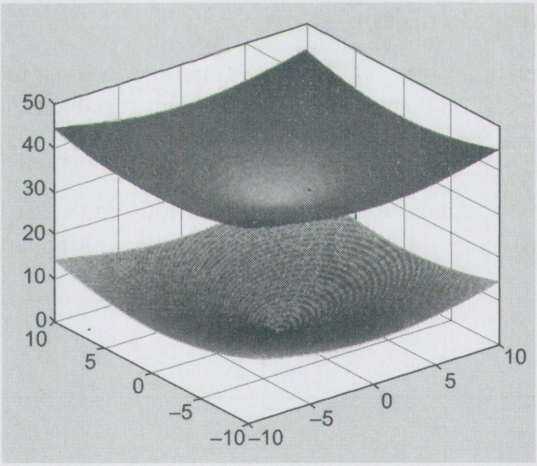
```
z11=32; z12=64; z21=41; z22=44;
x=[x1 x2];
y=[y1 y2];
z=[z11 z12; z21 z22];
estz=interp2(x,y,z,0.5,6.9)
```

estz =

45.5250

Here the function $z^2 = x^2 + y^2$ has been interpolated over a finer grid.

```
[X,Y] = meshgrid(-10:.2:10);
Z = sqrt(X.*X+Y.*Y);
[XX,YY] = meshgrid(-10:.02:10);
ZZ = interp2(X,Y,Z,XX,YY);
mesh(X,Y,Z), hold, mesh(XX,YY,ZZ+30)
```

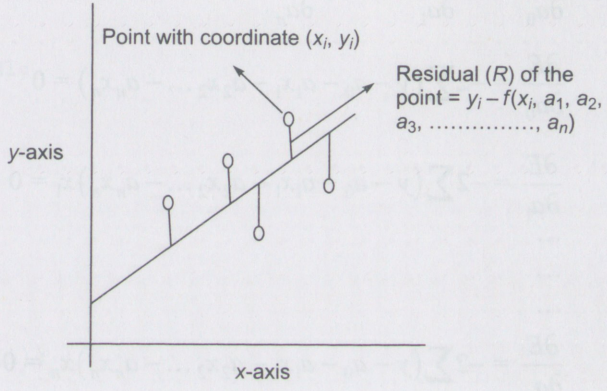


7.6 STRAIGHT LINE FIT USING LEAST SQUARE METHOD

The mathematical procedure for finding the best-fit curve for a given set of points is done minimizing the sum of the squares of the offsets or the residuals of the points from the best curve. The sum of the squares of the offsets is used instead of the absolute values of the off because this allows us to treat the residuals as a continuous and differentiable quantity. practice, the vertical offsets from a line are almost always minimized instead of the perpendicu offsets. The linear least squares fitting technique is the simplest and most commonly applied fo of linear regression and provides a solution to the problem of finding the best fitting straight l through a set of points. The formulas for linear least squares fitting were independently deriv by Gauss and Legendre.

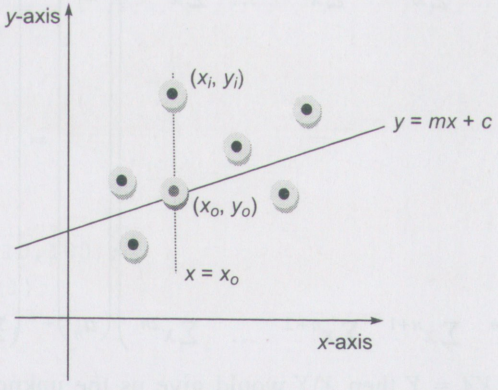
The vertical least squares fitting proceeds by finding the sum of the squares the vertical deviations E^2 of a set of n data points.

$$E^2 = \Sigma[y_i - f(x_i, a_0, a_2, a_3, \ldots, a_n)]^2$$



The equation of the line passing through (x_o, y_o) and parallel to the y-axis is $x = x_o$.
Therefore the distance of the point x_i, y_i form x_o, y_o is,

$$E^2 = (x_i - x_o)^2 + (y_i - y_o)^2$$



Since $x_i = x_o$ as the point (x_i, y_i) lies on the line $x = x_o$,
So,

$$E^2 = (y_i - y_o)^2$$

In order to minimize E^2,
Here $a_0, a_2, a_3, \ldots, a_n$ are the n unknowns to be found if an nth degree polynomial has to be o be fit through the n points.

$$\frac{\partial(E^2)}{\partial a_i} = 0 \text{ for } \quad i = 1, \ldots, n.$$

So as E^2 is minimized with respect to $a_0, a_2, a_3, \ldots, a_n$,

$$\frac{\partial E}{\partial a_0} = 0, \frac{\partial E}{\partial a_1} = 0 \ldots, \frac{\partial E}{\partial a_n} = 0$$

$$\frac{\partial E}{\partial a_0} = -2\sum\left(y - a_0 - a_1x_1 - a_2x_2 \ldots - a_nx_n\right) = 0$$

$$\frac{\partial E}{\partial a_1} = -2\sum\left(y - a_0 - a_1x_1 - a_2x_2 \ldots - a_nx_n\right)x_1 = 0$$

$$\ldots$$

$$\ldots$$

$$\ldots$$

$$\frac{\partial E}{\partial a_n} = -2\sum\left(y - a_0 - a_1x_1 - a_2x_2 \ldots - a_nx_n\right)x_n = 0$$

If this is considered a system of n equations whose variables are a_0, a_1, $a_2 \ldots a_n$. Then it could be written

$$\begin{pmatrix} \Sigma 1 & \Sigma x^1 & \Sigma x^2 & \ldots & \Sigma x^n \\ \Sigma x^1 & \Sigma x^2 & \Sigma x^3 & \ldots & \Sigma x^{n+1} \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ \Sigma x^n & \Sigma x^{n+1} & \Sigma x^{n+2} & \ldots & \Sigma x^{2n} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \\ \\ \\ \\ \\ \\ a_n \end{pmatrix} = \begin{pmatrix} \Sigma y \\ \Sigma yx \\ \\ \\ \\ \\ \\ \\ \Sigma yx^n \end{pmatrix}$$

If this is of the form $XA = Y$ then $X\backslash Y$ would give us the unknown matrix A. Let us no consider this for a line fit through a given set of data. A straight line is a polynomial of the for $y = a_0 + a_1x_1$. Here we have only two variables a_0 and a_1. These two variables can be determine form the equation,

$$\begin{pmatrix} \Sigma 1 & \Sigma x_1 \\ \Sigma x_1 & \Sigma x_1^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \Sigma y \\ \Sigma yx_1 \end{pmatrix}$$

A MATLAB program to find the solution to the above equation is given below.

```
n=input('enter the  number of data sets:')
for i=1:n
x(i)=input('enter the  x value of data:')
y(i)=input('enter the  y value of data:')
end
```

Conte

Contd.

```
M1=0;
for i=1:n
   M1=(x(i)^2)+M1;
end
M2=0;
for i=1:n
   M2=M2+x(i);
end
M3=0;
for i=1:n
   M3=M3+x(i)*y(i);
end
M4=0;
for i=1:n
   M4=M4+y(i);
end
M5=0;
for i=1:n
   M5=M5+1;
end
M=[M5 M2;M2 M1];
L=[M4;M3];
P=M\L
xfit = linspace(-10,10,100);
yfit = P(2)*xfit + P(1);
plot(x,y,'o',xfit,yfit,'-');
xlabel('X-Axis');
ylabel('Y-Axis');
title('Line Fit Using The Method of Least Square');
disp('The value of P(1) which is the intercept is:')
P(1)
disp('The value of P(2)which is the slope is:')
P(2)
```

Here let us try to do the same problem as above and try to fit a straight line for the Ohm's law experiment.

Current (*X*-Axis)	4	6	8	9	10
Voltage (*Y*-Axis)	4	5	9	10	9

enter the number of data sets:5

```
n =
     5
enter the x value of data:4
x =
     4
enter the y value of data:4
y =
     4
enter the x value of data:6
x =
     4     6
enter the y value of data:5
y =
     4     5
enter the x value of data:8
x =
     4     6     8
enter the y value of data:9
y =
     4     5     9
enter the x value of data:9
x =
     4     6     8     9
enter the y value of data:10
y =
     4     5     9    10
enter the x value of data:10
x =
     4     6     8     9    10
```

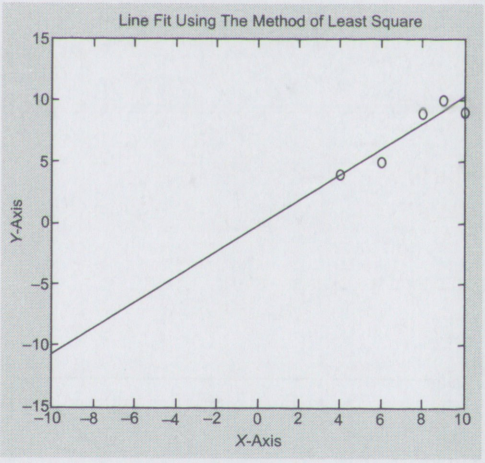
```
enter the y value of data:9
y =
     4     5     9    10     9
P =
   -0.3190
    1.0431
```

The value of $P(1)$ which is the intercept is:

```
ans =
   -0.3190
```

The value of $P(2)$ which is the slope is:

```
ans =
    1.0431
```



Here $P(1)$ gives the intercept of the y-axis and $P(2)$ gives the slope of the line. We find that slope of the best fit line obtained by using the least square method is 1.0431 which is the ne as obtained by fitting a line using MATLAB functions polyval and polyfit before.

CURVE FITTING USING BUILT-IN FUNCTIONS polyval AND polyfit

ng MATLAB we can also fit a polynomial curve easily using the built in functions. The two lt-in functions used are **polyfit** and **polyval**. For a given set of n data points it is fairly easy it a straight line or a curve of any order n using the built in functions **polyfit** and **polyval.** e objective here is to find the $n + 1$ coefficients (a_0, a_1, a_2, a_3, ... and a_n) of the polynomial ve that is the best fit curve for the $n + 1$ data points or more that are known to us. The best olynomial for the $(n + 1)$ points that are given will be of the form:

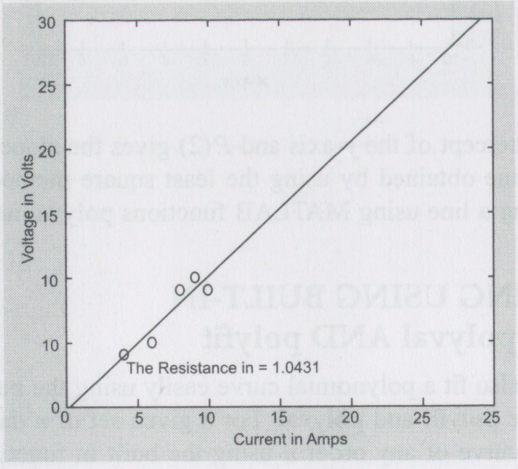
$$y = a_0 + a_1x + a^2x^2 + a_3x^3 + \ldots + a_nx^n$$

The MATLAB code for fitting a straight line through a set of given points as well as the be fit polynomials of higher degree using in-built MATLAB functions **polyfit** and **polyval** has be given below.

y = polyval(a, x) returns the value of a polynomial of degree n at x. The input argument a vector of length $(n + 1)$ whose elements are the coefficients in descending powers of t polynomial which has to be evaluated. On the other hand a = polyfit(x, y, n) finds the coefficier of a polynomial $a(x)$ of degree n that fits the data points $(x(1), y(1))$ to $(x(i)$ to $y(i))$. The le square method is adopted here. The result a is a row vector of length $(n + 1)$ containing t polynomial coefficients in descending powers:

The MATLAB program using polyval and polyfit to fit a straight line through the same da points given above is shown below.

```
x =[ 4 6 8 9 10];
y =[4 5 9 10 9];
a =polyfit(x,y,1);
d_fit = 0:20:300;
f_fit = polyval(a,d_fit);
plot(x,y, 'o', d_fit, f_fit)
axis([0 30 0 30])
xlabel('Current in Amps');
ylabel('Voltage in Volts');
k=a(1);
text(4,3,'The Resistance in ohms =');
text(15,3,num2str(k));
```

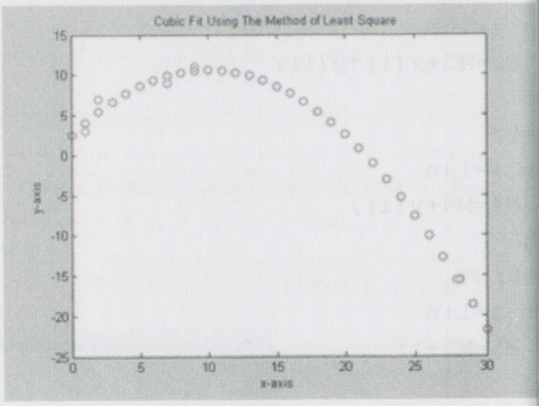


The resistance is found to be the same as found above.

8 CUBIC FIT USING LEAST SQUARE METHOD

```
=input('enter the  number of data sets:');
or i=1:n
  x(i)=input('enter the  x value of data:');
  y(i)=input('enter the  y value of data:');
nd
=0;
or i=1:n
 M1=(x(i)^2)+M1;
nd
=0;
or i=1:n
 M2=M2+x(i);
nd
=0;
or i=1:n
 M3=M3+x(i)*y(i);
nd
=0;
or i=1:n
 M4=M4+y(i);
nd
=0;
or i=1:n
 M5=M5+1;
nd
=0;
or i=1:n
 M6=M6+(x(i)^3);
nd
7=0;
or i=1:n
 M7=M7+ (x(i)^4);
 M8=0;
nd
or i=1:n
 M8=M8+(y(i)*(x(i)^2));
nd
[M5 M2 M1;
 M2 M1 M6;
 M1 M6 M7];
[M4;M3;M8];
```

```
P=M\L
xfit = 0:1:30;
yfit = P(1) + (P(2).*xfit) +(P(3).*xfit.*xfit);
plot(xfit,yfit,'ro',x,y,'go');
xlabel('x-axis')
ylabel('y-axis')
title('Cubic Fit Using The Method of Least Square');
disp('The value of P(1) is:')
P(1)
disp('The value of P(2) is:')
P(2)
disp('The value of P(3) is:')
P(3)
```

Output	Figure
enter the number of data sets:4 enter the *x* value of data:1 enter the *y* value of data:3 enter the *x* value of data:7 enter the *y* value of data:9 enter the *x* value of data:2 enter the *y* value of data:7 enter the *x* value of data:9 enter the *y* value of data:11 *P* = 2.5831 1.6082 −0.0807 The value of *P*(1) is: ans = 2.5831 The value of *P*(2) is: ans = 1.6082 The value of *P*(3) is: ans = −0.0807	

.9 FINDING ROOTS OF A POLYNOMIAL

A polynomial of degree n has exactly n roots and the roots may be real or complex. If the leading oefficient of a polynomial $f(x)$ of degree n is 1 and the roots of the polynomial are $x_1, x_2, x_3, \ldots,$ nd x_n, then,

$$f(x) = (x - x_1)(x - x_2)(x - x_3)\ldots(x - x_n)$$

Here, in this section the use of roots function for finding out the roots of a polynomial using MATLAB and the Newton-Raphson Method has been discussed.

.9.1 Finding Roots of a Polynomial Using the roots(f) MATLAB Function

et us assume a polynomial to be of the form,

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

Then such a polynomial is represented in MATLAB as,

$$f = [a_n\ a_{n-1} \ldots a_2\ a_1\ a_0]$$

Here the coefficient of the highest power of x comes first and the coefficient of the lowest ower of x comes last.

The roots of this polynomial can be found using the roots (f) function. Let us find the roots f the function,

$$f = x^3 - 1$$

Hence, in MATLAB it can be represented as,

$$f = [1\ 0\ 0\ -1]$$

The roots of this polynomial can be found using the command roots (f),

```
> f = [1 0 0 -1]
=
  1    0    0   -1
> roots (f)
ns =
-0.5000 + 0.8660i
-0.5000 - 0.8660i
 1.0000
```

Therefore, the real root of the equation is 1 and the other two roots are complex numbers.

9.2 The Newton-Raphson Method

is also possible to find the roots of a function using the Newton-Raphson method. The ewton-Raphson method converges relatively fast for most functions regardless of the initial lue. Here one needs to know not only the function $f(x)$, but also its derivative, $f'(x)$, to find a lution. In the Newton-Raphson technique an approximate value for the root is first chosen. If e approximate root is x_1 then a tangent to the curve $y = f(x)$ is drawn at the point $(x_1, f(x_1))$. ne point where this tangent cuts the x-axis, x_2 becomes the new approximate value of the root.

The process is repeated again and a tangent to the curve at $(x_2, f(x_2))$ is drawn. The point where this tangent cuts the x-axis, x_3, now becomes the new approximate value of the root. This process is repeated until for a value of $x = x_n$, obtained after n iterations, is obtained such that the value of the function at this point $x = x_n$ is very close to zero. The level of accuracy can be decided by us. For example, here we have chosen a root $x = x_n$ such that abs $(f(x_n)) < 10^{-5}$.

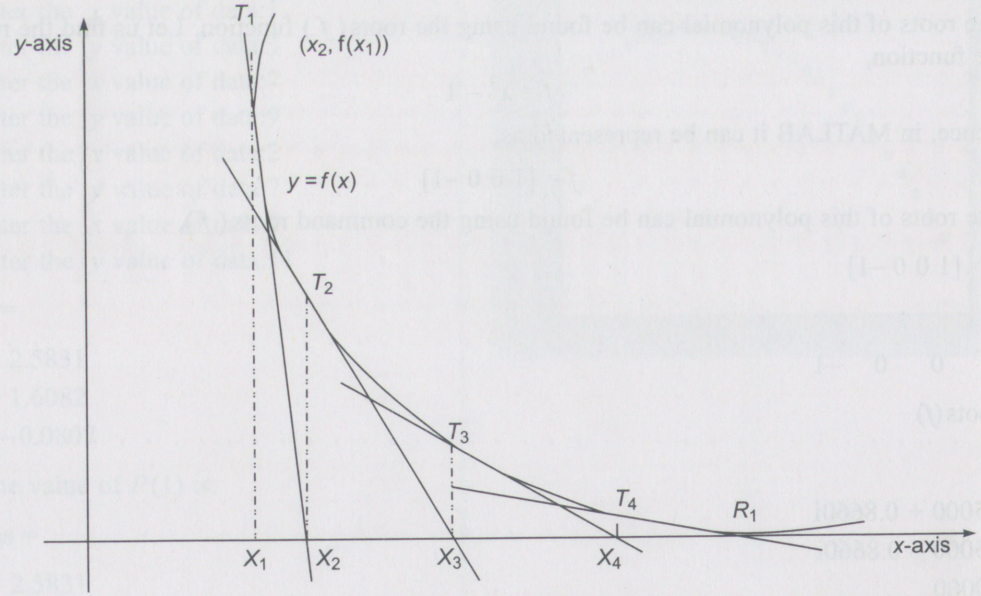
Let $y = f(x)$ be the function whose root has to be determined. The equation of the tangent at a point $(x_n, f(x_n))$ on the function $f(x)$ is given by,

$$f(x) = f(x_{n-1}) + f'(x_n)(x - x_n)$$

Putting $f(x) = 0$ in the above equation,

$$x_{n+1} = x_n - f(x_n)/f'(x_n)$$

Thus if x_n is known then x_{n+1} can be found from the above equation. This is the point where the tangent at the point $(x_n, f(x_n))$ cuts the x-axis. Initially, an approximate root is entered and gradually after several iterations the root of the polynomial is reached. Using the Newton Raphson method the root of a polynomial can be reached at a very fast rate. The figure below illustrates how the root is obtained using the Newton-Raphson method.



The Newton-Raphson method

The figure above gives a geometrical interpretation of the Newton-Raphson iteration. In the figure above, the curve $y = f(x)$ meets the x-axis at R_1. Let x_1 be the present estimate of the root. The tangent to the curve $y = f(x)$ at the point $(x_1, f(x_1))$ has the equation.

$$y = f(x_1) + (x - x_1) f'(x_1)$$

Let x_2 be the x-intercept of the tangent line on the x-axis. Then

$$x_2 = x_1 - f(x_1)/f'(x_1)$$

Let us now use the Newton-Raphson method to find out the root of the equation which has been considered earlier.

$$y = x^3 - 1$$

The MATLAB program for finding out the roots of the above equation is as follows.

```
a=input('enter a value of root:')
i=1;
for i=1:1:10
f=(a*a*a)-1;
fdash=3*a*a;
i=i+1;
anew=a-(f/fdash);
fnew=(anew*anew)-1;
i
anew
if (abs(fnew<0.0000001))
disp('root is:')
anew
break
else
    a=anew;
end
end
```

Output

```
enter a value of root:5

a =

     5

i =

     2

anew =

    3.3467

i =

     3

anew =

    2.2609

i =

     4

anew =

    1.5725

i =

     5

anew =

    1.1831

i =

     6

anew =

    1.0269

i =

     7

anew =

    1.0007

i =

     8

anew =

    1.0000

i =

     9

anew =

    1.0000

root is:

anew =

    1.0000
```

Thus, after 9 iterations the real root $x = 1$ is found.

7.10 SOLUTION TO ORDINARY DIFFERENTIAL EQUATIONS (ODEs)

In solving the ordinary differential equations (ODEs) there are two main reasons why the Euler's method is *not* commonly used in scientific computing. Firstly, the truncation error in every step associated with the Euler's method is larger than those with other methods. Secondly, the Euler's method is found to be prone to numerical instabilities. In most situations the 4th order Runge-Kutta integration method provides an appropriate compromise between low truncation error and computational cost per step.

7.10.1 The 4th Order Runge-Kutta Method

Runge-Kutta methods are one of the most popular ordinary differential equations (ODEs) solvers. They were first developed by Carle Runge and Martin Kutta around 1900. Later developments were made by John Butcher in the 1960s.The Runge-Kutta method is a method for numerically integrating ODEs by using a trial step at the midpoint of an interval to eliminate lower-order error terms. This is a simple method and is a general method for numerical solution of differential equations. The fourth order Runge-Kutta method is one of the most powerful predictor-corrector algorithms. As it is very accurate, most software packages that find numerical solutions for differential equations use it by default. This method is also often referred to as "RK4" in short or "the classical Runge-Kutta method" or simply "the Runge-Kutta method". The initial value problem is of the form

$$y' = f(t, y) \text{ and } y(t_o) = y_o$$

where y is the unknown function of time (t) which we would like to approximate and y' is the first derivative of y with respect to t and is a function of t and y. At the initial time ($t = t_o$) the corresponding y-value is y_o. In the problem the function f and the values t_o and y_o are given.

Let us consider a step-size $h > 0$,

$$y_{n+1} = y_n + 1/6h(k_1 + 2k_2 + 2k_3 + k_4)$$

$$t_{n+1} = t_n + h$$

For $n = 0, 1, 2, 3,\ldots$

where

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + 1/2h, y_n + h/2k_1)$$

$$k_3 = f(t_n + 1/2h, y_n + h/2k_2)$$

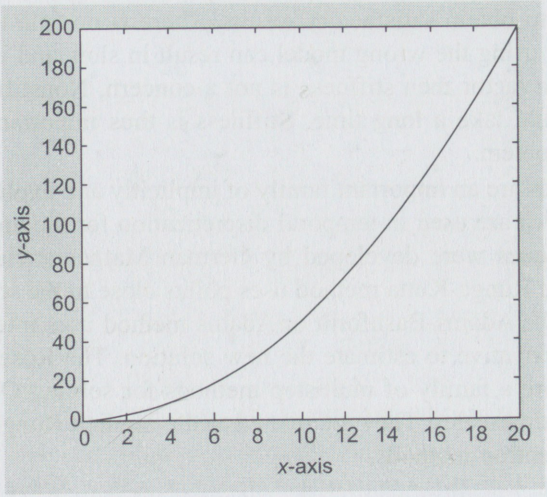
$$k_4 = f(t_n + h, y_n + hk_3)$$

y_{n+1} is the fourth order Runge-Kutta approximation of $y(t_{n+1})$, and y_{n+1} is determined from the y_n plus the weighted average of the four increments (k_1, k_2, k_3 and k_4). Each increment is the product of the size of the interval (h) and an estimated slope specified by function f.$y' = f(t, y)$ and $k_1 = f(t_n, y_n)$, k_1 is the increment based on the slope at the beginning of the interval obtained by using y' (Euler's method). k_2 is the increment based on the slope at the midpoint of the interval, using $y' + 1/2hk_1$. k_3 is the increment based on the slope at the midpoint, but now using $y' + 1/2hk_2$ and k_4 is the increment based on the slope at the end of the interval.

sing $y' + hk_3$. In averaging the four increments, greater weight is given to the increments at ıe midpoint. The weights are chosen such that if f is independent of y, so that the differential quation is equivalent to a simple integral. The RK4 method is a fourth-order method, meaning ıat the local truncation error is of the order of $O(h^5)$, while the total accumulated error is rder $O(h^4)$. The Simpson's rule gives very accurate numerical approximations for integrals ith only a few number of points. This is because the Simpson's rule is a fourth-order method. he error in the approximation is proportional to h^4 where h is the step size. In Simpson's rule, ve evaluate the function at the right endpoint, the left endpoint, and the middle, and then take weighted average of those values is considered. The analog of Simpson's rule for differential quations is the fourth order Runge-Kutta method.

The MATLAB program for solving an ODE using the 4th Order Runge-Kutta method is given elow.

```
=0.5;
 = 0:h:20;
 = zeros(1,length(x));
(1) = 0;
 = @(x,y) x;
or i=1:1:(length(x)-1)
    k1 = f(x(i),y(i));
    k2 = f(x(i)+0.5*h,y(i)+0.5*h*k1);
    k3 = f((x(i)+0.5*h),(y(i)+0.5*h*k2));
    k4 = f((x(i)+h),(y(i)+k3*h));
    y(i+1) = y(i) + (1/6)*(k1+2*k2+2*k3+k4)*h;
nd
lot(x,y)
label('x-axis')
label('y-axis')
```



7.10.2 ODE Solvers in MATLAB

The various ODE solvers available in MATLAB are:

Solver	Kind of problems that can be solved	Method used for solving the ODE
ode45	Nonstiff differential equations	Runge-Kutta
ode23	Nonstiff differential equations	Runge-Kutta
ode113	Nonstiff differential equations	Adams
ode15s	Stiff differential equations and DAEs (differential algebraic equations)	NDFs (BDFs) NDFs are numerical differentiation formulas and BDFs are backward differentiation formulas
ode23s	Stiff differential equations	Rosenbrock
ode23t	Moderately stiff differential equations and DAEs	Trapezoidal rule
ode23tb	Stiff differential equations	TR-BDF2
ode15i	Fully implicit differential equations	BDFs

In mathematics, a stiff differential equation is a differential equation for which certain numeric methods for solving the equation are numerically unstable, unless the step size is taken to b extremely small. An ODE is said to be stiff if the solution being sought is varying slowly bu there are nearby solutions that vary rapidly. Under such circumstances the numerical metho must take small steps to obtain satisfactory results. There is no rule to decide which is a sti or non-stiff system but using the wrong model can result in slow and inaccurate results. If tim of computation is not a factor then stiffness is not a concern. Nonstiff methods can solve sti problems but they would take a long time. Stiffness is thus important to decide how we ca efficiently solve the problem.

Runge-Kutta methods are an important family of implicitly and explicitly iterative methods i numerical analysis which are used in temporal discretization for the approximation of solution of ODE. These techniques were developed by German Mathematicians C. Runge and M.W Kutta around 1900. The Runge-Kutta method uses points close to the solutions value to evaluat derivative functions. The Adams-Bashforth or Adams method uses interpolation along with th current solution and derivative to estimate the new solution. The Rosenbrock methods for sti differential equations are a family of multistep methods for solving ODEs that contain a larg range of characteristic timescales. They are related to the implicit Runge–Kutta methods and ar also known as Kaps-Rentrop methods.

Firstly, a differential equation is any equation containing one or more derivatives. An rdinary differential equation often known as the ODE is an equality involving a function and s derivatives. An ODE of order n has the form,

$$F(t, y', y'', y''', \ldots, y^{(n)'}) = 0$$

here y is a function of t and $y' = dy/dt$ is the first derivative of y with respect to t, similarly, $'' = d^2y/dt^2$, which is the second derivative of y with respect to t. Finally, $y(n)' = d^ny/dt^n$ is ıe n^{th} derivative of y with respect to t. A first order differential equation is

$$y' = f(y)$$

A few examples of the first order differential equations are,

$$\sin(t) + \cos(t) = y' \quad \ldots(1)$$

$$y' + y = e^x \quad \ldots(2)$$

The order of the differential equation is the order of the highest derivative it contains. n ordinary differential equation or ODE is an equation containing a function of one independent ıriable and its derivatives. Here the term ordinary is used in contrast with the term partial fferential equation which may be with respect to more than one independent variable. In a ıear differential equation the solutions can be added and multiplied by coefficients and an exact osed-form solution is obtained. Whereas in a nonlinear ODE we do not have additive solutions ıd solving nonlinear ODE is much more complicated. An ordinary differential equation (ODE) a differential equation having a single independent variable, so the derivatives it contains are dinary derivatives. On the other hand a partial differential equation (PDE) is a differential uation with two or more independent variables and as a result the derivative(s) it contains are rtial derivatives.

10.3 Solving First-Order Equations Using ODE23 and ODE45

Using Inline Function

umerically, approximate solution of the first order differential equation

$$\frac{dy}{dx} = \sin(x) + \cos(y)$$

here $y(0) = 0$ and on the interval x lies in the range [0 2*pi] can be found using the ode solver, le23 or ode45, in MATLAB.

For any differential equation in the form $y' = f(x, y)$, we begin by defining the function , y). For single equations, we can define $f(x, y)$ as an inline function. Here, an inline function n be defined as shown below. The advantage of an inline function is that a function can be fined directly in the command window without creating an M-file.

```
> fn=inline('sin(x)+cos(y)')
=
```

Inline function:

fn(x, y) = sin(x)+cos(y)

This function can be solved using the ode solver ode23 or ode45 using the command giv below.

ode45(function, domain, initial condition).

```
>> [x,y]=ode45(fn,[0 2*pi],0)
```

MATLAB returns two column vectors, the first containing the values of x and the seco containing the values of y. As x and y are vectors it can be plot using the command,

```
>>plot(x,y)
```

2. *Using Inline M-Files*

It is also possible to solve the same ODE by first defining $f(x, y)$ in an M-file.

```
function ydash = first(x,y);
ydash = sin(x) + cos(y);
```

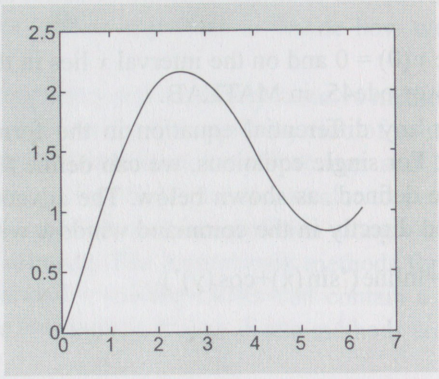
In this case we must use a pointer @ to indicate the M-file. That is, we use the followi commands.

```
>> xspan = [0 2*pi];
>> yo = 0;
>> [x,y]=ode45(@first,xspan,yo);
>> plot(x,y)
>>
```

The only difference in the two methods is the use of @. The ode solver ode45 is directed the file first.m using the @ symbol.

A comparison of the two methods is given below.

Command / M-file	Output

`>> fn=inline('sin(x)+cos(y)')`
`fn =`
`Inline function:`
`fn(x,y) = sin(x)+cos(y)`
`>> [x,y]=ode45(fn,[0 2*pi],0)`
	

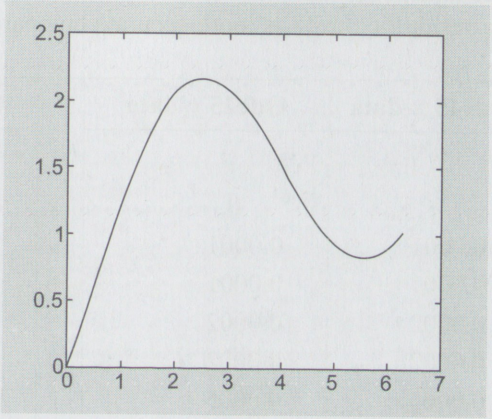
Con

```
> xspan = [0 2*pi];
> yo = 0;
> [x,y]=ode45(@first,xspan,yo);
> plot(x,y)
```

```
function ydash = first(x,y);
ydash = sin(x) + cos(y);
```

↓

his is the command in the file first.m file.



It should be noted that both the commands

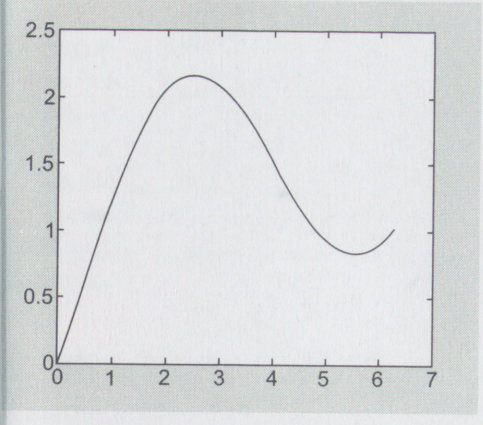
[x,y]=ode23 ('first',xspan,yo)

[x,y]=ode23 (@first,xspan,yo)

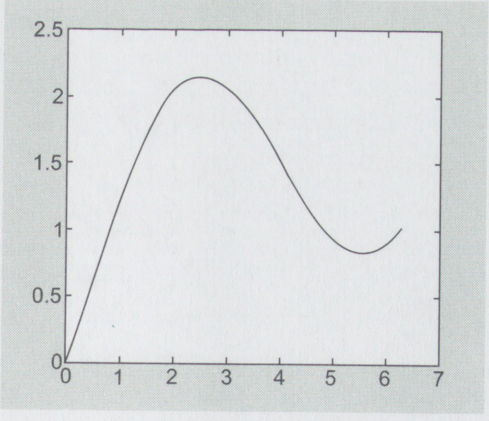
l give the same result.

0.4 Comparison of the Solution by ODE23 and ODE45

mparing the two plots obtained using ode45 and ode33 we can easily find out that the ode45 t is much smoother.



Solution using ode45



Solution using ode23

part of the data set of x and y from ode45 and ode23 is given below. The ode45 has a larger nber of data points compared to the ode23. The data points in ode45 are closer as compared

to the data points on ode23. This can be found by comparing the x data point in both the solv
As a result the ode45 is more accurate compared to the ode23.

Ode45 x-data	Ode45 y-data		Ode23 x-data	Ode23 y-data
$x =$	$y =$		$x =$	$y =$
0	0		0	0
0.0001	0.0001		0.0001	0.0001
0.0001	0.0001		0.0005	0.0005
0.0002	0.0002		0.0025	0.0025
0.0002	0.0002		0.0125	0.0126
0.0005	0.0005		0.0625	0.0644
0.0007	0.0007		0.1737	0.1878
0.0010	0.0010		0.3209	0.3653
0.0012	0.0012		0.4914	0.5839
0.0025	0.0025		0.6844	0.8368
...	...		...	...

8

HAPTER

Unconstrained Single Variable Optimization Methods

lexander Graham Bell (March 3, 1847 – August 2, 1922) was a noted scientist, inventor and engineer. In 76 at the age of 29 he invented the first telephone. AT&T was founded by Alexander Graham Bell in 1875. Few days later it carried the first sentence in Boston, USA. AT&T was the parent company of Bell Systems. e Bell systems provided the telephone services in America. Today AT&T is a global networking leader. All roughout AT&T was attached with research and innovation. Great inventions like the transistor, solar cell d communication satellites have taken place in AT&T labs. In 1984 AT&T gave up the telephone companies rwned and the logo was changed from a Bell to the new logo. Dennis Ritchie also worked at AT&T Bell Labs ng with colleagues like Ken Thompson and Brian Kernighan. UNIX was developed here at AT&T Bell Labs. nnis Ritchie developed C programming language here at AT&T Bell Labs."

is chapter explains the optimization methods used for finding optimal solution of an objective nction involving only one unknown variable and it is not subjected to any constraint. Since they al with only one variable, these methods are simple and easy to understand. All the techniques ated to the above problems can be classified into three major groups namely (a) bracketing thods (b) region elimination methods and (c) gradient-based methods. Although the details of se methods are explained in the following sections with an objective to minimize. They can also applied for maximization case after modifying objective function by taking its negative reciprocal.

1 BRACKETING METHODS

ie minimum point of an objective function is usually found in two stages. In the first stage, a mple and primitive method is used to find a search space containing the minimum. There are o methods under this category namely, (i) exhaustive search method and (ii) bounding phase thod. Both the methods find out an approximate interval that bracket the minimum point. nce, they are called as bracketing methods. In the next stage, a more advanced method is used find the minimum point with the desired accuracy by searching within the space identified in e first stage.

8.1.1 Exhaustive Search Method

This is the simplest and crudest numerical method to optimize a single variable objective functi when there are no constraints. Initially, a search domain is selected out of own experien assuming that the minimum point is believed to lie in this domain. This domain is equally divid into a number of sub-intervals. So it is also called an equal interval search method. The value the objective function is found out at each of these points separating the sub-intervals starti from the lower bound and proceeding towards the upper bound. At any time three consecuti points are taken and the values of objective function at these three points are compared. If t objective is to minimize and the function values are decreasing, search is continued by taki the immediate higher point and dropping the lowest point among the last three points considere Search is stopped when the function value starts increasing or the search over the entire doma is completed. The corresponding algorithm to minimize $f(x)$ where x is a real variable is outlin below.

Algorithm

Step 1: Read lower bound u, upper bound v, and number of divisions n
Find $d = (v - u)/n$
Set $x1 = u$, $x2 = x1 + d$, $x3 = x2 + d$

Step 2: Find $f(x1)$, $f(x2)$ and $f(x3)$
If $f(x1) \geq f(x2) \leq f(x3)$
Print $x^* \in (x_1, x_3)$ and stop
Else $x1 = x2$; $x2 = x3$; $x3 = x2 + d$

Step 3: If $x3 > v$
x^* does not lie in (a, b) or $x^* = a$ or b

The accuracy of the result obtained using this method is the length of the final interv (x_1, x_3) in which the minimum point lies.

The MATLAB program for the above algorithm is written in a file named *exhaust.m* as giv below.

```
%minimize a single variable function without any constraints usir
%exhaustive search Method
%file name: exhaust.m
%uses function file :f.m
%%%%%%%%%
u = input('Enter lower bound of search space u = ');
v = input('Enter upper bound of search space v = ');
n = input('Enter number of equal divisions of search space n = ');
global NF; % number of function evaluations NF as global variable
NF = 0;
x1 = u;
d = (v-u)/n;
```

Cont

```
2 = x1+d;
3 = x2+d;
1 = f(x1); f2 = f(x2);    f3 = f(x3);
 = 1; % iteration counter
hile(x3 <= v)
printf('f(%.3f)= %.3f, \t f(%.3f)= %.3f, \t f(%.3f)= %.3f\n', x1,f1,
2,f2,x3,f3);
f f1 >= f2 && f2 <= f3
     fprintf('Minimum point lies in the interval (%.3f,%.3f)\n', x1, x3);
     fprintf('No. of iterations = %d \n',k);
     fprintf('No. of function evaluations = %d \n',NF);
     return;
lse
     x1 = x2; x2 = x3; x3 = x2+d;
     f1 = f2; f2 = f3;    f3 = f(x3);
     k = k+1;
nd % end if
nd % end while
f x3 > v
     fprintf('Minimum point does not lie in (%.3f,%.3f)\n', u, v);
nd
```

To illustrate the working of the program, let us consider the problem to minimize $f(x) = x^2 + 6/x$ over the interval (0.1, 5.1). Accordingly, the MATLAB program for this function is written a file named *f.m* as outlined below.

```
unction [y] = f(x)
 returns the value of function f(x) = x^2 +16/x for a given value of x
 ensure that x != 0
 file name: f.m
lobal NF; % number of function evaluations NF as global variable
y = x.*x +16.0./x;
NF = NF+1;
eturn
```

As the purpose is to get familiarized with how the optimization method works, a simple unction like $f(x) = 16/x$ with known solution has been considered here. The solution of this unction can be easily found by the knowledge of differentiation. The first derivative, $f(x) = 2x - 6/x^2 = 0$ at optimal points. Solving this equation, $2x^3 - 16 = 0 \Rightarrow x = +2$. The double derivative $''(x) = 2 + 32/x^3$. At $x = 2$, $f''(x) = 2 + 32/2^3 = 6$. Since, $6 > 0$, $x = 2$ is a local minimum point.

Now, let us minimize the same function using exhaustive search method and check the orrectness of the solution obtained. On running the program, the following input values are rompted to key in. The input data fed by the user are shown underlined for clarity.

```
>>exhaust
Enter lower bound of search space u = 0.1
Enter upper bound of search space v = 5.1
Enter number of equal divisions of search space n = 10
f(0.100) = 160.010,    f(0.600) = 27.027,    f(1.100) = 15.755
f(0.600) = 27.027,     f(1.100) = 15.755,    f(1.600) = 12.560
f(1.100) = 15.755,     f(1.600) = 12.560,    f(2.100) = 12.029
f(1.600) = 12.560,     f(2.100) = 12.029,    f(2.600) = 12.914
Minimum point lies in the interval (1.600,2.600)
No. of iterations = 4
No. of function evaluations = 6
>>
```

The result obtained is thus correct as the local minima that occurs at $x = 2$ is contained i the interval (1.6, 2.6). The function values at each iteration have been listed to illustrate hov the search process proceeds under this method. The result is obtained after 4 iterations ($k = 4$) In the first iteration there are 3 function evaluations. Subsequently, there is only one evaluatio per iteration. Therefore, the total number of function evaluations becomes 6 ($n_f = 3 + k–1$). I stops when for the three consecutive points $x1$, $x2$ and $x3$ the condition $f(x1) \geq f(x2) \leq f(x3)$ i satisfied. At last interaction $f(1.6) = 12.56 \geq f(2.1) = 12.029 \leq f(2.6) = 12.914$. The accuracy of the result obtained is 2.6 – 1.6 = 1.0 i.e., 2 times of the step size i.e., (5.1 – 0.1)/10 = 0.5 The accuracy will be better if the step size is smaller, i.e., when number of divisions is larger For example, if the same program is run with $n = 1000$ keeping u and v values same as before i.e., 0.1 and 5.1, the result will be as shown below.

```
>>exhaust
Enter lower bound of search space u = 0.1
Enter upper bound of search space v = 5.1
Enter number of equal divisions of search space n = 1000
Minimum point lies in the interval (1.995, 2.005)
No. of iterations = 380
No. of function evaluations = 382
>>
```

The intermediate results for a few iterations have been shown and the results for the remaining iterations have been suppressed because of large number of iterations. The accuracy of results is now 2.005 – 1.995 = 0.01.

If the initial search space (u, v) is not properly selected, then the program will continue to find out function values till all the intermediate points in the search space is exhausted and finally shows the results that minimum point does not lie in the interval selected. For example, let $u = 3$, $v = 5$ and $n = 10$. Then, the output of the program will be as shown below.

```
>>exhaust
Enter lower bound of search space u = 3
Enter upper bound of search space v = 5
Enter number of equal divisions of search space n = 10
f(3.000) = 14.333,   f(3.200) = 15.240,   f(3.400) = 16.266
f(3.200) = 15.240,   f(3.400) = 16.266,   f(3.600) = 17.404
f(3.400) = 16.266,   f(3.600) = 17.404,   f(3.800) = 18.651
f(3.600) = 17.404,   f(3.800) = 18.651,   f(4.000) = 20.000
f(3.800) = 18.651,   f(4.000) = 20.000,   f(4.200) = 21.450
f(4.000) = 20.000,   f(4.200) = 21.450,   f(4.400) = 22.996
f(4.200) = 21.450,   f(4.400) = 22.996,   f(4.600) = 24.638
f(4.400) = 22.996,   f(4.600) = 24.638,   f(4.800) = 26.373
Minimum point does not lie in (3.000,5.000)
>>
```

In case when such a result occurs, the program should be rerun assuming some other values of u and v such that it is expected to contain the minimum point.

3.1.2 Bounding Phase Method

Exhaustive search method may or may not be successful in finding the minimum point but bounding phase method is always able to find out the minimum of a function if it exists. This is also applicable to a single variable unconstrained minimization problem. In the beginning it assumes an arbitrary guess value. Then it determines the search direction based upon the values of the objective function at this arbitrary point and two more adjacent points lying on its opposite sides at equal distances. If the function value decreases with the increase in x value, the search proceeds in the positive direction otherwise the search is done in the negative direction. Once search direction is finalized, the function values are calculated at the points lying at distances having some sort of exponential relationship till the minimum point is reached. The detailed procedure can be better understood from the algorithm mentioned below.

Algorithm

Step 1: Read start value x_0 and a positive value for step size d
Set $k = 0$

Step 2: Find $x_{01} = x_0 - d$, $x_{02} = x_0 + d$, $f(x_{01})$, $f(x_0)$ and $f(x_{02})$
If $f(x_{01}) \geq f(x_0) \geq f(x_{02})$
d is +ve

else if $f(x_{01}) \leq f(x_0) \leq f(x_{02})$

d is –ve

else go to Step 1 and change values of x_0 and d.

Step 3: Set $x_{k+1} = x_k + 2^k.d$

Step 4: If $f(x_{k+1}) < f(x_k)$

Set $k = k + 1$ and go to Step 3

Else print $x^* \in (x_{k-1}, x_{k+1})$ and stop

The corresponding MATLAB program to find the minimum of a single variable functio without any constraint is given below.

```
%minimize an unconstrained single variable function using boundin
%phase Method
%file name: bounding.m
%uses function file :f.m
%%%%%%%%%
clc;
clear all;
global NF; % number of function evaluations NF as global variable
NF = 0;
k=0;
while (1) % read initial values to decide search direction
x0=input('Enter initial guess value:');
d=input('Enter a +ve value for increment:');
x = [x0-d x0 x0+d];
fx = f(x);
disp('x = ');disp(x);
disp('fx = ');disp(fx);
if (fx(1) >= fx(2) && fx(2) >= fx(3))
     break;
elseif (fx(1) <= fx(2) && fx(2) <= fx(3))
     d = -1 * d; break;
else disp('Enter new set of initial values');
end %if
end %while
x(1) = x0-d;
fprintf('Initial value x0 = %.3f\n', x0);
fprintf('Increment d = %.3f\n', d);
while(1)
     x(3) = x(2) + 2^k*d;    % new x value
     fx(3) = f(x(3));        % new f(x) value
```

```
    fprintf('k = %3d\n',k);
    disp('x = '); disp(x);
    disp('fx = '); disp(fx);
    if fx(3) <fx(2)
            k = k+1;
            x(1) = x(2); x(2) = x(3); %discard the oldest x value &
            update
            fx(1) = fx(2); fx(2) = fx(3); % find corresponding f(x) value
    else
    if (d > 0)
            fprintf('Minimum point lies in interval (%.3f, %.3f)\n',x(1),
            x(3));
    else
            fprintf('Minimum point lies in interval (%.3f, %.3f)\n',x(3),
            x(1));
    end
    fprintf('No. of iterations = %d\n',k);
    fprintf('No. of function evaluations = %d \n',NF);
    break;
    end
end
```

When the above program is executed, the following output is obtained.

```
>>bounding
Enter initial guess value:0.6
Enter a +ve value for increment:0.5
x =
   0.1000    0.6000    1.1000
fx =
  160.0100   27.0267   15.7555
Initial value x0 = 0.600
Increment d = 0.500
k =  0
x =
   0.1000    0.6000    1.1000
fx =
  160.0100   27.0267   15.7555
```

Contd.

Contd.

```
k =  1
x =
    0.6000    1.1000    2.1000
fx =
   27.0267   15.7555   12.0290
k =  2
x =
    1.1000    2.1000    4.1000

fx =
   15.7555   12.0290   20.7124
Minimum point lies in interval (1.100, 4.100)
No. of iterations = 2
No. of function evaluations = 4
>>
```

The explanation of the above output is given below to understand how the program has worked. The data fed into the computer from the screen are stored as $x_0 = 0.6$ and $d = 0.5$. The counter is set at $k = 0$. Thus, the vector $\boldsymbol{x}$ consisting of three elements $x_0 - d$, x_0 and $x_0 + d$ will have values as

$$\boldsymbol{x} = [0.1000 \quad 0.6000 \quad 1.1000]$$

The function values at each of these three x values are calculated and stored in the vector $\boldsymbol{fx}$ as

$$\boldsymbol{fx} = [f(x_0 - d)\, f(x_0)\, f(x_0 + d)] = [f(0.1)\, f(0.6)\, f(1.1)] = [160.0100 \; 27.0267 \; 15.7555]$$

Taking first one as the sample calculation, $f(0.1) = 0.1^2 + 16/0.1 = 160.0100$.

Since, $160.0100 \geq 27.0267 \geq 15.7555$, we set increment $d = +0.5$ and initial giess $x_0 = 0.6$.

With $k = 0$ and the next guess is calculated as $x_1 = x_0 + 2^0.d = 0.6 + 1(5) = 1.1$ and stored as the 3rd element of vector $\boldsymbol{x}$. Since, the function value $f(1.1) = 15.7555$ is less than $f(0.6) = 27.0267$, k is incremented from 0 to 1. The 1st element of vector $\boldsymbol{x}$ is discarded and the values of $\boldsymbol{x}$ are shifted from right to left. Thus, new values of first two elements of x are replaced with old values of its last two elements, i.e., new $\boldsymbol{x}(1)$ = old $\boldsymbol{x}(2) = 0.6$ and new $\boldsymbol{x}(2)$ = old $\boldsymbol{x}(3) = 1.1$. Similarly, the first two elements of vector $\boldsymbol{fx}$ are replaced with the corresponding function values, i.e., new $\boldsymbol{fx}(1)$ = old $\boldsymbol{fx}(2) = 27.0267$ and new $\boldsymbol{fx}(2)$ = old $\boldsymbol{fx}(3) = 15.7555$. The 3rd element of vector $\boldsymbol{x}$ always takes the next guess value calculated by the expression $x_{k+1} = x_k + 2^k d$. With $k = 1$, this gives $x_2 = x_1 + 2^1.d = 1.1 + 2(0.5) = 2.1$. Thus, $\boldsymbol{x}(3) = 2.1$. The function value $f(2.1) = 2.1^2 + 16/2.1 = 12.0290$ is stored as the 3rd element of vector $\boldsymbol{fx}$. At this stage, the values of k, $\boldsymbol{x}$ and $\boldsymbol{fx}$ are

```
k =    1
x = [0.6000     1.1000     2.1000]
fx = [27.0267    15.7555    12.0290]
```

Since, $f(2.1) = 12.0290$ is smaller than $f(1.1) = 15.7555$, it proceeds to next iteration and $k = 1$ is incremented by 1 to $k = 2$. Like in the previous iteration, the values of first two elements of vectors $\boldsymbol{x}$ and $\boldsymbol{fx}$ are updated with the values of last two elements. The 1st column values of x and fx are discarded. The 2nd and 3rd column values occupy 1st and 2nd columns respectively. The new values occupy 3rd column. Thus putting $k = 2$, the new value of $\boldsymbol{x}$ is obtained as $x_3 = x_2 + 2^2 d = 2.1 + 4(0.5) = 4.1$. The corresponding function value $f(4.1) = 20.7124$. Thus, values of x and fx with $k = 2$ will be

```
k =    2
x = [1.1000     2.1000     4.1000]
fx = [15.7555    12.0290    20.7124]
```

Now, $f(4.1) = 20.7124$ is no longer less than its previous value, i.e., $f(2.1) = 12.0290$. Here, the program stops with the final result displaying that minimum point lies in the interval (x_{k-1}, x_{k+1}). As $k = 2$, this interval becomes (x_1, x_3) i.e. (1.1, 4.1). The result is obtained within 2 iterations and the accuracy obtained is $4.1 - 1.1 = 3.0$.

To improve the accuracy, the increment d should be taken a smaller value. If the same program is run with $d = 0.001$ keeping the same value of $x_0 = 0.6$ the final result will be as given below supposing the intermediate results.

Output for the same program run with d = 0.001 keeping the same value of x0 = 0.6 the result is as given below. Iterations for k = 2 to k = 8 have been omitted here.

```
Enter initial guess value:0.6
Enter a +ve value for increment:0.001
Minimum point lies in interval (1.111, 2.647)
No. of iterations = 10
No. of function evaluations = 12
>>
```

Even if d takes a value as small as 0.001, the accuracy has not improved beyond $2.647 - 1.111 = 1.536$. The accuracy obtained with this method is poor. Therefore, this method is not recommended for any optimization problem demanding reasonable amount of accuracy. However, this method has the ability of giving an approximate result with in very few iterations. In this example, although d is as small as 0.001, it has taken only 10 iterations to give the final result. Therefore, this method is best suitable to be used as the first step for giving an approximate idea about the interval containing the minimum point. The result of this method can be taken as starting point for other algorithms to find out the minimum point with greater accuracy. However, the exhaustive search method has the ability to find the result within the desired accuracy, but

this method is very slow. As seen earlier, taking input data as $u = 0.1$, $v = 5.1$ and $n = 1000$ the result of exhaust.m was obtained with 380 iterations and 382 function evaluations which is much higher compared to 10 iterations and 12 function evaluations with bounding phase method.

The exhaustive search method is slower because it divides the search space (u, v) into a very large number of intermediate points and it continues to check for minimum point at each and every intermediate point starting from one end.

Let us now see the working of other features of bounding phase method by taking different sets of input data. Taking $x_0 = 3$ and $d = 0.5$, the following output is obtained

```
>>bounding
Enter initial guess value:3
Enter a +ve value for increment:0.5
x =
    2.5000    3.0000    3.5000
fx =
   12.6500   14.3333   16.8214
Initial value x0 = 3.000
Increment d = -0.500
k =  0
x =
    3.5000    3.0000    2.5000
fx =
   12.6500   14.3333   12.6500
k =  1
x =
    3.0000    2.5000    1.5000
fx =
   14.3333   12.6500   12.9167
Minimum point lies in interval (1.500, 3.000)
No. of iterations = 1
No. of function evaluations = 3
>>
```

In the above case, since initial point taken is on the right side of minimum point the search proceeds in the backward direction, i.e., from right to left. To make it possible, d is made –ve i.e. $d = -0.5$.

In another case, let us assume $x0 = 1.8$ and $d = 0.5$ and run the same program again.

```
>>bounding
Enter initial guess value:1.8
Enter a +ve value for increment:0.5
x =
    1.3000    1.8000    2.3000
fx =
   13.9977   12.1289   12.2465
Enter initial guess value:
```

In the above case, it stops without giving the result and asks for giving new set of initial values for x_0 and d. This happens when x_0 is close to minimum point and the three points $x_0 - d$, x_0 and $x_0 + d$ are not on the same side of the minimum point.

3.2 REGION ELIMINATION METHODS

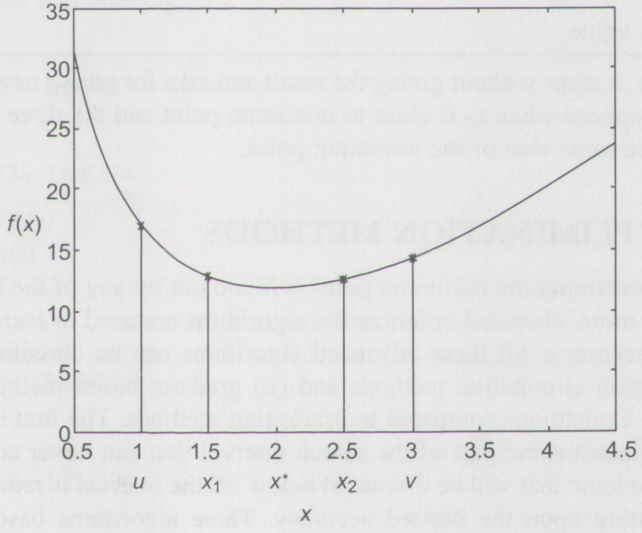
After the interval containing the minimum point is found out by any of the bracketing methods, in the second stage more advanced optimization algorithms are used to search for the minimum point with higher accuracy. All these advanced algorithms can be classified under two major groups, viz., (i) region elimination methods and (ii) gradient based methods. They need less number of function evaluations compared to bracketing methods. The first category of methods systematically eliminates some part of the search interval that can never contain the minimum point based on some logic that will be discussed below till the interval is reduced to a sufficiently smaller one depending upon the desired accuracy. Three algorithms based on this principle namely (i) interval halving method (ii) Fibonacci search method and (iii) golden section method are discussed in this section. All these methods use only the values of objective function and not its derivatives but, there is another group of methods called gradient-based methods that needs the derivatives of the objective function to find the minimum point.

For an objective function having one minimum point, the part of the search region that does not contain the minimum point can be identified by comparing the values of the function at two points as per the following logic.

Let x_1 and x_2 be any two intermediate points lying inside the interval (u, v) containing the minimum point and the point x_1 lies to the left of the point x_2 (i.e., $x_1 < x_2$). The region that will not contain the minimum point will be:

- (u, x_1) if $f(x_1) > f(x_2)$ i.e., left of x_1
- (x_2, v) if $f(x_1) < f(x_2)$ i.e., right of x_2
- (u, x_1) and (x_2, v) if $f(x_1) = f(x_2)$ i.e., both left of x_1 and right of x_2

The graphical plot of $f(x) = x^2 + 16/x$ has been shown in the figure below. It shows that has one minimum point at $x^* = 2$. Two points x_1 and x_2 are so selected in the search interva (u, v) that $u < x_1 < x_2 < v$. If the value of function f at x_1 is more than that at x_2, it is certain tha the minimum point will not be present on the left side of x_1 and the entire region from lowe boundary u up to x_1 can be eliminated from the search space. Thus, the search space gets reduce from (u, v) to (x_1, v). Otherwise, (u, v) will reduce to (u, x_2) if function value at x_1 is less tha that at x_2. The probability of arriving at the third possibility in numerical computation is ver rare. If it occurs, then both left and right regions i.e., (u, x_1) and (x_2, v) will be eliminated an the interval will be reduced to (x_1, x_2). Three algorithms based on the above fundamental concep of region elimination are explained in the following sub-sections.



Plot of the objective function f(x) = x² + 16/x

8.2.1 Interval Halving Method

This method begins with the assumption of an initial search space (u, v) that is known to hav the minimum point x^* within it from the results of some other algorithm. Here, u and v are th lower and upper bounds of the initial search space. Three intermediate points x_1, x_m and $x_2 (x_1 < x < x_2)$ are so selected that they divide this search space into four equal parts. Here, x_m become the mid-point of the entire search space dividing it into two halves. Similarly, the other tw points x_1 and x_2 are the mid-points of left and right halves of the search space respectively. I other words, x_1, x_m and x_2 are the first, second and third quartile points located at distances o $1/4^{th}$, $2/4^{th}$ and $3/4^{th}$ of the width of search space from lower bound.

Function values at these three intermediate points are evaluated. However, at any time, nction values at only two consecutive points (i.e., x_1 with x_m or x_2 with x_m) are compared and me portion of the search space is eliminated following the rule of region elimination already xplained in previous section. The comparison leads to any of the following three cases.

ase 1: If $f(x_1) < f(x_m)$, then the function value goes on increasing as x value increases beyond . Thus, the minimum point x^* cannot exist anywhere beyond x_m. So, the portion of search ace lying right side of x_m is eliminated. As x_m is the mid-point, the entire right half is struck off ading to 50% reduction. As a result, the interval (u, v) is reduced to only the left half i.e., (u, x_m).

ase 2: If $f(x_2) < f(x_m)$, the function values are higher for all the points lying left to the mid-int x_m. Thus, the minimum point x^* cannot lie anywhere in the region (u, x_m) i.e., the left half the search space. So, the region (u, x_m) is eliminated and the search space is reduced from , v) to its right half only i.e., (x_m, v) resulting in 50% reduction.

ase 3: This happens when neither of the above two cases occur. Thus, it consists of two sub-ses, i.e., when (i) $f(x_1) \geq f(x_m)$ and (ii) $f(x_2) \geq f(x_m)$.

When Case 1 does not occur, the relationship obtained is $f(x_1) \geq f(x_m)$. Hence, the function values are higher for all x values less than x_1. Thus, the minimum point x^* cannot lie anywhere in the region left to the point x_1. So, the region (u, x_1) is eliminated and the search space gets reduced to (x_1, v). Since, the point x_1 is located at a distance of one-fourth of the search space, the reduction is 25%.

Similarly, the relationship becomes $f(x_2) \geq f(x_m)$ when Case 2 does not happen. Following the same logic, the minimum point cannot exist in the region (x_2, v) and this portion is eliminated. x_2 being the third-quartile point of the search space (u, v), here also the reduction is 25%. Thus, combining both the sub-cases, the total reduction in Case 3 becomes 50%.

It is observed that whichever out of the above 3 cases occur, the search interval gets reduced exactly by half in each iteration. Hence, this method is named as interval halving method.

The step-by-step algorithm for this method is presented below.

ep 1: Select initial search space (u, v) and a small number e for desired level of accuracy
Calculate $w = v - u$ and $x_m = (u + v)/2$
Evaluate $f(x_m)$

ep 2: Set $x_1 = u + w/4$ and $x_2 = v - w/4$
Evaluate $f(x_1)$ and $f(x_2)$

ep 3: If $f(x_1) < f(x_m)$, set $v = x_m$, $x_m = x_1$; go to Step 5
Else go to Step 4

ep 4: If $f(x_2) < f(x_m)$, set $u = x_m$, $x_m = x_2$; go to Step 5
Else set $u = x_1$ and $v = x_2$

Step 5: Find $w = v - u$

If $|w| < e$, print $x^* \in (u, v)$ and stop

Else go to Step 2.

The corresponding MATLAB code is given below.

```
%minimize an unconstrained single variable function using Interval Halvi
%Method
%file name: intervalhalf.m
%uses function file :f.m
%%%%%%%%%
clc;
clear all;
global NF; % number of function evaluations NF as global variable
NF = 0;
k = 1; % number of iteration
u = input('Enter lower bound u = ');
v = input('Enter upper bound v = ');
e = input('Enter desired accuracy e = ');
w = v-u;      % length of search space
xm = (u+v)/2; % midpoint of search space
fm = f(xm); % f value at midpoint
fprintf('IterkLow,u\t  x1[f(x1)]\t\t  xm[f(xm)]\t  x2[f(x2)]\t\t  High,v
width,w   NF\n');
    while(abs(w)> e) % termination criteria
    w4 = w/4;
    x1 = u + w4; x2 = v - w4; % two intermediate points
    f1 = f(x1); f2 = f(x2);
    fprintf('%3d\t  %.3f\t  %.3f[%.3f]\t  %.3f[%.3f]\t  %.3f[%.3f]\t  %.3f
    %.3f\t %3d\n',k, u,x1,f1, xm, fm, x2,f2,v, w, NF);
if f1 < fm
    v = xm; xm = x1; fm = f1;%Case 1:delete right half, x1 as new mid-poi
elseif f2 <fm
    u = xm; xm = x2; fm = f2;%Case 2:delete left half, x2 as new mid-poi
else
    u = x1; v = x2;%Case 3: delete 25% from each end, midpoint unchang
    end  %end of if
    w = v - u; k = k+1;
    end  %end of while
fprintf('Minimum point lies in interval (%.3f, %.3f)\n',u,v);
fprintf('No. of iterations = %d \n',k-1);
fprintf('No. of function evaluations = %d \n',NF);
```

On execution of the above program along with the same function file *f.m* used by the earlier ɔgrams, the following messages are displayed on the screen.

```
>intervalhalf
nter lower bound u = 0
nter upper bound v = 5
nter desired accuracy e = 0.1
```

On entering the input data one by one for the three variables u, v and e (as shown above derlined) the following output is generated.

ter, k	Low, u	$x1[f(x1)]$	$xm[f(xm)]$	$x2[f(x2)]$	High, v	width, w	NF
1	0.000	1.250[14.363]	2.500[12.650]	3.750[18.329]	5.000	5.000	3
2	1.250	1.875[12.049]	2.500[12.650]	3.125[14.886]	3.750	2.500	5
3	1.250	1.563[12.681]	1.875[12.049]	2.188[12.099]	2.500	1.250	7
4	1.563	1.719[12.263]	1.875[12.049]	2.031[12.003]	2.188	0.625	9
5	1.875	1.953[12.007]	2.031[12.003]	2.109[12.035]	2.188	0.313	11
6	1.953	1.992[12.000]	2.031[12.003]	2.070[12.014]	2.109	0.156	13

nimum point lies in interval (1.953, 2.031)
. of iterations = 6
. of function evaluations = 13

Let us now discuss how the program has worked out the above results. At the beginning, NF declared as global variable to store the number of times the function $f(x)$ has been evaluated calling the function file f.m. The value of iteration counter k is initialized as 1. The following ee variables store the input data as given below:

Lower bound of search space $u = 0$, upper bound of search space $v = 5$ and desired accuracy 0.1.

First, the initial width of search space is calculated as $w = v - u = 5 - 0 = 5$. Then, the mid-nt of the search space becomes $xm = (u + v)/2 = (0 + 5)/2 = 2.5$. The value of function f at mid-point is evaluated as $fm = f(xm) = f(2.5) = 2.5^2 + 16/2.5 = 12.650$. Then, it prints the dline with titles for all the columns. Before entering the while loop, it checks if absolute value the width of search space $|w|$ is larger than the desired accuracy e. At this stage, $|w| = |+5| =$ nd $e = 0.1$. Since, $5 > 0.01$, it enters the while loop.

Two more intermediate points representing the midpoints of left and right halves respectively set at $x1 = u + w/4 = 0 + 5/4 = 1.25$ and $x2 = v - w/4 = 5 - 5/4 = 3.75$. The corresponding ction values are $f1 = f(x1) = f(1.25) = 1.25^2 + 16/1.25 = 14.363$ and similarly, $f2 = f(x2)$ .75) = 18.329. It now prints various values as $k = 1$, $u = 0$, $x1 = 1.25$, $f(x1) = 14.363$, $xm =$, $f(xm) = 12.650$, $x2 = 3.75$, $f(x2) = 18.329$, $v = 5.0$, $w = 5.0$. The last column displays the

number of function evaluations made so far, i.e., NF = 3 because it has evaluated $f(x1)$, $f($ and $f(xm)$ in the first iteration.

Then, it checks if $f1 < fm$. This is false as 14.363 > 12.650. So, it goes ahead to check if f fm. This is also false as 18.329 > 12.650. This represents Case 3. So, it eliminates $1/4^{th}$ reg each from either end of the search space. Accordingly, the two intervals (0, 1.25) and (3.75, are eliminated. This is numerically done by setting $u = x1 = 1.25$ and $v = x2 = 3.75$. Thus, search space (0, 5) gets reduced to (1.25, 3.75), the width of which is now $w = 3.75 - 1.25$ 2.5 i.e. exactly half of the previous width ($w = 5$). This completes one iteration and then it g back to the beginning of while loop to begin the next iteration i.e. $k = k + 1 = 1 + 1 = 2$. Sin $|w| = 2.5 > e = 0.1$, it moves ahead to repeat all the operations mentioned inside while loop t were done in the previous iteration.

In 2nd iteration, the mid-point remains unchanged at $xm = (1.25 + 3.75)/2 = 2.5$. Therefo there is no need to evaluate $f(xm)$. So in the program, it does not calculate either xm or $f(xm$ Case 3 occurs. However, $x1$ and $x2$ take new values as $x1 = 1.25 + 2.5/4 = 1.875$ and $x2 = 3.7$ $2.5/4 = 3.125$. Accordingly, only two new function values are required to be evaluated, i.e., $f($ $= 12.049$ and $f(x2) = 14.886$. Since $f(x1) = 12.049 < f(xm) = 12.650$, it represents Case 1. the right half (2.5, 3.75) is deleted by setting $v = xm = 2.5$. The interval (1.25, 3.75) is redu to (1.25, 2.5) of which the mid-point becomes $xm = (1.25 + 2.5)/2 = 1.875$ which is same as $x1$ and therefore function evaluation at this point is not required again. It can be observed in the subsequent iterations that one out of three intermediate points, i.e., $x1$, xm and $x2$ rema unchanged and only two function evaluations are required. The first iteration is the except where the function is required to be evaluated thrice. Thus, there is a relationship $nf = 2k +$

The region eliminations which take place under Case 1 and Case 3 have been illustra above. Let us now see how Case 2 works. For example, at 4th iteration: $u = 1.563$, $x1 = 1.7$ $xm = 1.875$, $x2 = 2.031$ and $v = 2.188$. Here, $f(x2) = 12.003$ is less than $f(xm) = 12.049$. So, left half i.e. (1.563, 1.875) is eliminated and the interval is reduced to ($u = 1.875$, $v = 2.18$ Thus, new mid-point $xm = (1.875 + 2.188)/2 = 2.031$ becomes same as previous $x2$. So, funct evaluation at xm is not required.

After 13 function evaluations in 6 iterations, the minimum point is found to be in the inter (1.953, 2.031) and the program stops. Thus, the accuracy of result obtained is $2.031 - 1.95$ 0.078 ($< e = 0.1$). Since the interval is reduced to its half in each iteration, after k iterati the width of search space will be $(v - u)/ 2^k$, i.e., in terms of number of function evaluations $(v - u)/2^{(nf-1)/2}$. Thus, the total number of function evaluations necessary for attaining an expec accuracy of e can be found out by solving the equation $(v - u)/2^{(nf-1)/2} = e$. Usually in practi value of e is very low and therefore nf is large. In that case, this equation can be simplified $(v - u)/2^{(nf/2)} = e$.

Now, let us run the same program to achieve a higher accuracy of 0.001 starting from same initial search space and print only final results. The input dialogue on the screen would as shown below:

```
>>intervalhalf
Enter lower bound u = 0
Enter upper bound v = 5
Enter desired accuracy e = 0.001
```

Suppressing the iteration-wise detailed results, only the end results of the program are shown low.

```
Minimum point lies in interval (2.000, 2.000)
No. of iterations = 13
No. of function evaluations = 27
->
```

It is observed that in every iteration 50% of interval is eliminated in two function evaluations. ius, this method brings about 25% elimination per function evaluation. There are other methods r minimizing the single variable function with higher capability of region elimination per nction evaluation and these are discussed below.

2.2 Fibonacci Search Method

ke the previous method, this is also based on the principle of region elimination. The extent of mination depends on the ratio of two pre-specified numbers of Fibonacci series i.e., 1, 1, 2, 3, 8, 13 and so on. A number in this series is the sum of its two immediate preceding numbers. ius, it satisfies the relationship $F_n = F_{n-1} + F_{n-2}$ for $n = 2, 3, \ldots$.The series starts with the sumption of $F_0 = 1$ and $F_1 = 1$. The property of Fibonacci numbers is so exploited in this gorithm that only one function evaluation is required in each iteration. At any stage when two ints are compared for region elimination, one point is always same as either of the previous ints and the other point is the new one. To achieve this, at iteration k two intermediate points and x_2 are selected at an equal distance of d_k from either end of the search space. Based on the ative values of the objective function at the two intermediate points, a portion of the search ace is eliminated as per region elimination rule discussed before and let the remaining search ace be r_k. The width of initial search space is $w = v - u$ where u and v are lower and upper unds. If $d_k = (F_{n-k+1}/F_{n+1})\,w$ and $r_k = (F_{n-k+2}/F_{n+1})\,w$ where n is maximum number of function aluations to be done, it can be shown that $r_k - d_k = (F_{n-k+2} - F_{n-k+1})\,w/F_{n+1} = (F_{n-k}/F_{n+1})\,w$ as $_{-k} + F_{n-k+1} = F_{n-k+2}$ by definition of Fibonacci series. As $d_{k+1} = (F_{n-(k+1)+1}/F_{n+1})\,w = (F_{n-k}/F_{n+1})\,w$, it can be written as $r_k - d_k = d_{k+1}$. Depending upon the values of objective function f at two intermediate points x_1 and x_2, either the left portion, i.e., (u, x_1) or the right portion i.e., $x_2, v)$ is eliminated. Thus, the eliminated portion has a width of d_k and the remaining portion will $r_k = w - d_k$. Since $r_k - d_k = d_{k+1}$, one of the two points used in iteration k is also considered the next iteration $k + 1$. The algorithm for the above procedure is given below.

ep 1: Assume initial search space (u, v) and maximum number of function evaluations n
Find width of search space $w = v - u$
Set iteration counter $k = 2$

ep 2: Calculate $d_k = (F_{n-k+1}/F_{n+1})\,w$
Set $x_1 = u + d_k$ and $x_2 = v - d_k$

ep 3: Evaluate $f(x_1)$ or $f(x_2)$ whichever was not evaluated earlier
Eliminate a portion of region using region elimination rule
Set $u = x_1$ or $v = x_2$ as the case may be

Step 4: If $k = n$

print $x^* \in (u, v)$ and stop

Else set $k = k + 1$ and go to Step 2.

The corresponding MATLAB code for minimizing the function is outlined below. It uses function file fibonacci.m to generate Fibonacci numbers.

```
%minimize an unconstrained single variable function  using Fibonacci Sear
%Method
%file name: fibo.m
%uses function file :f.m and fibonacci.m
%%%%%%%%%%
clc;
clear all;
global NF; % number of function evaluations NF as global variable
NF = 0;
k = 1;
i = 2;
u = input('Enter lower bound u = ');
v = input('Enter upper bound v = ');
nf = input('Max. no. of function evaluation nf = ');
w = v-u;     % intitial length of search space
fb = fibonacci(nf+1); %generate Fibonacci numbers
d = fb(nf-i+1)/fb(nf+1)*w;
x1 = u + d; x2 = v - d; % two intermediate points
f1 = f(x1); f2 = f(x2); %evaluate functions at x1 and x2
fo = fopen('fibo_out.txt', 'w+'); %opens a file for writing output
fprintf(fo,'  k Lower,u\t x1[f(x1)]\t\t x2[f(x2)]\t\t Upper,vwidth,w  NF\n'
fprintf(fo,'%3d\t   %.3f\t   %.3f[%.3f]\t   %.3f[%.3f]\t   %.3f\t   %.3f
%3d\n',k,u,x1,f1,x2,f2,v,v-u,NF);
if f1 > f2 %Case 1:delete left of x1
          u = x1;
else      v = x2; %Case 2:delete right of x2
end
while(i<nf) % terminate when max. no. of function evaluations is reache
i = i+1;
d = fb(nf-i+1)/fb(nf+1)*w;
%update x1 & x2
if f1 > f2 %Case 1: x2 is new point
      x1 = x2; f1 = f2;   x2 = v-d; f2 = f(x2);
else %Case 2: x1 is new point
      x2 = x1; f2 = f1;   x1= u+d; f1 = f(x1);
end
```

```
sp(d)
sp(x2)
= k+1;
rintf(fo,'%3d\t  %.3f\t  %.3f[%.3f]\t  %.3f[%.3f]\t  %.3f\t  %.3f\t
d\n',k,u,x1,f1,x2,f2,v,v-u,NF);
 f1 > f2 %Case 1:delete left of x1
       u = x1;
se %Case 2:delete right of x2
       v = x2;
d
d
rintf(fo,'Minimum point lies in interval (%.3f, %.3f)\n',u,v);
rintf(fo,'No. of iterations = %d \n',k);
rintf(fo,'No. of function evaluations = %d \n',NF);
lose(fo);
```

e control of function file fibonacci.m for generating first n Fibonacci numbers is given below.

```
nction f = fibonacci(n)
enerates the first n Fibonacci numbers.
le name: Fibonacci.m
%%%%%%%
= zeros(n,1);
1) = 1;
2) = 2;
r k = 3:n
k) = f(k-1) + f(k-2);
d
```

When the above program is run, the following messages appear on the screen. On feeding input data as shown underlined below, the results are printed in a file called 'fibo_out.txt'.

```
>fibo
nter lower bound u = 0
nter upper bound v = 5
lax. no. of function evaluation nf = 5
 1.1538
 1.9231
 0.7692
 2.3077
 0.3846
 1.9231
>
```

The content of the output file 'fibo_out.txt' is given below.

```
 k    Lower,u      x1[f(x1)]        x2[f(x2)]      Upper,v   width,w    NF
 1     0.000     1.923[12.018]   3.077[14.667]     5.000     5.000      2
 2     0.000     1.154[15.198]   1.923[12.018]     3.077     3.077      3
 3     1.154     1.923[12.018]   2.308[12.259]     3.077     1.923      4
 4     1.154     1.538[12.767]   1.923[12.018]     2.308     1.154      5
Minimum point lies in interval (1.538, 2.308)
No. of iterations = 4
No. of function evaluations = 5
```

After 5 function evaluations, the result shows that the minimum point lies in the interv (1.538, 2.308). This is completed in 4 iterations. Each row shows the iteration number, lower a upper bounds of search space u and v, the width of search space $w = v - u$, the two intermedia points $x1$ and $x2$, values of objective function at these two intermediate points $f(x1)$ and $f(x$ and the cumulative number of function evaluations done so far NF.

Let us now discuss how the results are calculated. Initially, $u = 0$, $v = 5$ and $i = 2$. Thu width $w = v - u = 5$. Maximum number of function evaluations $nf = 5$. First $nf + 1$ number Fibonaaci numbers are generated and stored as $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$, $F_6 =$ by calling the function fibonaaci.m. The distance d is calculated as $d = (F_{nf-i+1}/F_{nf+1})w = (F_6)w = (5/13)5 = 1.923$. Thus, the two intermediate points are $x1 = u + d = 1.923$ and $x2 = v - = 5 - 1.923 = 3.077$. The corresponding function values are $f(x1) = 12.018$ and $f(x2) = 14.66$ Since, $f(x2) > f(x1)$, the right side region $(x2, v)$ i.e., (3.077, 5) is eliminated. So, now $u = 0$ a $v = 3.077$. Since $i = 2 \neq 5$, the value of i is increased by 1 to repeat the cycle. This complet one iteration of this algorithm.

For better accuracy of result, usually a high value for maximum number of function evaluatio nf is taken. If the same program is run with $nf = 1000$, the following result is obtained. T iteration-wise results are suppressed and only the final result is shown.

```
>>fibo
Enter lower bound u = 0
Enter upper bound v = 5
Max. no. of function evaluation nf = 1000
   1.1803
   1.9098
   0.7295
   ...
   ...
   8.7876e-209
   2.0000
   4.3938e-209
   2.0000
```

```
inimum point lies in interval (2.000, 2.000)
o. of iterations = 999
o. of function evaluations = 1000
```

2.3 Golden Section Method

ere are two shortcomings of Fibonacci method. Fibonacci numbers are to be calculated and red. Secondly, the proportion of eliminated region is not the same in every iteration. These o problems are overcome in the golden section method. It also needs only one function aluation per iteration. In this method, the search space (u, v) is first linearly mapped to an erval having unit width, i.e., (0, 1). Then two intermediate points are selected in the search ace which are equidistant from either end. If this distance is g times the width of search space any iteration, then the eliminated region would be $(1 - g)$ times the width of search space the previous iteration. This is possible for only one particular value of g, i.e., when $1 - g =$ Solving this equation, the value of g is obtained as 0.618. This is called the golden number. ke Fibonacci method, in this method also one of the two intermediate points is always the me as that considered in the previous iteration. The step-by-step algorithm of this method is esented below.

ep 1: Assume lower and upper bounds of initial search space as u and v
Also assume a small number e for desired accuracy
Normalize decision variable x by $z = (x - u)/(v - u)$ so that $u_z = 0$, $v_z = 1$, $w_z = v_z - u_z = 1$
Set iteration counter $k = 1$

ep 2: Set $z_1 = u_z + 0.618*w_z$ and $z_2 = v_z - 0.618*w_z$
Evaluate $f(z_1)$ or $f(z_2)$ whichever was not evaluated earlier
Eliminate a portion of region using region elimination rule
Update bounds $u_z = z_1$ or $v_z = z_2$ as the case may be
Update $w_z = v_z - u_z$

ep 3: If $|w_z| < e$
print $x^* \in (u + u_z(v - u), u + v_z(v - u))$ and stop
Else set $k = k + 1$ and go to Step 2.

The MATLAB code for the above algorithm is outlined below.

```
inimize an unconstrained single variable function using Golden Section
Method
le name: golden.m
ses function file :f.m
%%%%%%%%
c;
ear all;
obal NF; % number of function evaluations NF as global variable
 = 0;
```

```
k = 1; % iteration counter
u = input('Enter lower bound u = ');
v = input('Enter upper bound v = ');
e = input('Enter desired accuracy e = ');
w = v-u;
uz = 0; vz = 1; wz = 1;
gwz = 0.618*wz;
z1 = uz+gwz; z2 = vz-gwz; % two intermediate points
x1 = u+w*z1; x2 = u+w*z2;
f1 = f(x1); f2 = f(x2);
fo = fopen('golden_out.txt', 'w+');
fprintf(fo,'   k Lower,uz\t z1[f(z1)]\t\t z2[f(z2)]\t Upper,vzwidth
NF\n');
while(1)
fprintf(fo,'%3d\t %.3f\t %.3f[%.3f]\t %.3f[%.3f]\t %.3f\t %.3f\t %3d
',k,uz,z1,f1,z2,f2,vz,wz,NF);
if f2 > f1
        uz = z2; %delete left of z2
elsevz = z1; %delete right of z1
end
wz = vz-uz;
if abs(wz)< e
        break;
end
gwz = 0.618*wz;
if f2 > f1
        z2 = z1; f2 = f1;
        z1 = uz+gwz; x1 = u+w*z1; f1 = f(x1);
else
        z1 = z2; f1 = f2;
        z2 = vz-gwz; x2 = u+w*z2; f2 = f(x2);
end
    k = k+1;
end
fprintf(fo,'Minimum point z* lies in interval (%.3f, %.3f)\n',uz,vz)
fprintf(fo,'Minimum point x* lies in interval (%.3f, %.3f
n',u+w*uz,u+w*vz);
fprintf(fo,'No. of iterations = %d \n',k);
fprintf(fo,'No. of function evaluations = %d \n',NF);
fclose(fo);
```

When the above program is executed and the following messages appear on screen.

```
>>golden
Enter lower bound u = 0
Enter upper bound v = 5
Enter desired accuracy e = 0.1
>>
```

The contents of output file 'golden_out.txt' is given below.

k	Lower,uz	z1[f(z1)]	z2[f(z2)]	Upper,vz	width,w	NF
1	0.000	0.618[14.726]	0.382[12.025]	1.000	1.000	2
2	0.000	0.382[12.025]	0.236[14.948]	0.618	0.618	3
3	0.236	0.472[12.350]	0.382[12.025]	0.618	0.382	4
4	0.236	0.382[12.025]	0.326[12.470]	0.472	0.236	5
5	0.326	0.416[12.020]	0.382[12.025]	0.472	0.146	6

```
Minimum point z* lies in interval (0.382, 0.472)
Minimum point x* lies in interval (1.910, 2.361)
No. of iterations = 5
No. of function evaluations = 6
```

When the input data are fed, they are stored as $u = 0$, $v = 5$ and $e = 0.1$. So, width of search space is $w = v - u = 5$. On mapping to a new variable z, the search space is converted to (0, 1). Thus, bounds are $uz = 0$, $vz = 1$ and width $wz = 1 - 0 = 1$. Then, the two intermediate points are at $z1 = 0 + 0.618*1 = 0.618$ and $z2 = 1 - (0.618)*1 = 0.382$. These two points in terms of x variable are $x1 = u + w*z1 = 0 + 5*0.618 = 3.09$ and $x2 = u + w*z2 = 0 + 5*0.382 = 1.91$. The values of objective function at these two points are $f(x1) = (3.09)^2 + 16/3.09 = 14.726$. Similarly, $f(x2) = (1.91)^2 + 16/1.91 = 12.025$. Since $f(0.618) > f(0.382)$, the right portion, i.e., (0.618,1) is eliminated and the upper bound is updated as $vz = 0.618$ and the corresponding width $wz = 0.618-0 = 0.618$. Since $|wz|$ is not less than desired accuracy $e = 0.1$, k is increased by 1 and move back to repeat the cycle. This completes one iteration of the golden section algorithm. In this case, the program stops after 5 iterations involving $5 + 1 = 6$ function evaluations and gives final result that minimum point of x lies in the interval (1.910, 2.361). It can be observed that only one new function evaluation is required in each iteration except the first one where there are two evaluations. For example, in 2nd iteration 0.382 was already considered in previous iteration and 0.236 is the only new point where function is to be evaluated.

For better accuracy of result, comparatively smaller value of e is taken. Taking $e = 0.001$, if the same program is executed with the suppression of intermediate results, the following final result is obtained.

```
>>golden
Enter lower bound u = 0
Enter upper bound v = 5
Enter desired accuracy e = 0.001
>>
```

The output is written in the file 'golden_out.txt' is as shown below.

```
Minimum point z* lies in interval (0.399, 0.400)
Minimum point x* lies in interval (1.997, 2.001)
No. of iterations = 15
No. of function evaluations = 16
```

8.3 GRADIENT-BASED METHODS

In all the algorithms discussed so far in this chapter, they deal with values of objective functio for finding the local optimal solution. None of these requires derivative of the objective functio This section explains few algorithms for minimizing an objective function when there is n constraint and all these algorithms need derivative of the objective function. Some need onl first derivative and the rest need both first and second derivatives. There are many objectiv functions which are too complex to find their derivatives. In these cases, the algorithms to b discussed hereafter are not useful. In spite of this difficulty, these algorithms are the most efficie among all. Therefore, these algorithms are always recommended for minimizing a function it is possible to find out its derivatives. All these algorithms terminate when gradient, i.e., firs derivative of the objective function approaches zero because at local optimum point the tange to its graphical plot is horizontal.

8.3.1 Newton-Raphson Method

This method makes a linear approximation to the first derivative of the objective function at point using Taylor's series approximation. This expression is equated to zero to determine th next guess. If the current point is x_k, the point in the next iteration $k + 1$ would be $x_{k+1} = x_k$ $f'(x_k)/f''(x_k)$. The algorithm for this method is outlined below.

Step 1: Assume starting solution x_1 and a small number e for desired accuracy
Set iteration counter $k = 1$
Compute $f'(x_k)$

Step 2: Compute $f''(x_k)$

Step 3: Find next solution $x_{k+1} = x_k - f'(x_k)/f''(x_k)$
Compute $f'(x_{k+1})$

tep 4: If $|f'(x_{k+1})| < e$

print $x^* = x_{k+1}$ and stop

Else set $k = k + 1$ and go to Step 2.

There are various ways to find out derivatives in MATLAB such as symbolic toolbox, diff() nction etc. However, we present here two subroutines, namely, der1.m and der2.m for finding rst and second derivatives of a function numerically. They use the following two expressions ased on the central difference method for finding the first and second derivatives

$$f'(x_k) = \frac{f(x_k + \Delta x_k) - f(x_k - \Delta x_k)}{2\Delta x_k}$$

$$f''(x_k) = \frac{f(x_k + \Delta x_k) + 2f(x_k) + f(x_k - \Delta x_k) - 2f(x_k)}{(\Delta x_k)^2}$$

The parameter Δx_k usually takes a very small value for better accuracy of derivative result. s a standard practice, its value is taken as 1% of x_k as given below:

$$\Delta x_k = \begin{cases} 0.01|x_k| & \text{if } |x_k| > 0.01 \\ 0.01 & \text{otherwise} \end{cases}$$

It may be observed that for calculation of derivatives using the central difference method, e first derivative needs two function evaluations whereas the second derivative needs three. The contents of the two function fields der1.m and der2.m are given below.

```
nction y = der1(x)
finds approximate value of first derivative of any function f(x)
uses central difference method
file name: der1.m
uses function file: f.m
%%%%%%%%
sx = abs(x);
absx> 0.01
   dx = 0.01*absx;
se dx = 0.01;
d
= (f(x+dx)-f(x-dx))./(2*dx);
turn

nction y = der2(x)
finds approximate value of 2nd derivative of any function f(x)
uses central difference method
```

```
%file name: der2.m
% uses function file: f.m
%%%%%%%%%%
absx = abs(x);
if absx> 0.01
dx = 0.01*absx;
else
    dx = 0.01;
end
y = (f(x+dx)-2*f(x) + f(x-dx))./(dx.*dx);
return
```

The MATLAB code for Newton-Raphson method is presented below.

```
%minimize an unconstrained single variable function using Newto
%Raphson Method
%uses approximate values for 1st and 2nd derivative with der1.m ar
%der2.m
%file name: newton.m
%uses function files: f.m, der1.m, der2.m
%%%%%%%%%%
clc;
clear all;
k = 1; % iteration counter
x = input('Enter initial value x0 = ');
e = input('Enter desired accuracy e = ');
d1f = der1(x);
fprintf('  k\t\t x\t\t f1dash(x)\t f2dash(x)\n');
while abs(d1f) > e
    d2f = der2(x);
fprintf('%3d\t %7.3f\t %7.3f\t %7.3f\n',k,x,d1f,d2f);
    x = x - d1f/d2f;
    d1f = der1(x);
    k = k+1;
end
fprintf('%3d\t %7.3f\t %7.3f\n',k,x,d1f);
fprintf('Minimum point x* = %.3f\n',x);
fprintf('No. of iterations = %d \n',k);
```

On executing the above program to minimize the same function *f* the following messag appear for data input.

```
>newton
nter initial value x0 = 1
nter desired accuracy e = 0.1
```

On feeding the required data as shown above underlined, the following output is obtained. It splays the values of x, $f'(x)$ (i.e. f1dash(x)) and $f''(x)$ (i.e. f2dash(x)) at every iteration k as own below.

k	x	$f1\text{dash}(x)$	$f2\text{dash}(x)$
1	1.000	−14.002	34.003
2	1.412	−5.205	13.374
3	1.801	−1.332	7.479
4	1.979	−0.128	6.129
5	2.000	−0.001	

```
Minimum point x* = 2.000
No. of iterations = 5
>
```

The minimum point is found to be at $x = 2.0$ within 5 iterations. The program stops here at x 2.0 as $|f'(x)| = 0.001 < 0.1$. Let us now do hand calculation for one iteration to have a feeling out how the algorithm proceeds. As per data input, $x_1 = 1$ and $e = 0.1$. We set $k = 1$. The first rivative at $x_1 = 1$ is calculated as shown below:

$\Delta x_1 = 0.01*|x_1| = 0.01$. Function values are $f(x_1 + \Delta x_1) = f(1.01) = (1.01)^2 + 16/1.01 = 16.862$, d $f(x_1 - \Delta x_1) = f(0.99) = (0.99)^2 + 16/0.99 = 17.142$. So, first derivative at $x_1 = 1$ is

$$f'(1) = \frac{f(1.01) - f(0.99)}{2 \times 0.01} = \frac{16.862 - 17.142}{0.02} = -14.002$$

The value of function f at $x_1 = 1$ is $f(x_1) = f(1) = 1^2 + 16/1 = 17$. This is required in calculating cond derivative at $x_1 = 1$ as shown below:

$$f''(1) = \frac{f(1.01) - 2 \times f(1) + f(0.99)}{0.01^2} = \frac{16.862 - 2 \times 17 + 17.142}{0.0001} = 34.003$$

Now, $x_2 = x_1 - f'(x_1)/f''(x_1) = 1 - (-14.002)/34.003 = 1.412$

For finding the first derivative at $x_2 = 1.412$, $\Delta x_2 = 0.01*|x_2| = 0.01*1.412 = 0.014$

$$f'(1.412) = \frac{f(1.426) - f(1.398)}{2 \times 0.01} = \frac{13.254 - 13.401}{0.02} = -5.205$$

Since, $|f'(1.412)| = 5.205 > e = 0.1$, $k = k + 1 = 1 + 1 = 2$ and the same cycle is repeated till $(x_k)| < e$.

If the same program is executed for a higher accuracy say $e = 0.001$ with same initial soluti at x_1=1, the following result is displayed.

```
>>newton
Enter initial value x0 = 1
Enter desired accuracy e = 0.001
```

k	x	$f1dash(x)$	$f2dash(x)$
1	1.000	−14.002	34.003
2	1.412	−5.205	13.374
3	1.801	−1.332	7.479
4	1.979	−0.128	6.129
5	2.000	−0.001	6.001
6	2.000	−0.000	

```
Minimum point x* = 2.000
No. of iterations = 6
>>
```

It is observed that this method is capable of finding minimum point within a few iteratio even for a very high accuracy. Therefore, if it is possible to find derivatives of objective functic gradient based algorithms are always preferred for finding local optimal solution.

8.3.2 Bisection Method

It is also one type of gradient-based algorithm for finding local optimum point. Newton-Raphs method needs both first and second derivatives of the objective function. The second derivati requires as many as three function evaluations. The bisection method has one merit that it do not require second derivative. It uses only first derivative. It is based on the principle of regi elimination. So, it can search the minimum point within a given search space if it is guarante that minimum point lies in this space. Therefore, it needs the values of lower and upper boun (i.e., u and v) of the initial search space but unlike other algorithms based on region eliminatic this uses the sign of the first derivative of the objective function to eliminate a portion search space. At the minimum point, the first derivative is zero and on either side of this poi the sign of this derivative is opposite. Thus, the minimum point lies in the interval (u, v) $f'(u) > 0$ and $f'(v) > 0$. The sign of the first derivative at the mid-point of the search space used to eliminate half of the search space. If it is negative, the left half is eliminated and if is positive, the right half is eliminated. If it is zero, the midpoint is the minimum point. T algorithm for this procedure is given below.

Step 1: Assume any two points u and v such that $f'(u) < 0$ and $f'(v) > 0$
Also assume a small number e for the desired accuracy
Set $x_1 = u$ and $x_2 = v$

Step 2: Find mid-point $z = (x_1 + x_2)/2$
Compute $f'(z)$

Step 3: If $|f'(z)| < e$
print $x^* = z$ and stop
Else if $f'(z) < 0$, $x_1 = z$ and go to Step 2
Else if $f'(z) > 0$, $x_2 = z$ and go to Step 2

The MATLAB code for the above algorithm is given below.

```
%minimize an unconstrained single variable function using Bisection
%method
%uses approximate value of 1st derivative with der1.m
%file name: bisection.m
%uses function files: f.m and der1.m
%%%%%%%%%%
clc;
clear all;
e = input('Enter desired accuracy e = ');
while(1)
    u = input('Enter lower bound u = ');
    v = input('Enter upper bound v = ');
    f1u=der1(u); % first derivative of f at u;
    f1v=der1(v); % first derivative of f at v;
fprintf('1st derivative at %.3f = %.3f\n',u,f1u);
fprintf('1st derivative at %.3f = %.3f\n',v,f1v);
if f1u < 0 && f1v > 0
break;
end
disp('Sorry, try with other two values');
end
x1=u; x2=v;
k=1; % iteration counter
fprintf('  k\t\t x1\t\t\t x2\t\t\t mid\t\t f1dash(mid)\n');
while(1)
mid = (x1+x2)/2;
f1m = der1(mid);
abf1 = abs(f1m);
```

Contd.

Contd.

```
fprintf('%3d\t %7.3f\t %7.3f\t %7.3f\t   %7.3f \n',k,x1,x2,mid,f1m);
if abf1 <= e
     break;
else if f1m < 0
     x1 = mid; % delete left of mid-point
else x2 = mid; % delete right of mid-point
end
     k=k+1;
end
end
fprintf('Minimum point x* = %.3f\n',mid);
fprintf('No. of iterations = %d \n',k);
```

The above program is executed with the following input data as shown underlined below t minimize the same unconstrained objective function $f(x) = x^2 + 16/x$.

```
>>bisection
Enter desired accuracy e = 0.1
Enter lower bound u = 1
Enter upper bound v = 10
1st derivative at 1.000 = -14.002
1st derivative at 10.000 = 19.840
```

k	*x*1	*x*2	mid	*f*1dash (mid)
1	1.000	10.000	5.500	10.471
2	1.000	5.500	3.250	4.985
3	1.000	3.250	2.125	0.706
4	1.000	2.125	1.563	-3.429
5	1.563	2.125	1.844	-1.020
6	1.844	2.125	1.984	-0.095

```
Minimum point x* = 1.984
No. of iterations = 6
>>
```

When the above program is run, the initial two points selected are say $u = 1$ and $v = 10$. The firs derivatives at these two points are $f'(1) = -14.002$ and $f'(10) = 19.840$. Since they are of opposite signs, these two points are accepted and the algorithm proceeds. Otherwise, it will prompt fo selecting another two bounds. For example, if the data input for the program are $u = 5$ and $v = 10$, the following messages are shown.

```
>>bisection
Enter desired accuracy e = 0.1
Enter lower bound u = 5
Enter upper bound v = 10
1st derivative at 5.000 = 9.360
1st derivative at 10.000 = 19.840
Sorry, try with other two values
Enter lower bound u =
```

Continuing with the data input $u = 1$ and $v = 10$, in the next step $x1 = u = 1$ and $x2 = v = 10$. Then, the mid-point is found to be $z = \text{mid} = (x_1 + x_2)/2 = (1 + 10)/2 = 5.5$. Then, the first derivative is found to be $f'(5.5) = 10.471$. As $|f'(5.5)| = 10.471$ is not less than 0.1, the program is not terminated. Since the value of first derivative $f'(5.5)$ is positive, the point 5.5 is on the right side of the minimum point. Hence, it is clear that minimum point is no where in the right side of 5.5. So, the interval (5.5,10) is eliminated, and upper bound is set at $v = 5.5$ and lower bound is kept unchanged. This completes one cycle. After 6 iterations, the termination criteria is satisfied, i.e., $|f'(1.984)| = 0.095 < e = 0.1$. The minimum point is shown to exist at $x^* = 1.984$ before the program terminates.

If a higher accuracy is desired for the solution, say $e = 0.001$, the result is obtained in 15 iterations as shown below.

```
>>bisection
Enter desired accuracy e = 0.001
Enter lower bound u = 1
Enter upper bound v = 10
1st derivative at 1.000 = -14.002
1st derivative at 10.000 = 19.840
```

k	$x1$	$x2$	mid	$f1$dash(mid)
1	1.000	10.000	5.500	10.471
2	1.000	5.500	3.250	4.985
3	1.000	3.250	2.125	0.706
4	1.000	2.125	1.563	−3.429
5	1.563	2.125	1.844	−1.020
6	1.844	2.125	1.984	−0.095
7	1.984	2.125	2.055	0.319
8	1.984	2.055	2.020	0.116
9	1.984	2.020	2.002	0.011
10	1.984	2.002	1.993	−0.042
11	1.993	2.002	1.998	−0.015
12	1.998	2.002	2.000	−0.002
13	2.000	2.002	2.001	0.005

Contd.

Contd.

14	2.000	2.001	2.000	0.001
15	2.000	2.000	2.000	−0.000

Minimum point $x^* = 2.000$
No. of iterations = 15
>>

This shows that minimum point lies at $x = 2.000$. In this method, only the first derivative evaluated and that too only at one new point in each iteration. Hence, there are only two functi evaluations per iteration. Here, exactly half of the search space is eliminated in each iterati and it is done based on only sign of the first derivative.

8.3.3 Secant Method

Like bisection method, the secant method is also based on gradient and region eliminati principles to search for the optimal point. Another similarity is that it uses only the first derivativ However, it uses both the information of gradient i.e., its magnitude and sign to generate ne point whereas the bisection method uses only the sign of the gradient. In this method, th gradient is assumed to vary linearly between the two boundary points of the search space. Th two boundary points are so selected that the signs of the first derivative at these two points a opposite to each other. This implies that there exists a point in between these two boundary poin where the sign of first derivative changes, i.e., the value of the first derivative is zero. Assumin linear relationship over the interval between two points x_1 and x_2, the point z with zero derivativ in this interval can be found using the following expression.

$$z = x_2 - f'(x_2)\left(\frac{x_2 - x_1}{f'(x_2) - f'(x_1)}\right)$$

Unlike the bisection method, the amount of region elimination per iteration is not fixed. may be more than 50% in some iteration depending on the magnitude of the first derivativ at the two end points x_1 and x_2. Similarly, it may also be less than 50% in some iteration. Th algorithm for this method is same as that of the bisection method except in Step 2 for findin the new point as given below.

Step 2: Find $z = x_2 - f'(x_2)\left(\frac{x_2 - x_1}{f'(x_2) - f'(x_1)}\right)$

Compute $f'(z)$

The MATLAB code for the above algorithm is given below.

```
%minimize an unconstrained single variable function using Secant metho
%uses approximate value of 1st derivative with der1.m
%file name: secant.m
%uses function files: f.m and der1.m
```

```
%%%%%%%%%
lc;
lear all;
 = input('Enter desired accuracy e = ');
ile(1)
    u = input('Enter lower bound u = ');
    v = input('Enter upper bound v = ');
    f1u=der1(u); % first derivative of f at u;
    f1v=der1(v); % first derivative of f at v;
rintf('1st derivative at %.3f = %.3f\n',u,f1u);
rintf('1st derivative at %.3f = %.3f\n',v,f1v);
 f1u < 0 && f1v > 0
eak;
d
sp('Sorry, try with other two values');
d
=u; x2=v;
1; % iteration counter
rintf('  k\t\t x1\t\t\t x2\t\t\t z\t\t f1dash(z)\n');
ile(1)
    f1d1 = der1(x1);f1d2 = der1(x2);
    z = x2-f1d2*(x2-x1)/(f1d2-f1d1); % new point z
    f1z = der1(z); % first derivative at z
    abf1 = abs(f1z);
rintf('%3d\t %7.3f\t %7.3f\t %7.3f\t  %7.3f \n',k,x1,x2,z,f1z);
 abf1 <= e
eak;
se if f1z < 0
 = z; % delete left of z
se x2 = z; % delete right of z
d
   k=k+1;
d
d
rintf('Minimum point x* = %.3f\n',z);
rintf('No. of iterations = %d \n',k);
```

The program is then executed to minimize the same unconstrained objective function $f(x) =$ $+ 16/x$. Upon feeding the following input data (shown underlined), the iteration-wise results then printed as shown below.

```
>>secant
Enter desired accuracy e = 0.1
Enter lower bound u = 1
Enter upper bound v = 10
1st derivative at 1.000 = –14.002
1st derivative at 10.000 = 19.840
```

k	x1	x2	z	fldash(z)
1	1.000	10.000	4.724	8.730
2	1.000	4.724	3.294	5.112
3	1.000	3.294	2.680	3.133
4	1.000	2.680	2.373	1.904
5	1.000	2.373	2.209	1.137
6	1.000	2.209	2.118	0.668
7	1.000	2.118	2.067	0.388
8	1.000	2.067	2.038	0.224
9	1.000	2.038	2.022	0.129
10	1.000	2.022	2.012	0.074

```
Minimum point x* = 2.012
No. of iterations = 10
>>
```

Initially, the first derivatives at the two end points $u = 1$ and $v = 10$ are calculated, i.e., f' $= -14.002$ and $f'(10) = 19.840$. As both are of opposite signs, both these points are accepted proceed with further steps. Otherwise, it asks for guessing another two boundary points as simi to the bisection method. Next, it was set as $x1 = u = 1$ and $x2 = v = 10$. The first derivatives objective function $f(x)$ at the two points $x1$ and $x2$ are $f'(10) = -14.002$ and 19.840 as calculat earlier. Thus, the next intermediate point is calculated as

$$z = x_2 - f'(x_2)\left(\frac{x_2 - x_1}{f'(x_2) - f'(x_1)}\right) = 10 - 19.840 \times \frac{10-1}{19.840 - (-14.002)} = 4.724$$

For finding first derivative at $z = 4.724$, $\Delta z = 0.01*|z| = 0.047$.

The two function values required to find first derivative are calculated below:

$$f(z + \Delta z) = f(4.724 + 0.047) = f(4.771) = (4.771)^2 + 16/4.771 = 26.118, \quad \text{and}$$

$$f(z - \Delta z) = f(4.724 - 0.047) = f(4.677) = (4.677)^2 + 16/04.677 = 25.293.$$

So, first derivative at $z = 4.724$ is

$$f'(4.724) = \frac{f(4.771) - f(4.677)}{2 \times 0.047} = \frac{26.118 - 25.293}{0.094} = 8.730, \text{ i.e., not less than } e = 0.1.$$

As $f'|z = (4.724)| = 8.730 > e = 0.1$ the program is not terminated and region elimination is lone. Since, $f'(z = 4.724)$ is positive, z lies on right side of minimum point. So, all the portion f search interval on right side of $z = 4.724$ is eliminated. Thus, right extreme point $x2 = 10$ is pdated as $x2 = z = 4.724$ keeping the left extreme point $x1 = 1$ unchanged. Thus, the search nterval (1,10) is reduced to (1,4.724). This completes one iteration. After 10 such iterations, the ermination criteria is satisfied space i.e. $|f'(2.012)| = 0.074 < e = 0.1$. The minimum point is hown to exist at $x^* = 2.012$.

If the value of desired accuracy e is reduced to 0.001, the following result is obtained.

```
>>secant
Enter desired accuracy e = 0.001
Enter lower bound u = 1
Enter upper bound v = 10
1st derivative at 1.000 = –14.002
1st derivative at 10.000 = 19.840
```

k	$x1$	$x2$	z	$f1\text{dash}(z)$
1	1.000	10.000	4.724	8.730
2	1.000	4.724	3.294	5.112
3	1.000	3.294	2.680	3.133
4	1.000	2.680	2.373	1.904
5	1.000	2.373	2.209	1.137
6	1.000	2.209	2.118	0.668
7	1.000	2.118	2.067	0.388
8	1.000	2.067	2.038	0.224
9	1.000	2.038	2.022	0.129
10	1.000	2.022	2.012	0.074
11	1.000	2.012	2.007	0.042
12	1.000	2.007	2.004	0.024
13	1.000	2.004	2.002	0.014
14	1.000	2.002	2.001	0.008
15	1.000	2.001	2.001	0.005
16	1.000	2.001	2.000	0.003
17	1.000	2.000	2.000	0.001
18	1.000	2.000	2.000	0.001

```
Minimum point x* = 2.000
No. of iterations = 18
>>
```

In the above example it is observed that as the level of desired accuracy is raised from 0. to 0.001 this method takes only 8 more iterations to locate the minimum point. Further, th percentage region elimination in the first iteration is $\frac{(10-1)-(4.724-1)}{10-1} \times 100 = 58.62\%$ i.e more compared to 50% in bisection method. This falls down to $\frac{(4.724-1)-(3.294-1)}{4.724-1} \times 100$ = 38.40% in the second iteration, i.e., less than 50%. Thus, this percentage is not constant, rathe it reduces gradually in subsequent iterations.

9

HAPTER

Statistical Functions

*he term **firewall** actually means a wall that confines fire within a building. It is referred to fireproof walls de of steel or bricks that prevent the spread of fire. A computer firewall does not involve fire protection in the e sense but serves a similar purpose. Firewall technology was developed in the 1980s. Before this the routers th built-in firewall were used for network security. Routers can be configured to control traffic from certain addresses or block requests based on certain criteria. A computer firewall is designed to prevent unauthorized ess to a private network. It protects unauthorized internet users from accessing a private network, like the ranets, that is connected to the internet. Those messages entering the intranet that do not pass through the cified security conditions are blocked. A firewall is a software or hardware based network security system t controls the incoming and outgoing network traffic by analyzing the data packets and determines whether y should be allowed or not. A personal computer always contains a software based firewall which protects inst threats of public internet. Firewalls can be both software-based and hardware-based."*

1 STATISTICS

tistics is a type of mathematical analysis which involves the use of quantified representations, dels and summaries for a given set of empirical data or real-world observations. Statistics olves the study of data. It deals with how to collect, summarize and present it. Statistical alysis involves the collection of data and analyzing them and summarizing them into a merical form. Few statistical techniques are regression analysis, mean, kurtosis, skewness, alysis of variance and variance. Several economic, social, political and military decisions are sed on statistical techniques. If the data set is collected from a large population then analysts draw inferences form the data and predict the probability of occurrence of phenomena from se data. Statistics is thus the science of collection, organization, analysis, interpretation and sentation of data.

2 MEAN, MEDIAN, MODE AND RANGE

2.1 Mean

an is the average. Here we add up all the numbers and then divide by the number of numbers. e set of numbers is often a set of results obtained in an experiment, or a survey.

```
n=input('enter the number of elements:')
for i=1:n
    a(i)=input('enter the element:');
end
sum=0;
for i=1:n
    sum=sum+a(i);
end
mean=sum/n
s1=0;
```

enter the number of elements:4

n =

4

enter the element:1

enter the element:2

enter the element:4

enter the element:6

mean =

3.2500

Mean can also be found using the function mean()

```
>> a=[1 2 4 6];
>> mean(a)
```

ans =

3.2500

The mean() function can also be used to find the mean of an array.

```
>> a=[1 2 4 6; 10 20 30 40; 11 14 16 17];
>> mean(a)
```

ans =

7.3333 12.0000 16.6667 21.0000

9.2.2 Median

The median is the middle value in the set of given numbers. In order to find the median all numbers have to be arranged in ascending or descending order. The number in the middle is median.

The function median() gives the median value of an array. The syntaxes for the median function
e,

```
edian(a)
edian(a,d)
```

median(a) returns the median value of the elements along the columns of the matrix a.

```
> a=[1 2 3;4 5 6;7 8 9]
=
 1  2  3
 4  5  6
 7  8  9
> median(a)
s =
 4  5  6
```

If a is a matrix, as in the example given above, median(a) returns a row vector of the median
lues treating the columns of the matrix a as vectors.
median(a,d) returns the median values for elements along the dimension of a specified by
For example, if d=2 in the above problem,

```
 a=[1  2  3;4  5  6;7  8  9]
=
   1  2  3
   4  5  6
   7  8  9
 median(a,2)
s =
   2
   5
   8
```

Whereas d=1 gives the same result as median(a).

```
s =
   4  5  6

 median(a,1)
s =
   4  5  6
```

A program created for sorting could be extended to give us the median of a set of numbe

```
n=input('enter the value of n:')
for i=1:n
    a(i)=input('enter value');
end
i=1;
for j=1:n
for i=1:(n-1)
if a(i)>a(i+1)
        temp=a(i);
        a(i)=a(i+1);
        a(i+1)=temp;
end
end
end
disp('The sorted array is:')
a

m=input('enter the value 1 if OK if NOT OK enter 0:')
for m=0
for j=1:n
for i=1:(n-1)
if a(i)>a(i+1)
        temp=a(i);
        a(i)=a(i+1);
        a(i+1)=temp;
end
end
end
end
disp('The sorted array is:')
a

if (mod(n,2)~=0)
k=ceil(n/2);
disp('the median is :');
a(k)
else
k=(n/2);
disp('the median is:')
(a(k)+a(k+1))/2
end
```

ıter the value of n:5

=

5

ıter value1
ıter value5
ıter value2
ıter value7
ıter value8

ıe sorted array is:

=

1 2 5 7 8

ıter the value 1 if OK if NOT OK enter 0:1

=

1

ıe sorted array is:

=

1 2 5 7 8

e median is :

ıs =

5

2.3 Mode

ıe mode is the value that occurs most frequently. If there is no number that is repeated, then ere is no mode in the given set of numbers. When there is more than one value occurring same ımber of times, `mode` returns the smallest of those values.

The syntax for the mode function is,

ode(a)
ode(a,x)

ode(a) gives the value that occurs most frequently column-wise whereas mode(a,2) gives the ılue that occurs most frequently in the row-wise.

> a=[1 2 2;4 4 6;7 8 9]

=

1 2 2
4 4 6
7 8 9

```
>> mode(a)
ans =
     1   2   2
>> mode(a,2)
ans =
     2
     4
     7
```

9.2.4 Range

The function range(a) is used to find the range of values of the in X. The range is the differen in maximum and minimum of the values in X. For example,

```
>> a=[1 4 8 2 9 10 18 20];
>> range(a)
ans =
     19
>>
```

Note here that the difference of the maximum and minimum is 19.

```
>> max(a)
ans =
     20
>> min(a)
ans =
     1
```

range(a,2) gives the range in the row direction.

```
>> a=[1 2 3 4;5 6 7 8]
a =
     1   2   3   4
     5   6   7   8
>> range(a,2)
ans =
     3
     3
```

In the first row the maximum-minimum is 4 – 1 = 3 and in the second row the maximum-inimum is 8 – 5 = 3.

3 STANDARD DEVIATION, VARIANCE

statistics standard deviation is the measure of deviation or variation of the data from the erage or the mean value. Low value of standard deviation suggests that the data is very close the mean value whereas a high standard deviation indicates a large deviation from the mean lue. Standard deviation is in fact the square root of its variance. The standard deviation has e same units as the data itself. So the standard deviation gives us the idea about how widely read the data is in comparison to the mean value. Its symbol is σ. The variance is measured the average of the squared differences from the man value.

So for a data set $(x_1, x_2, x_3, x_4 \ldots, x_n)$ the variance and standard deviation values are,

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

$$\text{Variance} = \sigma^2 = \frac{1}{(n-1)}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

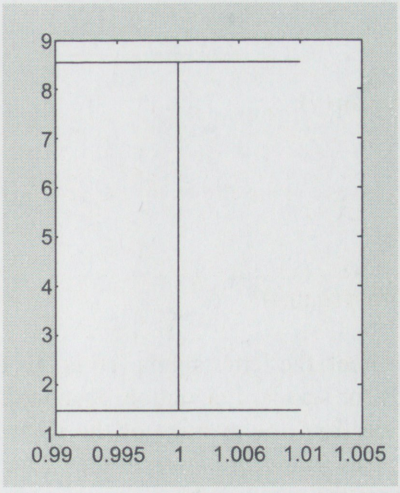
The MATLAB program of find the standard deviation of a set of data points is given below.

```
=input('enter the number of elements:')
or i=1:n
  a(i)=input('enter the element:');
nd
m=0;
or i=1:n
  sum=sum+a(i);
nd
ean=sum/n
=0;
or i=1:n
  s1=s1+(a(i)-mean).*(a(i)-mean);
nd
=s1/(n-1);
dv=sqrt (s2)
rorbar(mean,stdv)
```

ter the number of elements:5

=

5



The plot of the mean and the variance of the data.

```
enter the element:1
enter the element:2
enter the element:5
enter the element:8
enter the element:9
mean =
     5
stdv =
    3.5355
```

Here in this program errorbar has been used to plot the mean and the variance.

For the above set of data the command.

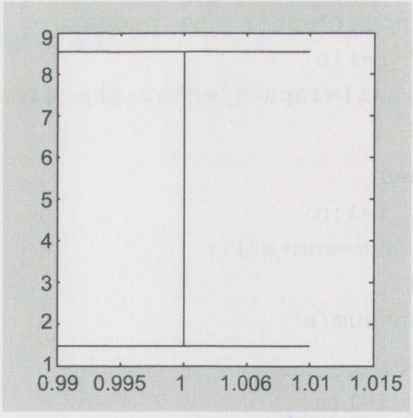
```
>> errorbar(1,5,3.5355)
```

Can be used to plot the errorbar. Here 1 is the x coordinate for sample 1,5 is the mean val and 3.5355 is the errorbar length or the variance.

Another example is given below.

In-built function std can also be used to find the standard deviation. Here in the example belo standard deviation has been found using the MTALAB function std.

```
>> x=[1 2 3 4 5];
y=[1 2 5 8 9];

>> s=std(y)

s =

    3.5355

>> m=mean(y)

m =

     5

>> errorbar(1,m,s)
```



The same plot as seen above has been obtained using MATLAB function std.

The Greek the letter sigma (σ) is the 18th letter of all the alphabets. In statistics the symb σ stands for standard deviation. Standard deviation is the measure of how widely apart the da points are from the average or the mean. Data points with a low standard deviation indicat that the data points are close to the mean. A high standard deviation indicates the points are f apart from the mean and have a large range of values.

.4 erf (x), ERROR FUNCTION

he error function also known as the Guass error function is defined as

$$\text{erf}(z) \equiv \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$$

has a sigmoid shape and is used in probability, statistics and partial differential equation. It is ncountered during integration of the normal distribution.

$$\text{erf}(-z) = -\text{erf}(z)$$

nplies that the error function is an odd function.

.5 erfc(x), COMPLIMENTARY ERROR FUNCTION

rfc is the complimentary error function, therefore

$$\text{erfc}(x) = 1 - \text{erf}(x)$$

The complimentary error function is defined as,

$$\text{erfc}(z) \equiv 1 - \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$$

erf(x)	1 – erf(x)
>> a=erf(0.5)	>> 1–a
ı =	ans =
0.5205	0.4795
>> b=erfc(0.5)	
) =	
0.4795	

.6 NORMAL DISTRIBUTION

probability and statistics the normal distribution also known as the Gaussian distribution is a ery commonly occurring continuous probability distribution. The normal distribution function lls us the probability of the occurrence of an event or observation between two given real ımbers.

The normal distribution function is,

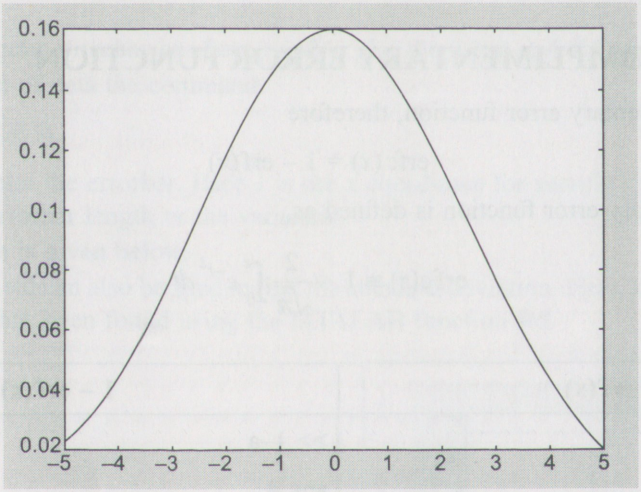
$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

The normal distribution or the Gaussian distribution is also called the bell curve as the curv looks like a bell. It should be noted that other distributions like the Cauchy's distribution or th logistic distribution is also bell shaped.

In MATLAB it is possible to plot a normal distribution using the function normpdf.

For example,

```
>> x = [–5:0.01:5];
norm = normpdf(x,0,2.5);
>> plot(x,norm)
```



Normal distribution plot

Here the mean (μ) is 0 and the standard deviation (σ) is 2.5.

In a standard bell curve, most data points (about 68%) fall within one standard deviation (1 of the mean. The vast majority (about 95%) fall within two standard deviations (2σ). An eve higher percentage (about 99.7%) fall within three standard deviations (3σ) of the mean. Just tiny fraction of points are outliers that are more than three standard deviations from the mea (about 0.15%). Let us consider the case where mean (μ) = 0 and standard deviation (σ) = 1. Th normal distribution plot is as shown below.

As can be seen most of the points lie in the region of $-\sigma$ to σ and as larger standa deviation is considered then the area under the curve increases and almost 95% of the are is covered between -2σ and 2σ. If the area is considered between -3σ and 3σ then the are increase slightly.

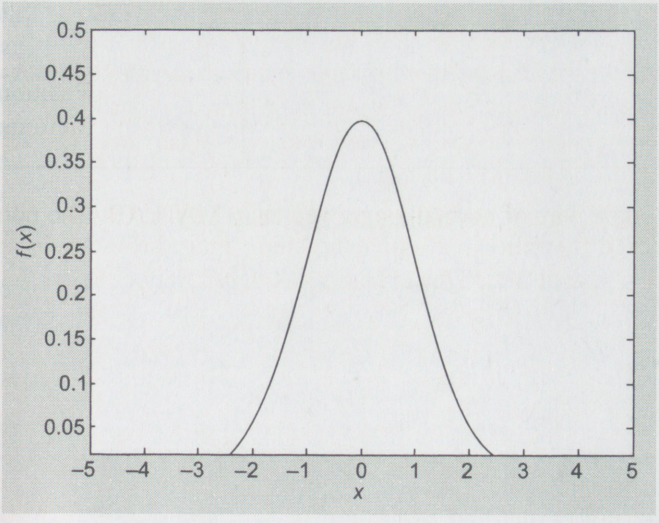
```
>> x = [–5:0.01:5];
>> norm = normpdf(x,0,1);
>> plot(x,norm)
```



It is possible to plot a normal distribution by writing the program using the normal distribution
nction,

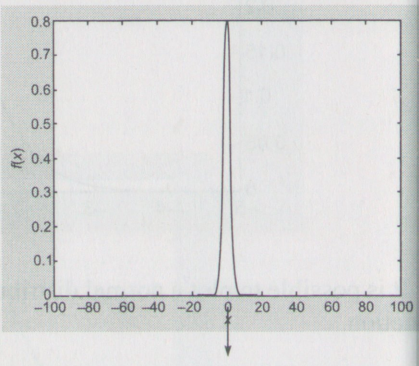
$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

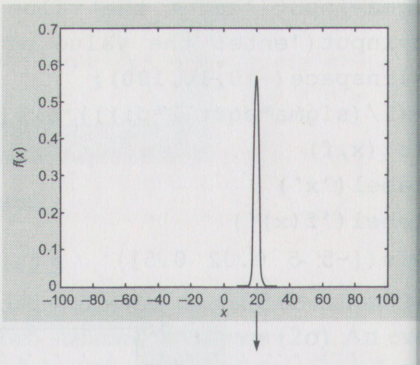
```
igma=input('enter the value of sigma or standard deviation:')
u=input('enter the value of mu or mean:')
=linspace(-10,10,100);
=(1/(sigma*sqrt(2*pi)))*exp(-(x-mu).*(x-mu)/2*sigma*sigma);
lot(x,f)
label('x')
label('f(x)')
xis([-5 5 0.02 0.5])
```



The normal distribution plot using the normal distribution function
A few more exalples are given below.

MATLAB Program	Input the values of σ and μ	Output

`sigma=input('enter the value of sigma')`
`mu=input('enter the value of mu')`
`x=linspace(-100,100,1000);`
`f=(1/ (sigma*sqrt(2*pi)))*exp(-(x-mu).*(x -mu)/2*sigma*sigma);`
`plot(x,f)`
`xlabel('x')`
`ylabel('f(x)')`
	enter the value of sigma0.5
	sigma =
	0.5000
	enter the value of mu0.2
	mu =
	0.2000
		
		Position of mean (μ)
		Here mean (μ) = 0.2

	enter the value of sigma0.7 sigma = 0.7000 enter the value of mu20 mu = 20	 Position of mean (μ) Here mean (μ) = 0.2

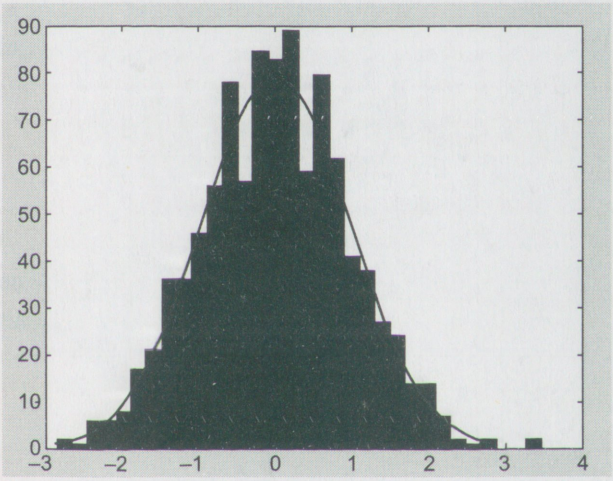
There is also a simple way of normalizing a vector in MATLAB. The normalized vector ca be obtained by,

```
>> v=[1,3];
>> n=v/norm(v)

n =

   0.3162   0.9487
```

The distribution of results of a test taken by a large population people is found to be normally istributed. Normal distributions are very important in probability and statistics and are used very requently for real-valued random variables in natural and social sciences whose distributions re unknown.

Here in the example below,

```
> x = randn(1000,1);
> histfit(x)
```



A bell-curve histogram from a Gaussian data

The hist(*x*) creates a histogram bar plot of the data. All the elements in data are sorted into 0 equally spaced containers along the *x*-axis between the minimum and the maximum value f the data and returns the number of elements in each container as a row vector. The bins are isplayed as rectangle where the height of the rectangle indicates the number of elements in the in. If *x* is an *m*-by-*p* matrix, hist treats the columns of *x* as vectors and returns a 10-by-*p* matrix. ach column of contains the results for the corresponding column of *x*. The elements of *x* cannot e complex or of type integer.

ndn(*m*) returns an *m* by *m* matrix containing pseudorandom values drawn from the standard ormal distribution.

```
> x = randn(2)
=
   -1.4064   -0.6689
    0.7199    0.5623
```

`andn(m,n)` or `randn(m,n)` returns an `m`-by-`n` matrix

```
> x = randn(3,2)
=
```

```
    0.7737   -0.7611
   -0.9014    0.8841
    1.2843    0.3543
```

`randn(m,n,p,...)` or `randn([m,n,p,...])` returns an m-by-n-by-p-by-... array.

```
>> x = randn(3,2,2)

x(:,:,1) =

   -0.3021   -0.5843
   -0.3145   -0.3035
   -1.0312    0.4537

x(:,:,2) =

    2.2466    1.5111
   -1.2702    0.2606
    1.8740   -1.6168

>> x = randn(3,2,2)

x(:,:,1) =

    0.3344   -1.5120
   -0.9622   -1.4373
    0.1343   -1.3733

x(:,:,2) =

   -1.1499    1.9211
   -0.2192   -0.4613
    1.0047    1.3241
```

9.7 NORMALIZATION

In probability and statistics a normalizing constant is a constant which when multiplied to a everywhere non-negative function gives an area equals to 1. This is used to make a probabili density function or a probability mass function. The norm a matrix is a scalar that gives som measure of the magnitude of the elements of the matrix. In MATLAB also it is possible t normalize a vector using the function norm. For example,

```
>> a=[1 2 3 4];
>> norm(a)

ans =

    5.4772
```

This is the same as

$$\sqrt{(1^2 + 2^2 + 3^2 + 4^2)} = \sqrt{27} = 5.4772$$

Thus, the norm of a matrix is a scalar that gives some measure of the magnitude of the lements of the matrix.

When a is a vector `norm(a,p)` returns

`sum(abs(a).^p)^(1/p)`, for any 1 <=p<= ∞.norm(a)

hereas norm(a) returns `norm(a,2)`.

Thus, the MATLAB function `norm` calculates several different types of matrix norms `orm(a,p)` depending on the value of p.

```
> a=[1 2 3 4];
> norm(a,1)
ns =
    10
```

orm(a,1) gives the sum of the elements of vector a.

orm(a,inf) returns max(abs(a)) whereas norm(a,-inf) returns min(abs(a)).

```
> norm(a,inf)
ns =
    4
> norm(a,-inf)
ns =
    1
```

If $\psi(x, y)$ is the wave function then the probability density of the particles displacement of x $|\psi(x, t)|^2$. Thus, the probability of the displacement between a and b is given by,

$$P_{x \in a:b(t)} = \int |\psi(x,t)|^2 \, dx$$

Condition for normalization is,

$$P_{x \in a:b(t)} = \int |\psi(x,t)|^2 \, dx = 1$$

Now let us consider the problem of particle in a box. This is one of the simplest quantum echanical problems. Consider a particle having a wavefunction,Ψ, in one dimension within a quare well potential. The potential in the well is defined as,

$$\begin{aligned} &\infty: x < 0 \\ V(x) &= 0: 0 \le x \le L \\ &\infty: x > L \end{aligned}$$

For the particle in the box with infinite walls the probability of finding the particle in the box 1. Applying the condition of the potential in the well to the particle gives,

$$\psi(x) = A\sin(kx) + B\cos(kx)$$

From the boundary conditions we get,

$$\Psi(0) = 0 \text{ and } \Psi(L) = 0$$

Since $\Psi(0) = 0$

$$B = 0$$

So,

$$\Psi(x) = A\sin(kx)$$

Again since $\Psi(L) = 0$

So,

$$A\sin(kx) = 0$$

or

$$kL = n\pi$$

or

$$k = \frac{n\pi}{L}$$

The condition of normalization gives,

$$\int \psi x \psi \, dx = \int_0^L A_n^2 \sin^2\left(\frac{n\pi}{L}x\right) dx = 1$$

or

$$A_n^2\left[\frac{x}{2} - \frac{\sin\left(\frac{n\pi}{L}x\right)}{\frac{4n\pi}{L}}\right]_0^L = 1$$

This leads to $$A_n^2 \frac{L}{2} = 1$$

or $$A_n = \sqrt{\frac{2}{L}}$$

Therefore the normalized wave function is

$$\psi(x) = \sqrt{2/L}\sin\left(\frac{n\pi}{L}x\right)$$

It is often desired to arrange a very large array of numbers to fit between 0 and 1. All th elements of the array have to between 0 and 1. The normalized value of array elements can b obtained using the formula,

```
-min(a)/(max(a)-min(a))
 = [ 10 20;30 40];
s = (m-min(m(:))./(max(m(:)-min(m(:))));
sp('the minimum value is:');
in(ms(:))
sp('the maximum value is:');
ax(ms(:))
sp('the scaled matrix is:');
s
e minimum value is:
s =
    0
e maximum value is:
s =
    1
e scaled matrix is:
s =
    0         0.3333
    0.6667    1.0000
```

3 REPRESENTATION OF UNION, INTERSECTION, EXCLUSIVE OR AND SUBTRACTION

$A \cup B$ or A union B implies that either A or B or both occurs.

$A \cap B$ or A intersection B implies that both A and B occur.

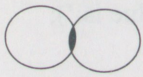
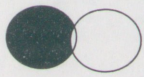
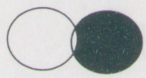
$A \subseteq B$ or A is a subset of B implies that if A occurs so does B.

A' or $\bar{A}$ implies that event A does not occur.

XOR or exclusive OR yields true if exactly one (but not both) of the two conditions true. The th table of A XOR B is given below.

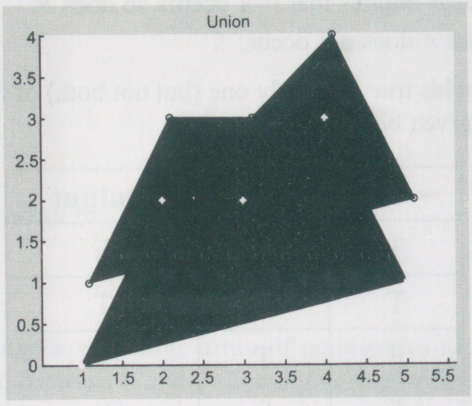
A	B	Output
T	T	F
T	F	T
F	T	T
F	F	F

A few Venn diagrams showing the $A \cup B$, $A \cap B$, subtraction A–B and B–A and $(A \cap B)'$.

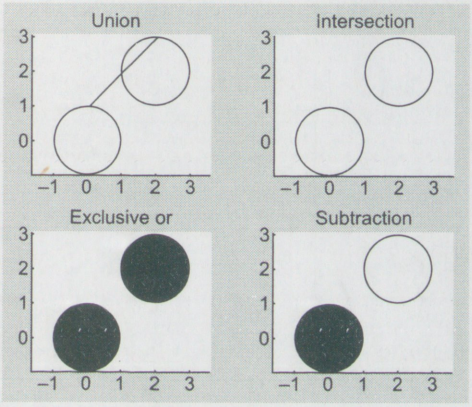
Union $A \cup B$	
Intersection $A \cap B$	
Subtraction A–B	
Subtraction, B–A	
$(A \cap B)'$	

A few illustrations of union, intersection, exclusive OR and subtraction are given below.

```
x=[1 2 3 4 5];
y=[0 2 2 3 1];
xdash = x+0.1;
ydash = y+1;
[x1, y1] = polybool('union', x, y, xdash, ydash);
patch(x1, y1, 1, 'FaceColor', 'r')
axis equal, hold on
plot(x, y, xdash, ydash, 'Color', 'k')
plot(x,y,'y*',xdash,ydash,'ob');
title('Union')
```



MATLAB Program	Output	Explanation

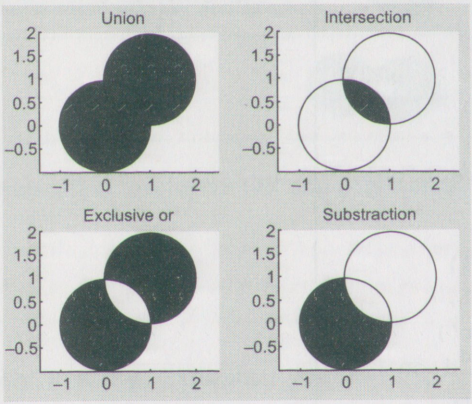
theta = linspace(0, 2*pi, 100);
x = sin(theta);
y = cos(theta);
a = x+2;
b = y+2;
[x1, y1] = polybool('union', x, y, a, b);
[x2, y2] = polybool('intersection', x, y, a, b);
[x3, y3] = polybool('xor', x, y, a, b);
[x4, y4] = polybool('subtraction', x, y, a, b);
 	
subplot(2, 2, 1)
patch(x1, y1, 1, 'FaceColor', 'r')
axis equal, hold on
plot(x, y, a, b, 'Color', 'k')
title('Union')
 	
subplot(2, 2, 2)
patch(x2, y2, 1, 'FaceColor', 'r')
axis equal, hold on
plot(x, y, a, b, 'Color', 'k')
title('Intersection')
 	
subplot(2, 2, 3)
[f, v] = poly2fv(x3, y3);
patch('Faces', f, 'Vertices', v, 'FaceColor', 'r', ...
'EdgeColor', 'none')
		The four figures in the subplots show union, intersection, exclusive OR and subtraction.

Contd.

Contd.

```
axis equal, hold on
plot(x, y, a, b, 'Color',
'k')
title('Exclusive Or')

subplot(2, 2, 4)
patch(x4,    y4,    1,
'FaceColor', 'r')
axis equal, hold on
plot(x, y, a, b, 'Color',
'k')
title('Subtraction')
```

```
theta = linspace(0,
2*pi, 100);
x = sin(theta);
y = cos(theta);
a = x+1;
b = y+1;
[x1, y1] =
polybool('union', x, y,
a, b);
[x2, y2] =
polybool('intersection',
x, y, a, b);
[x3, y3] =
polybool('xor', x, y, a,
b);
[x4, y4] =
polybool('subtraction',
x, y, a, b);

subplot(2, 2, 1)
patch(x1, y1, 1,
'FaceColor', 'r')
axis equal, hold on
plot(x, y, a, b,
'Color', 'k')
title('Union')

subplot(2, 2, 2)
patch(x2, y2, 1,
'FaceColor', 'r')
```



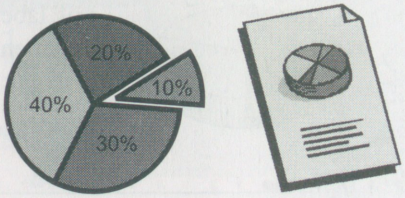
Conto

Contd.

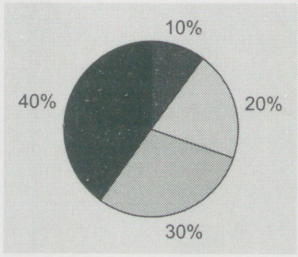
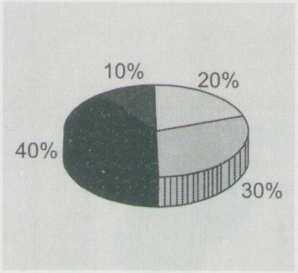
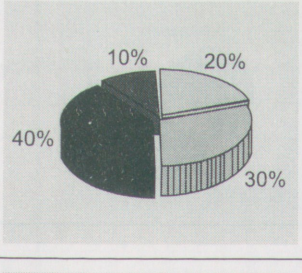
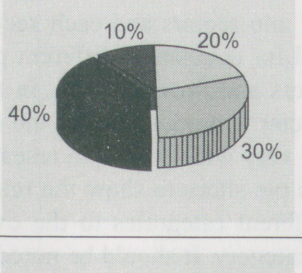
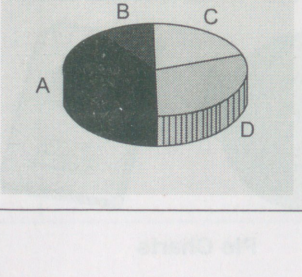
```
axis equal, hold on
plot(x, y, a, b,
'Color', 'k')
title('Intersection')

subplot(2, 2, 3)
[f, v] = poly2fv(x3,
y3);
patch('Faces', f,
'Vertices', v,
'FaceColor', 'r', ...
'EdgeColor', 'none')
axis equal, hold on
plot(x, y, a, b,
'Color', 'k')
title('Exclusive Or')

subplot(2, 2, 4)
patch(x4, y4, 1,
'FaceColor', 'r')
axis equal, hold on
plot(x, y, a, b,
'Color', 'k')
title('Subtraction')
```

9.9 PIE CHART

A pie chart is a circular chart divided into sectors and each sector represents the relative size of each data. A pie chart is very useful for comparing different parts of a whole amount. They are used widely in representing business and financial data. In the pie chart the arc length or the central angle or the area of each sector is proportional to the quantity it represents. Its name is due to its resemblance to a pie. It is very widely used in research, business and mass media. A pie chart is a circular chart and uses pie slices to show the relative sizes of data. A pie chart shows the relative contribution of different categories to the overall total. The wedge of the circle represents the contribution of a category. It should be noted that every 1% contribution of a category to the total is equivalent to 3.6° in the circular chart.

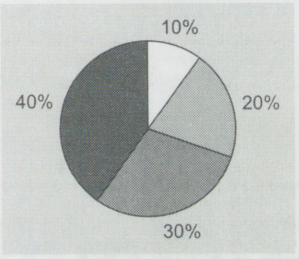
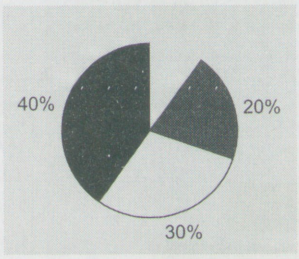
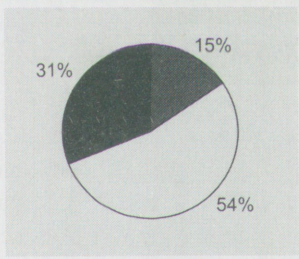


Pie Charts

MATLAB Command	Output	Explanation
>> x=[40 30 20 10]; >> pie(x) >>		`pie(X)` draws a pie chart using the data given in `X`. Each element in `X` is represented as a slice in the pie chart.
>> x=[40 30 20 10]; >> pie3(x) >>		`pie3(X)` draws a 3-dimensional pie chart using the data given in `X`. Each element in `X` is represented as a slice in the pie chart.
>> x=[40 30 20 10]; >> explode=[1 0 1 0]; >> pie3(x,explode) >>		`pie3(X,explode)` specifies whether to offset a slice from the center of the pie chart. `X(i,j)` is offset from the center of the pie chart if `explode(i,j)` is nonzero. `explode` must be the same size as `X`.
>> x=[40 30 20 10]; >> explode=[1 0 0 0]; >> pie3(x,explode) >>		Here the first slice in the 3-dimensional pie chart is emphasized by setting its corresponding `explode` element to 1.
>> x=[40 30 20 10]; >> pie3(x,{'A','B','C','D'}) >>		`pie3(...,labels)` specifies text labels for the slices. The number of labels must equal the number of elements in `X`.

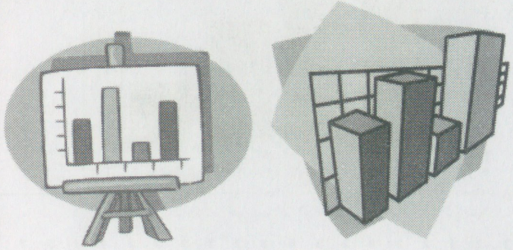
Contd

ontd.

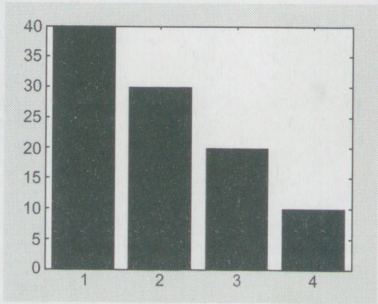
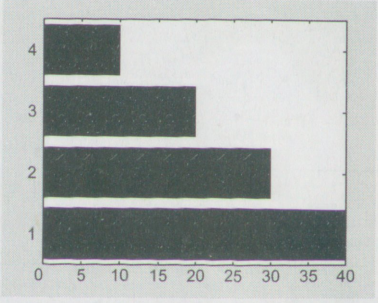
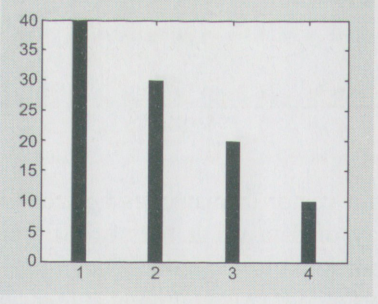
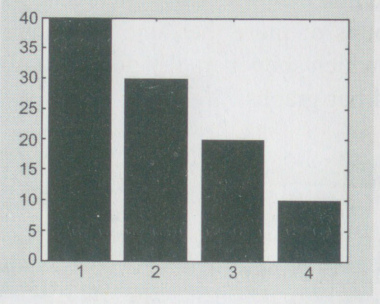
>> x=[40 30 20 10]; >> pie(x) >> colormap autumn >>		Here the colormap is set to autumn.
>> x=[0.4 0.3 0.2]; >> pie(x) >>		When the sum of the elements in the first input argument is less than 1, `pie` and `pie3` do not normalize the elements of vector `x`. They draw a partial pie as shown in this case.
>> x=[0.4 0.7 0.2]; >> pie(x) >>		When the sum of the elements in the first input argument is equal to or greater than 1, `pie` and `pie3` normalize the values. So, given a vector of elements x, each slice has an area of $x_i/\text{sum}(x_i)$, where x_i is an element of x. The normalized value specifies the fractional part of each pie slice.

.10 BAR CHART

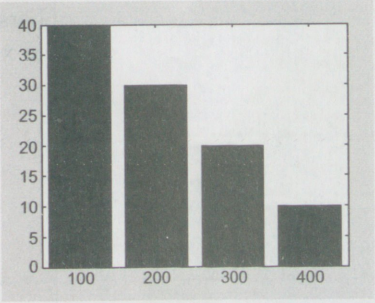
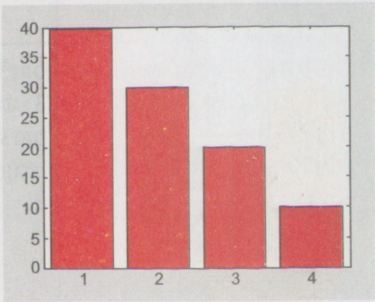
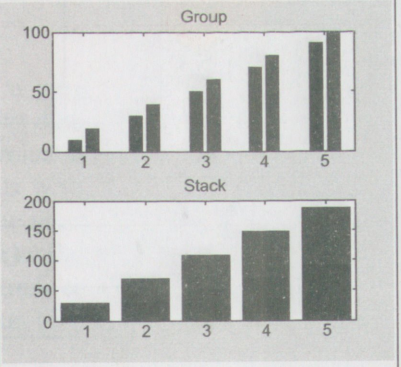
bar graph displays the values in a vector or matrix as horizontal or vertical bars. A bar chart r bar graph is a graphical display of data using bars of different heights. It is a chart with ectangular bars with lengths proportional to the values they represent. The bars can be plotted orizontally and vertically. It shows a comparison among categories. One of the axes shows the pecific categories being compared and the other axis represents discrete value. Some bar graphs lso present bars clustered in groups of more than one. They are also known as grouped bar raphs. There are also bar graphs which show bars divided into subparts to show the cumulative ffect. They are known as stacked bar graphs.



Bar charts

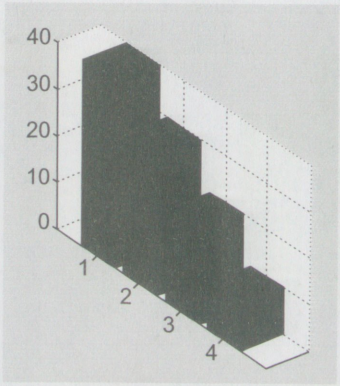
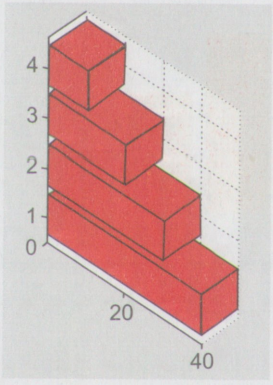
MATLAB Command	Output	Expalantion
>> x=[40 30 20 10]; >> bar(x)		`bar(x)` draws one bar for eac element in `x`.
>> x=[40 30 20 10]; >> barh(x)		`barh(x)` draws horizontal bars The length of the bar depend on the value it represents.
>> x=[40 30 20 10]; >> bar(x,0.2)		bar(...,width) sets the relative bar width and controls the separation of bars within a group. The default width is 0.8 If width is 1 then the bars within a group touch one another.
>> x=[40 30 20 10]; >> bar(x,0.8)		The value of width must be a scalar. In the examples given here the width is selected as 0.2 and 0.8.

Contd.

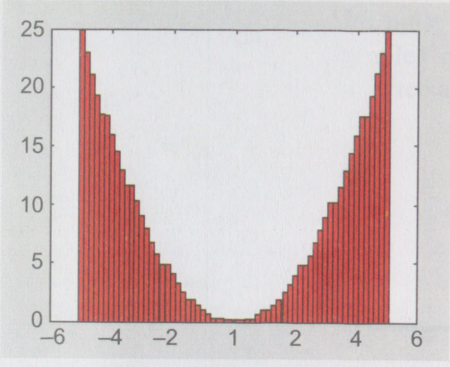
>> x=[40 30 20 10]; >> y=[100 200 300 400]; >> bar(y,x)		`bar(x,Y)` draws a bar for each element in `Y` at locations specified in `x`, where `x` is a vector defining the *x*-axis intervals for the vertical bars.
>> x=[40 30 20 10]; >> bar(x,'r')		`bar(...,'bar_color')` displays all bars using the color specified by the single-letter abbreviation `'r', 'g', 'b', 'c', 'm', 'y', 'k',` or `'w`. Here the selected colour is `'r'` or red.
x=[10 20; 30 40; 50 60; 70 80; 90 100]; subplot(2,1,1) bar(x,'grouped') title 'Group' subplot(2,1,2) bar(x,'stacked') title 'Stack'		bar(..., *'style'*) specifies the style of the bars. The' style' could be 'grouped' or 'stacked'. 'group' is the default mode of display. In 'grouped' displays *m* groups of *n* vertical bars, where *m* is the number of rows and *n* is the number of columns in *Y*. The group contains one bar per column in *Y*. Whereas 'stacked' displays one bar for each row in *Y*. The bar height is the sum of the elements in the row. Each bar is multicolored, with colors corresponding to distinct elements and showing the relative contribution each row element makes to the total sum.

Contd.

Contd.

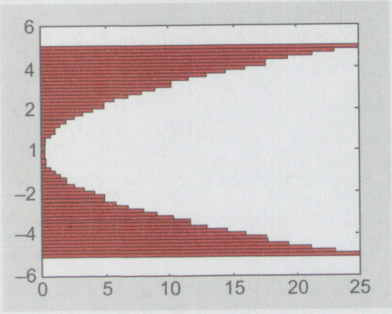
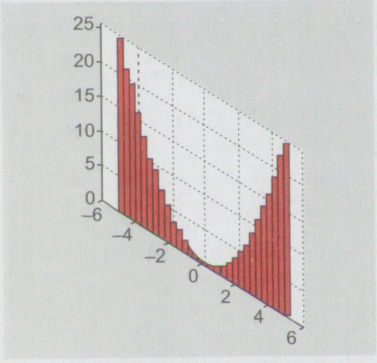
>> x=[40 30 20 10]; >> bar3(x)		bar3(x) draws 3-dimesiona bars one bar for each elemen in x.
>> x=[40 30 20 10]; >> bar3h(x,'red')		bar3h(x) is used to create 3-dimesional horizontal bars. x determines the bar length.

A few other examples using bar and bar3 function are given below. Here a function $y=x^2$ ha been used in the bar plot.

MATLAB Commands	Output
>> x=-5:0.2:5; >> bar(x,x.*x,'r')	

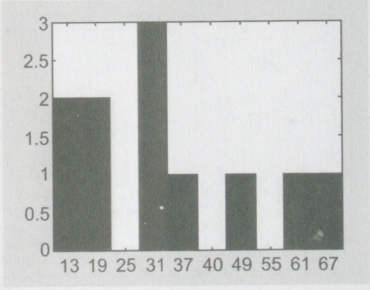
Contd

ontd.

>> x=-5:0.2:5; >> barh(x,x.*x,'r')	
>> x=-5:0.2:5; >> bar3(x,x.*x,'r')	

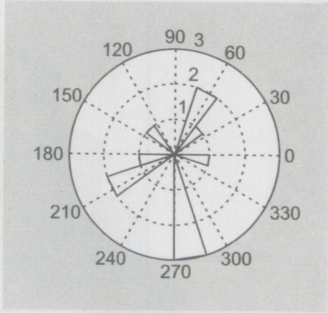
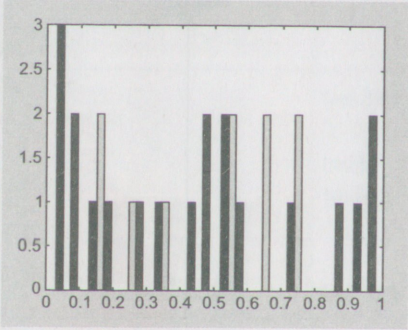
.11 HISTOGRAM

histogram in statistics is a graphical representation of the distribution of data. It gives us n estimate of the probability distribution of a continuous variable. It was introduced by Karl earson. A histogram shows us the tabulated frequencies in the form of adjacent rectangles laced over discrete intervals with an area proportional to the frequency of the observation in ne interval. The height of the rectangle is equal to the frequency density of the interval in other vords the frequency divided by the width of the interval. The total area of the histogram gives us ne number of data. A histogram could also be normalized showing relative frequencies. In such case it shows the proportion of cases that fall into each category and the total area equals 1.

MATLAB Command	Output	Explanation
>> x=[10 20 30 50 10 30 70 40 60 30 20]; >> hist(x)		

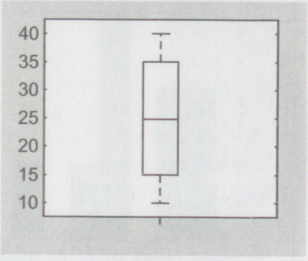
Contd.

Contd.

>> x=[10 20 30 50 10 30 70 40 60 30 20]; >> rose(x)		A rose plot is histogram created i a polar coordinat system.
>> x=rand(10,3) x = 0.0759 0.1622 0.4505 0.0540 0.7943 0.0838 0.5308 0.3112 0.2290 0.7792 0.5285 0.9133 0.9340 0.1656 0.1524 0.1299 0.6020 0.8258 0.5688 0.2630 0.5383 0.4694 0.6541 0.9961 0.0119 0.6892 0.0782 0.3371 0.7482 0.4427 >> >> hist(x)		When x is a matrix hist creates a set o bins for each column displaying each set i a separate color.

9.12 boxplot

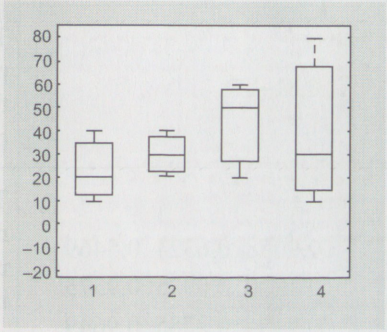
The boxplot(x) function produces a box plot of the data in x. There is one box per column. box plot is a useful non-parametric statistical plot. Box plots are especially useful when the da are not normally distributed.

MATLAB Command	Output	Explanation
>> x=[40 30 20 10]; >> boxplot(x) >>		This plot shows that the maximum is 40 and the minimum is 10. It also shows that the mean is 25.

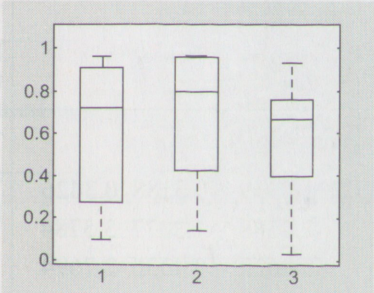
Conto

Contd.

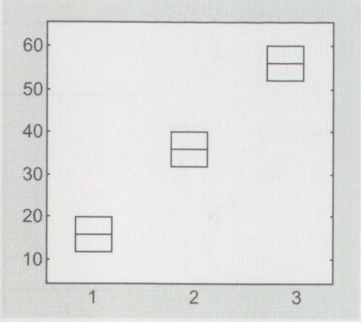
```
>> x=[40 30 20 10;20 40 50
30;10 20 60 30];
>> boxplot(x)
>>
```




```
>> x=rand(10,3)
x =
    0.8147    0.1576    0.6557
    0.9058    0.9706    0.0357
    0.1270    0.9572    0.8491
    0.9134    0.4854    0.9340
    0.6324    0.8003    0.6787
    0.0975    0.1419    0.7577
    0.2785    0.4218    0.7431
    0.5469    0.9157    0.3922
    0.9575    0.7922    0.6555
    0.9649    0.9595    0.1712
>> boxplot(x)
```




```
>> x = [10,20,30,40,50,60];
>> group = [1,1,2,2,3,3];
>> boxplot(x,group)
```



boxplot groups allows different values of one big vector to be mapped to different boxes on the same canvas.

13 RANDOM NUMBERS

pseudorandom number generator is a mechanism for generating random numbers on a computer at are indistinguishable from truly random numbers. It is called pseudorandom because it is t possible to generate a truly random number from a deterministic thing like computer. A truly ndom number is generated by a random physical process such as radioactive decay.

Function	Output	Explanation
>> rand	ans = 0.8147	The rand function returns pseudorandom scalar drawn fro the standard uniform distributio on the open interval (0,1).
>> rand (3)	ans = 0.9058 0.6324 0.5469 0.1270 0.0975 0.9575 0.9134 0.2785 0.9649	The function rand (n) returns a n-by-n matrix of pseudorandor uniform values from the ope interval (0,1).
>> randn	ans = 0.5377	The function randn returns psuedorandom scalar drawn fro the standard normal distributio
>> randn (3)	ans = 1.8339 0.3188 0.3426 −2.2588 −1.3077 3.5784 0.8622 −0.4336 2.7694	The function randn (n) retur an n-by-n matrix containin pseudorandom normal values.
>> randi (3)	ans = 3	The function randi (n) returns random integer on the interv 1: n.
>> randi (3,3)	ans = 3 2 2 1 1 3 3 1 3	The function randi (n,m) retur an m-by-m matrix containin pseudorandom integer valu drawn from the discrete unifor distribution on the interval 1: n
>> randi (3,[2,3])	ans = 1 3 3 3 2 1	The function randi (n,[k,l]) retur an k-by-l matrix containin pseudorandom integer valu drawn from the discrete unifor distribution on the interval 1: n
>> randi ([3,10],[2,2])	ans = 8 9 3 10	The function randi ([m,n],.. returns an array containin integer values drawn from th discrete uniform distribution o the interval m:n.

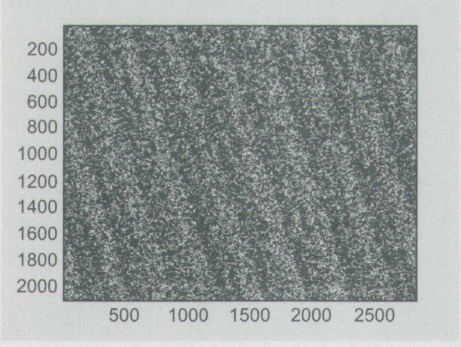
Cont

ntd.

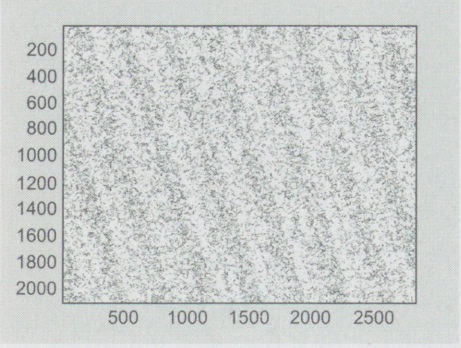
>random('Normal',0,1,3,3)	ans = 0.5377 0.8622 −0.4336 1.8339 0.3188 0.3426 −2.2588 −1.3077 3.5784	The funciton random(name,A) where name is the name of the type of distribution that takes a single parameter, returns random numbers from the one-parameter family of distributions specified by name. Parameter values for the distribution are given in A. For example, in random('Normal',0,1,3,3) the distribution is normal between 0 and 1 and generates an array of 3 by 3.

A few examples of images created using random numbers is given below.

```
> cd D:\Pictures
> a=imread('bird.jpg','jpg');
> dotpic=rand(size(a));
> image(dotpic)
```

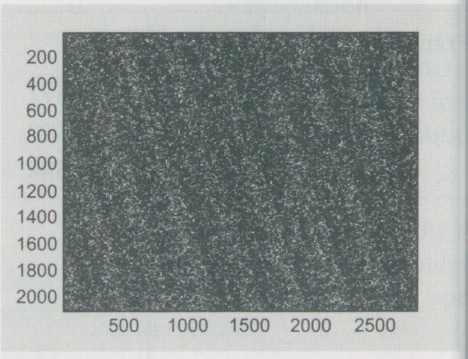



```
> cd D:\Pictures
> a=imread('bird.jpg','jpg');
> dotpic=rand(size(a))>0.1;
> image(dotpic)
```



Contd.

```
>> cd D:\Pictures
>> a=imread('bird.jpg','jpg');
>> dotpic=rand(size(a))>0.6;
>> image(dotpic)
```

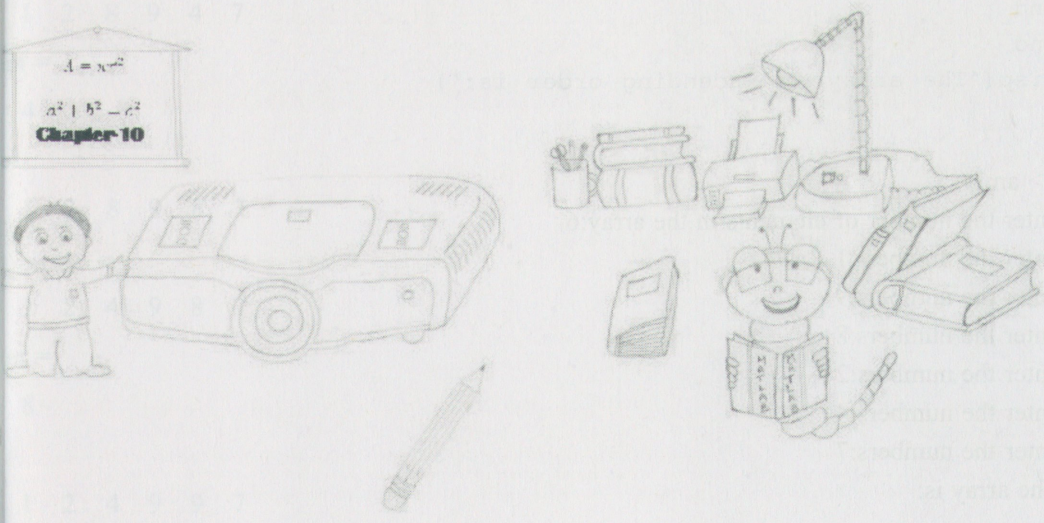


10
CHAPTER

Solved Problems

novation distinguishes between a leader and a follower."

...Steve Jobs



1. Arrange an array in ascending order using the linear sort method.

Solution

```
m=input('enter the number of elements in the array:');
for i=1:m
k=input('enter the numbers:');
a(i)=k;
end
disp('The array is:')
a
for i=1:(m-1)
    min=a(i);
for j=(i+1):m
if(a(j)<min)
            min=a(j)
            a(j)=a(i)
            a(i)=min
end
end
end
disp('The array in ascending order is:')
a
```

```
>> amin
enter the number of elements in the array:6
enter the numbers:1
enter the numbers:9
enter the numbers:8
enter the numbers:2
enter the numbers:4
enter the numbers:7
The array is:
a =
   1   9   8   2   4   7
min =
   8
a =
   1   9   9   2   4   7
```

=

1 8 9 2 4 7

in =

2

=

1 8 9 8 4 7

=

1 2 9 8 4 7

in =

8

=

1 2 9 9 4 7

=

1 2 8 9 4 7

in =

4

=

1 2 8 9 8 7

=

1 2 4 9 8 7

in =

8

=

1 2 4 9 9 7

=

1 2 4 8 9 7

in =

7

=

1 2 4 8 9 8

```
a =
    1   2   4   7   9   8
min =
    8
a =
    1   2   4   7   9   9
a =
    1   2   4   7   8   9
```

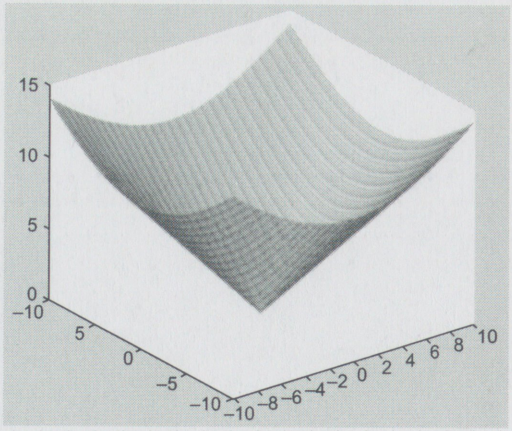
The array in ascending order is:

```
a =
    1   2   4   7   8   9
```

2. Use figurepalette to plot a surface $z = \sqrt{x^2 + y^2}$

Solution

```
>> [x,y]=meshgrid([-10:0.1:10]);
>> z=sqrt(x.*x+y.*y);
>> figurepalette
>> plot3(x, y, z, 'DisplayName', 'x, y, z'); figure(gcf)
>>
```



3. Use linear interpolation to estimate the value of *ay* for a given value of *x*.

Solution

```
m=input('enter the number of elements:');
for i=1:m
```

```
=input('enter the number:');
(i)=k;
nd
isp('The array is:')


or j=1:m
=input('enter the number:');
(j)=k;
nd
isp('The array is:')


x=input('enter the point where we have to find y:');
nt=0;
=m-1;
or k=1:n
f xx<=x(k+1)
        int=y(k)+(y(k+1)-y(k))*((xx-x(k))/(x(k+1)-x(k)));
nd
nd

isp('The interpolated value of y at this value of x is:')
nt
```

```
nter the number of elements:4
nter the number:1
nter the number:2
nter the number:3
nter the number:4
he array is:
=
  1   2   3   4
nter the number:2
nter the number:4
nter the number:6
nter the number:8
he array is:
=
  2   4   6   8
```

enter the point where we have to find y:2.5
The interpolated value of y at this value of x is:

int =

5

4. Using the four images given below make a movie in MATLAB.

Solution

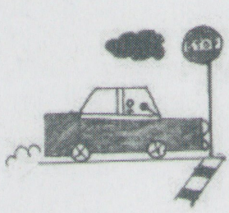


Image-1



Image-2

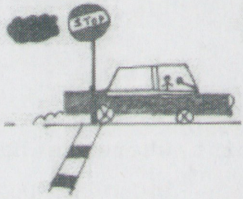


Image-3

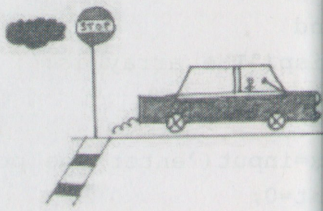
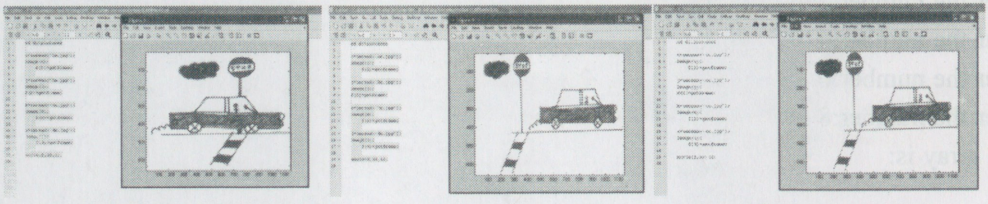


Image-4

```
cd d:\pictures
i=imread('1m.jpg');
image(i);
    f(1)=getframe;

j=imread('2m.jpg');
image(j);
f(2)=getframe;

k=imread('3m.jpg');
image(k);
    f(3)=getframe;

l=imread('4m.jpg');
image(l);
    f(4)=getframe;

movie(f,10,12)
```



Three images of the figure window showing the changes in frames with time

5. Find the element by element exponential of a matrix.

lution

```
a=[1 2 3 4 5];
exp(a)
```

```
s =

   2.7183   7.3891   20.0855   54.5982   148.4132
```

Again if a is higher order matrix element by element exponential of the matrix the matrix can so be found out using function exp(a).

```
a=[1 2 3 4 5;2 3 4 5 6];
exp(a)
```

```
s =

   2.7183    7.3891   20.0855    54.5982   148.4132
   7.3891   20.0855   54.5982   148.4132   403.4288
```

6. Find the transpose of a square matrix.

```
input('enter the value of rows:');
input('enter the value of columns:')
r i=1:m
r j=1:n
      a(i,j)=input('enter the element');
d
d
sp('the matrix is:')

r i=1:m
r j=i:n
      temp=a(i,j);
      a(i,j)=a(j,i);
      a(j,i)=temp;
d
d
sp('the transposed matrix is:')
a
```

```
er the value of rows:3
er the value of columns:3
```

```
n =
    3
enter the element1
enter the element2
enter the element3
enter the element4
enter the element5
enter the element6
enter the element7
enter the element8
enter the element9
the matrix is:
a =
    1   2   3
    4   5   6
    7   8   9
the transposed matrix is:
a =
    1   4   7
    2   5   8
    3   6   9
```

7. Draw 4 atoms at the 4 vertices of a tetrahedron.

The coordinates of the atoms are,

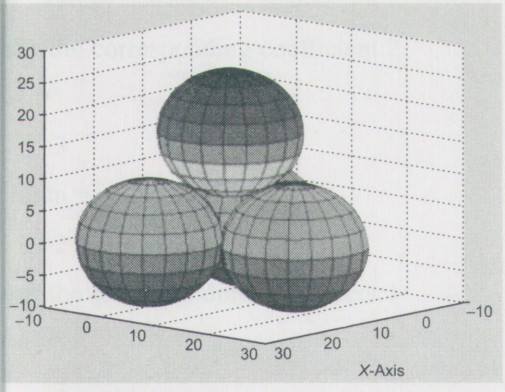
$$(0, 0, 0), (2R, 0, 0), (R, \sqrt{3R}, 0) \text{ and } \left(R, \frac{R}{\sqrt{3}}, 2\sqrt{\frac{2}{3}R}\right)$$

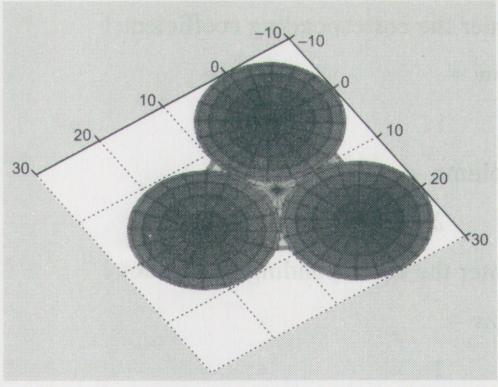
where R is the radius of the atom.

Solution

```
[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
a1=10*cos(v).*sin(u);
b1=10*sin(v).*sin(u);
c1=10*cos(u);
surf(a1,b1,c1);
hold on
[u,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
a2=20+10*cos(v).*sin(u);
b2=0+10*sin(v).*sin(u);
```

```
=0+10*cos(u);
rf(a2,b2,c2);
ld on
,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
=10+10*cos(v).*sin(u);
=10*sqrt(3)+10*sin(v).*sin(u);
=0+10*cos(u);
rf(a3,b3,c3);
abel('X-Axis')
abel('Y-Axis')
ld on
,v] = meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
=10+10*cos(v).*sin(u);
=(10/sqrt(3))+10*sin(v).*sin(u);
=(2*sqrt(2)/sqrt(3))*10+10*cos(u);
rf(a4,b4,c4);
abel('X-Axis')
abel('Y-Axis')
```





8. Write a MATLAB program to solve a system of linear equations.

lution

```
input('enter the number of equations:');
r(i=1:m)
r(j=1:m)
  row=i
  column=j
  a(i,j)=input('enter the corresponding coefficient:');
d
d
```

```
disp('The matrix of coefficients is:')
a

for(i=1:m)
equation=i
    c(i,1)=input('enter the constant term:');
end
disp('The matrix of constant terms is:')
c
disp('The solution is:')
x=a\c
```

```
enter the number of equations:3
row =
     1
column =
     1
enter the corresponding coefficient:1
row =
     1
column =
     2
enter the corresponding coefficient:1
row =
     1
column =
     3
enter the corresponding coefficient:2
row =
     2
column =
     1
enter the corresponding coefficient:1
```

```
row =
     2
column =
     2
enter the corresponding coefficient:1
row =
     2
column =
     3
enter the corresponding coefficient:1
row =
     3
column =
     1
enter the corresponding coefficient:2
row =
     3
column =
     2
enter the corresponding coefficient:1
row =
     3
column =
     3
enter the corresponding coefficient:1
The matrix of coefficients is:
a =
     1   1    2
     1   1   -1
     2   1   -1
```

```
equation =
     1
enter the constant term:4
equation =
     2
enter the constant term:1
equation =
     3
enter the constant term:2
The matrix of constant terms is:
c =
     4
     1
     2
The solution is:
x =
     1
     1
     1
>>
```

9. In the string 'This is the first sentence' find out the position of 'th'. Also replace 'Th' by 'th' in the above string.

Solution

```
>> sentence='This is the first sentence.'
sentence =
This is the first sentence.
>> regexp(sentence,'th')
ans =
     9
>> regexpi(sentence,'th')
```

```
ans =
     1   9
>> regexprep(sentence,'Th','th')
ans =
this is the first sentence.
>>
```

Here we should not that the function regexp is case sensitive and shows the position of 'th' as 9 only whereas the function regexpi is case insensitive and shows both 1 and 9 as the positions of 'th'. In the case of regexpi both 'Th' and 'th' are treated the same. Using regexprep function we can replace a string by another string.

Function	Command	Output
regexp	regexp(string1,'string2')	Finds all occurrences of 'string2' in string1. regexp is case sensitive.
regexpi	regexpi(string1,'string2')	Finds all occurrences of 'string2' in string1. regexpi is case insensitive.
regexprep	regexprex(string1,'string2','string3'	In string1 replace string2 by string3. regexprep is case sensitive.

A few examples using these functions have been given below.

```
>> sentence='This is the first sentence.'
sentence =
This is the first sentence.
>> regexp(sentence,'the first')
ans =
     9
>> regexp(sentence,'the four')
ans =
     []
>>
>> regexpi(sentence,'the FIRST')
ans =
     9
```

```
>>
>> sentence='This is the first sentence.'
sentence =
This is the first sentence.
>> regexprep(sentence,' ','''')
ans =
This''is''the''first''sentence.
>>
>> regexprep(sentence,'This is','Read')
ans =
Read the first sentence.
>> regexprep(sentence,'this is','Read')
ans =
This is the first sentence.
```

10. Find the minimum and maximum number in an array and also find the position of the maximum and maximum number in the array.

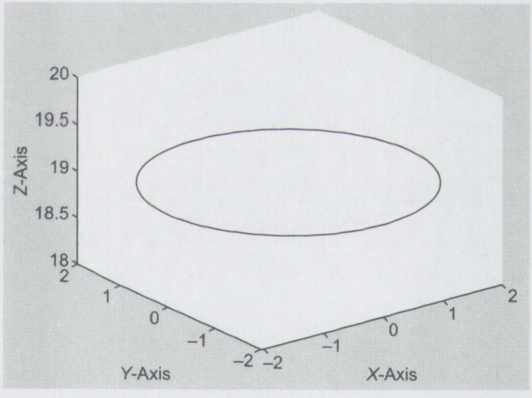
Solution

```
>> a=[1 2 3 4;2 8 1 6;9 1 6 1]
a =
    1   2   3   4
    2   8   1   6
    9   1   6   1
>> [v,p]=min(a)
v =
    1   1   1   1
p =
    1   3   2   3
>>
>> [v,p]=max(a)
v =
    9 8 6 6
p =
    3 2 3 2
>>
```

11. Draw a circle having a center as (0,0,z) coordinate and a given radius.

Solution

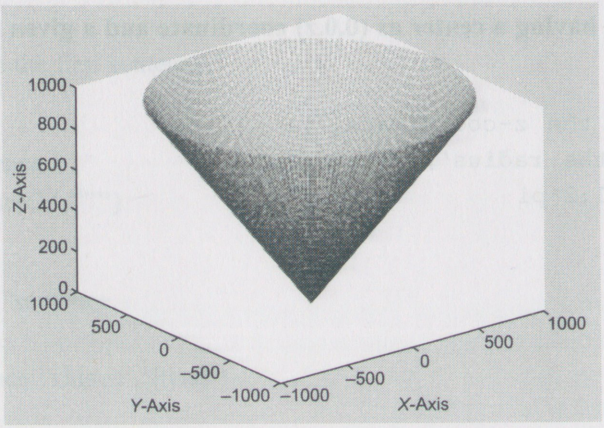
```
z1=input('enter the z-coordinate:');
r=input('enter the radius');
for theta=0:0.01:2*pi
x=r*cos(theta);
y=r*sin(theta);
plot3(x,y,z)
hold on
end
xlabel('X-Axis')
ylabel('Y-Axis')
zlabel('Z-Axis')
```



12. Create a tornado.

Solution

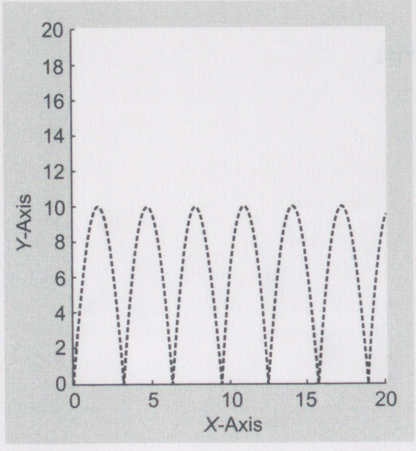
```
for i=10:10:1000
for theta=0:0.01*pi:2*pi
    x=i*cos(theta);
    y=i*sin(theta);
    z=i;
    plot3(x,y,z)
    hold on
end
    hold on
end
    xlabel('X-Axis');
    ylabel('Y-Axis');
    zlabel('Z-Axis');
```



13. Generate 10abs(sin(x)) curve using animation.

Solution

```
xlabel('X-Axis')
ylabel('Y-Axis')
x=linspace(0,200*pi,10000);
y=10*abs(sin(x));
hb=line(x(1),y(1),'marker','o','color','g','markersize',16,'erase','xor');
ht=line(x(1),y(1),'marker','.','color','g','erase','none');
axis('square');
axis([0 20 0 20])
for k=2:length(x)
   set(hb,'xdata',x(k),'ydata',y(k));
   set(ht,'xdata',x(k),'ydata',y(k));
   drawnow
end
```



14. Read an array and find the number of times a number appears in the array.

Solution

```
n=input('enter the number of rows:');
m=input('enter the number of columns:');
for i=1:n
for j=1:m
    x(i,j)=input('enter the number:');
end
end
disp('the array is:')
x
k=input('enter the number to be checked:');
l=0;
for i=1:n
for j=1:m
if (x(i,j)==k)
        l=l+1;
    disp('row:')
    i
    disp('column:')
    j
end
end
end
disp('the number of times this number appears in the matrix is:')
l
```

enter the number of rows:3
enter the number of columns:3
enter the number:2
enter the number:3
enter the number:4
enter the number:2
enter the number:5
enter the number:6
enter the number:2
enter the number:8
enter the number:2

```
the array is:
x =
     2   3   4
     2   5   6
     2   8   2
enter the number to be checked:2
row:
i =
     1
column:
j =
     1
row:
i =
     2
column:
j =
     1
row:
i =
     3
column:
j =
     1
row:
i =
     3
column:
j =
     3
```

the number of times this number appears in the matrix is:

```
l =

    4
```

15. Wirte a MATLAB program to fit a polynomial of nth degree through a given set of points.

Solution

```
n=input('enter the number of data sets:');
for i=1:n
    x(i)=input('enter the x value of data:');
    y(i)=input('enter the y value of data:');
end
M1=0;
for i=1:n
   M1=(x(i)^2)+M1;
end
M2=0;
for i=1:n
   M2=M2+x(i);
end
M3=0;
for i=1:n
   M3=M3+x(i)*y(i);
end
M4=0;
for i=1:n
   M4=M4+y(i);
end
M5=0;
for i=1:n
   M5=M5+1;
end
M6=0;
for i=1:n
   M6=M6+(x(i)^3);
end
 M7=0;
for i=1:n
   M7=M7+ (x(i)^4);
   M8=0;
end
```

```
for i=1:n
   M8=M8+(y(i)*(x(i)^2));
end
M=[M5 M2 M1;
   M2 M1 M6;
   M1 M6 M7];
L=[M4;M3;M8];
P=M\L
xfit = 0:1:30;
yfit = P(1) + (P(2).*xfit) +(P(3).*xfit.*xfit);
plot(xfit,yfit,'ro',x,y,'go');
xlabel('x-axis')
ylabel('y-axis')
title('Line Fit Using The Method of Least Square');
disp('The value of P(1) is:')
P(1)
disp('The value of P(2) is:')
P(2)
disp('The value of P(3) is:')
P(3)
```

enter the number of data sets:4
enter the x value of data:1
enter the y value of data:1
enter the x value of data:2
enter the y value of data:4
enter the x value of data:3
enter the y value of data:9
enter the x value of data:4
enter the y value of data:16

P =

0.0000
−0.0000
1.0000

The value of P(1) is:

ans =

1.9033e-014

The value of P(2) is:

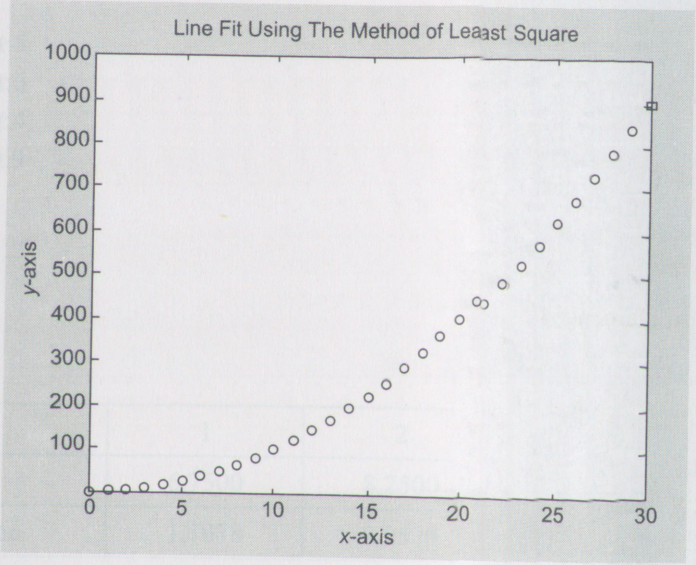
ans =

−1.8271e-014

The value of P(3) is:

ans =

1.0000



16. Write a MATLAB program to find the standard deviation of a set of points.

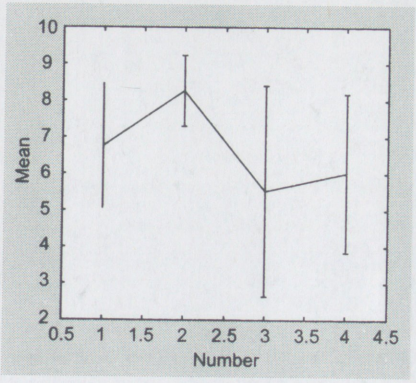
```
n=input('enter the number of elements:')
for i=1:n
    a(i)=input('enter the element:');
end

sum=0;
for i=1:n
    sum=sum+a(i);
end
mean=sum/n

s1=0;
for i=1:n
   s1=s1+(a(i)-mean).*(a(i)-mean);
end
s2=s1/(n-1);
stdv=sqrt(s2)
```

```
enter the number of elements:4
n =
     4
enter the element:5
enter the element:7
enter the element:6
enter the element:9
mean =
    6.7500
stdv =
    1.7078
>> stdr
enter the number of elements:4
n =
     4
enter the element:7
enter the element:9
enter the element:9
enter the element:8
mean =
    8.2500
stdv =
    0.9574
>> stdr
enter the number of elements:4
n =
     4
enter the element:9
enter the element:6
enter the element:5
enter the element:2
mean =
    5.5000
```

```
stdv =
    2.8868
>> stdr
enter the number of elements:4
n =
     4
enter the element:5
enter the element:6
enter the element:4
enter the element:9
mean =
     6
stdv =
    2.1602
```

sample set	1	2	3	4
mean	6.7500	8.2500	5.5000	6
standard deviation	1.7078	0.9574	2.8868	2.1602

```
>> x=[1 2 3 4];
>> y=[6.7500 8.2500 5.500 6];
>> z=[1.7078 0.9574 2.8868 2.1602];
>> errorbar(x,y,z)
>> xlabel('Number');
>> ylabel('Mean');
```



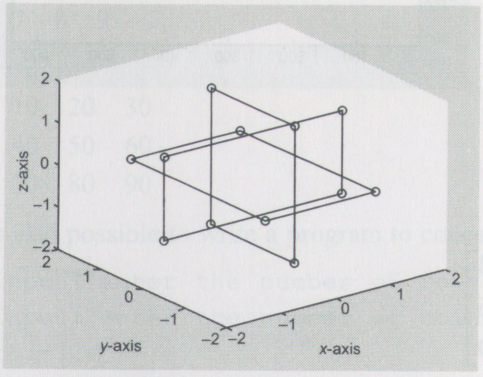
Standard deviation of the various sets of points

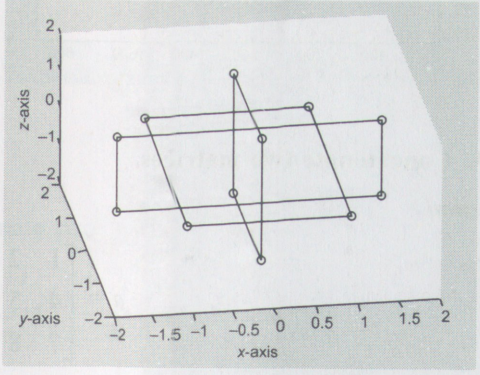
17. Given that the 12 vertices of an icosahedrons are $(0, \pm 1, \pm \varphi)$, $(\pm 1, \pm \varphi, 0)$ and $(\pm \varphi, 0, \pm 1)$. Where $\varphi = (1 + \sqrt{5})/2$. Φ is also known as the golden ratio. Show all the points and join the points corresponding to $(0, \pm 1, \pm \varphi)$, $(\pm 1, \pm \varphi, 0)$ and $(\pm \varphi, 0, \pm 1)$ forming three planes that intersect each other to form the icosahedrons.

Solution

```
x1=0;
y1=1;
z1=(1+sqrt(5))/2;

x2=0;
y2=-1;
z2=-(1+sqrt(5))/2;

x3=0;
y3=1;
z3=-(1+sqrt(5))/2;

x4=0;
y4=-1;
z4=(1+sqrt(5))/2;

x5=1;
y5=(1+sqrt(5))/2;
z5=0;

x6=-1;
y6=-(1+sqrt(5))/2;
z6=0;

x7=-1;
y7=(1+sqrt(5))/2;
z7=0;

x8=1;
y8=-(1+sqrt(5))/2;
z8=0;

x9=(1+sqrt(5))/2;
y9=0;
z9=1;

x10=-(1+sqrt(5))/2;
y10=0;
z10=-1;

x11=-(1+sqrt(5))/2;
y11=0;
z11=1;
```

```
x12=(1+sqrt(5))/2;
y12=0;
z12=-1;
plot3(x1,y1,z1,'go',x2,y2,z2,'go',x3,y3,z3,'go',x4,y4,z4,'go',x5,y5,z5
,'ro',x6,y6,z6,'ro',x7,y7,z7,'ro',x8,y8,z8,'ro',x9,y9,z9,'bo',x10,y10,
z10,'bo',x11,y11,z11,'bo',x12,y12,z12,'bo')
X1=[x1 x3 x2 x4 x1];
Y1=[y1 y3 y2 y4 y1];
Z1=[z1 z3 z2 z4 z1];
line(X1,Y1,Z1)
X2=[x5 x7 x6 x8 x5];
Y2=[y5 y7 y6 y8 y5];
Z2=[z5 z7 z6 z8 z5];
line(X2,Y2,Z2)
X3=[x9 x11 x10 x12 x9];
Y3=[y9 y11 y10 y12 y9];
Z3=[z9 z11 z10 z12 z9];
line(X3,Y3,Z3)
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
```





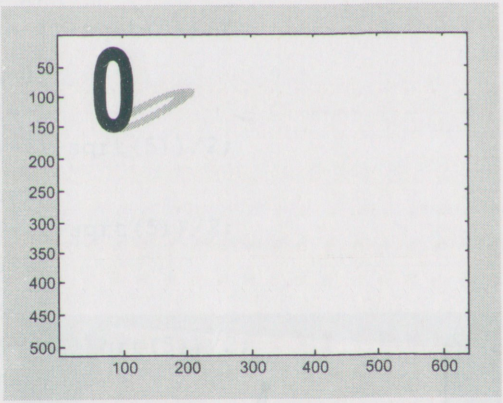
18. Write a MATLAB program such that it displays 0 when 0 is enter and display 1 when 1 is entered.

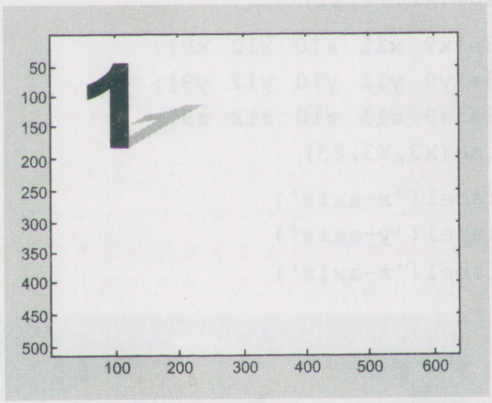
Solution

```
for i=1:100
 a=input('enter 0 or 1:');
if (a==0)
```

```
        cd d:\pictures
        i=imread('zero.jpg','jpg');
        imagesc(i)
elseif(a==1)
        cd d:\pictures
        i=imread('one.jpg','jpg');
        imagesc(i)
end
end
```

enter 0 or 1:0
enter 0 or 1:1
enter 0 or 1:





19. Concatenate two matrices.

Solution

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$b = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

The matrices could be concatenated in all the three dimensions. For this we can use the function cat(n,a,b) where n is the dimension along which concatenation has to be done and a and b are the two matrices.

```
a=[1 2 3;4 5 6;7 8 9];
b=[10 20 30;40 50 60;70 80 90];
c=cat(1,a,b)
```

```
c =

    1    2    3
    4    5    6
    7    8    9
   10   20   30
   40   50   60
   70   80   90
```

```
c=cat(2,a,b)
```

```
c =

    1    2    3   10   20   30
    4    5    6   40   50   60
    7    8    9   70   80   90
```

```
c=cat(3,a,b)
```

```
c(:,:,1) =

    1    2    3
    4    5    6
    7    8    9
c(:,:,2) =
   10   20   30
   40   50   60
   70   80   90
```

It is also possible to write a program to concatenate two matrices.

```
m=input('enter the number of rows:');
n=input('enter the number of columns:');
for i=1:m
for j=1:n
    a(i,j)=input('enter the element:');
end
end
disp('the array is:')
a

k=input('enter the number of rows:');
l=input('enter the number of columns:');
```

```
for i=1:k
for j=1:l
    b(i,j)=input('enter the element:');
end
end
disp('the array is:')
b

if (m==k)
for i=1:m
for j=(n+1):(n+l)
        c(i,j)=b(i,j-n);
end
end

elseif (n==l)
for i=1:n
for j=(m+1):(m+k)
        c(i,j)=b(i,j-m);
end
end
else
break
end

disp('the output array is:');
for i=1:m
for j=1:n
        c(i,j)=a(i,j);
end
end
    c
```

enter the number of rows:2
enter the number of columns:2
enter the element:1
enter the element:2
enter the element:3
enter the element:4

the array is:

a =

```
1  2
3  4
```

enter the number of rows:2
enter the number of columns:3
enter the element:1
enter the element:2
enter the element:3
enter the element:4
enter the element:5
enter the element:6
the array is:

b =

```
1  2  3
4  5  6
```

the output array is:

c =

```
1  2  1  2  3
3  4  4  5  6
```

20. Find the value of $\sin(x)$ using the Maclaurin's series.

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} \cdots$$

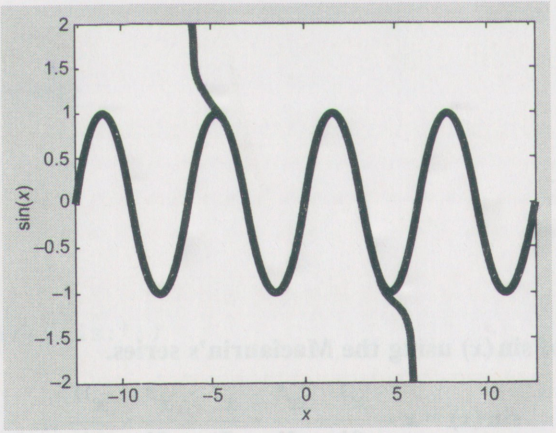
Solution

A comparison on $\sin(x)$ values obtained by using the Maclaurin's series and using sin() function is given below. We see that beyond 2π the value obtained for $\sin(x)$ using Maclaurin's series does not match with the $\sin(x)$ value. The value of $\sin(x)$ was obtained by considering the first six terms of the Maclaurin's series. It can also be seen that the values of $\sin(x)$ for low x values are missing in the plot. For lower x values it is better to consider only the first 2 or 3 terms.

```
t=input('enter the number of terms to be considered in the series:');
for x=-4*pi:0.005*pi:4*pi
sin1x=0;
count=0;
for j=1:2:2*(t)
count=count+1;
fact=1;
for i=1:j
```

```
k=i;
fact=fact*k;
k=k+1;
end
sin1x=sin1x+((-1)^(count+1))*((x^j)/fact);
end
sin1x;
axis([-4*pi 4*pi -2 2])
plot(x,sin1x,'ro',x,sin(x),'bo')
hold on
end
xlabel('x')
ylabel('sin(x)')
```

enter the number of terms to be considered in the series:6

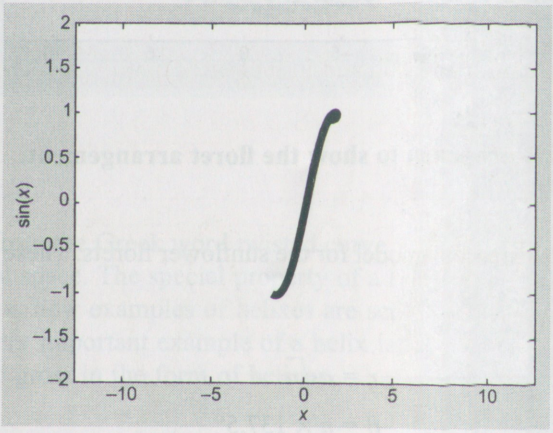


Now we try to find the value of $\sin(x)$ for x values between $-\pi/2$ to $\pi/2$.

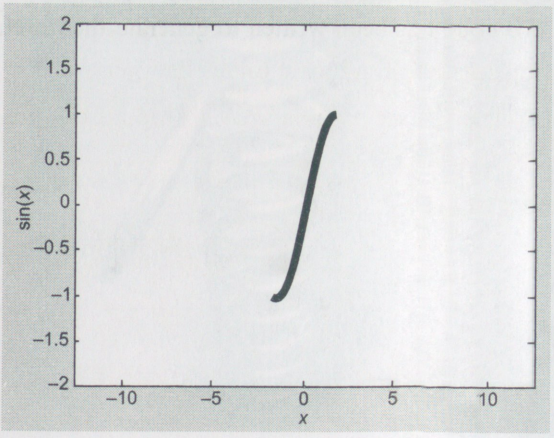
```
t=input('enter the number of terms to be considered in the series:');
for x=-pi/2:0.005*pi:pi/2
sin1x=0;
count=0;
for j=1:2:2*(t)
count=count+1;
fact=1;
for i=1:j
k=i;
fact=fact*k;
k=k+1;
```

```
end
sin1x=sin1x+((-1)^(count+1))*((x^j)/fact);
end
sin1x;
axis([-4*pi 4*pi -2 2])
plot(x,sin1x,'ro',x,sin(x),'bo')
hold on
end
xlabel('x')
ylabel('sin(x)')
```

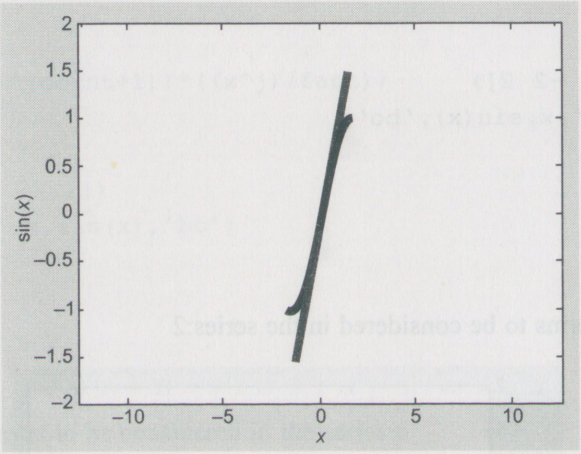
enter the number of terms to be considered in the series:2



enter the number of terms to be considered in the series:3



enter the number of terms to be considered in the series:1



21. Write a MATLAB program to show the floret arrangement.

Solution

H. Vogel proposed a mathematical model for the sunflower florets. These florets can be generated using these equations.

The two equations are:

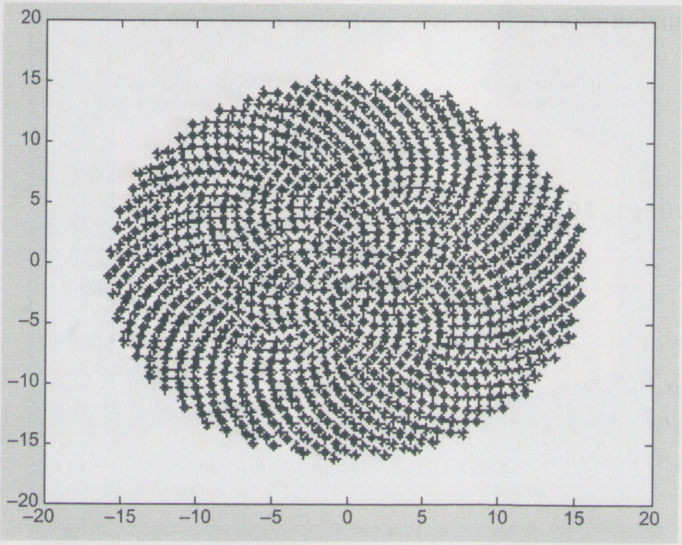
$$r = c\sqrt{n}$$

$$\theta = n \times 137.5°$$

Here θ is the angle and r the radius in polar coordinates. n has been varied for 1 to 1000 to generate all the points. c is constant known as the scaling factor.

Here a simple MATLAB code has been written to generate the floret.

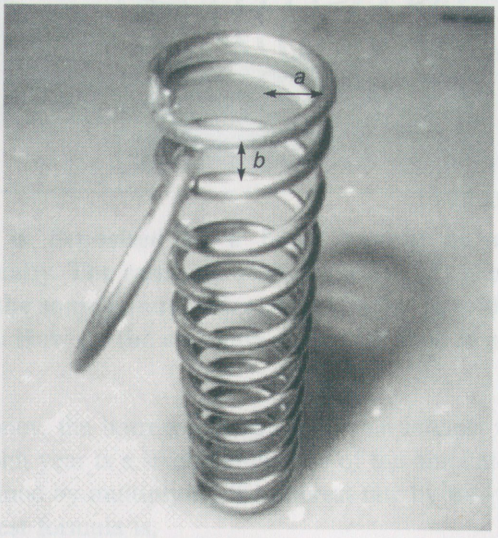
```
% generation of florets
for n=1:1000
    r=0.5*sqrt(n);
    theta=n*(pi/180)*137.5;
    x=r*cos(theta);
    y=r*sin(theta);
    plot(x,y,'k*')
    hold on
end
```



22. Draw a helix.

Solution

The word helix comes from the Greek word twisted curve. A helix is a type of space curve. It is a curve in 3-dimensional space. The special property of a helix is that it makes a constant angle with the axis of the helix. Few examples of helixes are screws, spiral staircases and the groove in a drill bit. Another very important example of a helix is the DNA molecule. There are also a number of creepers who grow in the form of helixes.

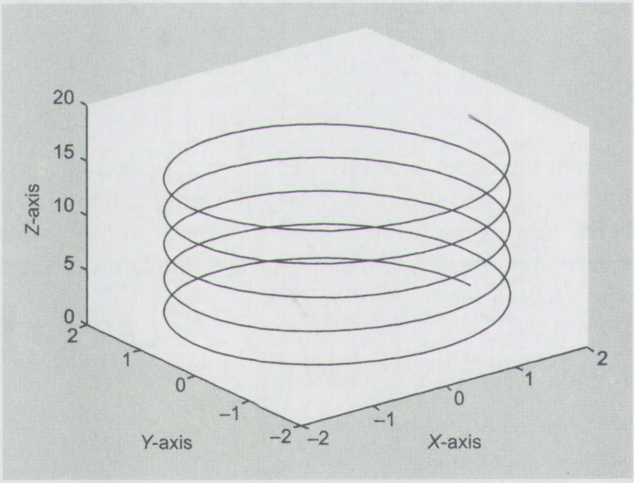


A spring

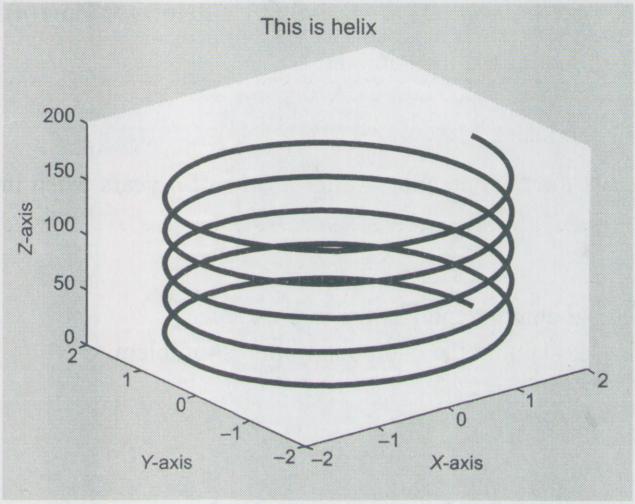
The parametric equation of a circular helix of radius a and $2\pi b$ is,

x=acos(θ)
y=asin(θ)
z= bθ

```
t=linspace(0,10*pi,1000);
x=2*cos(t);
y=2*sin(t);
z=0.5*t;
plot3(x,y,z)
xlabel('X-Axis');
ylabel('Y-Axis');
zlabel('Z-Axis');
```

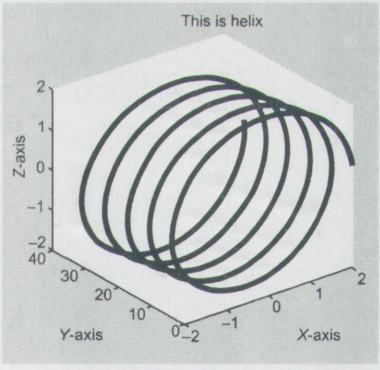



```
t=linspace(0,10*pi,1000);
x=2*cos(t);
y=2*sin(t);
z=5*t;
plot3(x,y,z,'*r')
xlabel('X-Axis');
ylabel('Y-Axis');
zlabel('Z-Axis');
title('This is a helix');
```



The axis of the helix can be changed. This has been shown below.

Program	Output	Comments

`t=linspace(0,10*pi,1000);`
`x=2*cos(t);`
`z=2*sin(t);`
`y=t;`
`plot3(x,y,z,'*r')`
`xlabel('X-Axis');`
`ylabel('Y-Axis');`
`zlabel('Z-Axis');`
`title('This is a helix');`
		Here $x = 2\cos(t)$ and $z = 2\sin(t)$ whereas $y = t$. This makes y–axis the axis of the helix.

23. An amount (P) is deposited in a bank paying an annual interest rate (r) compounded annually. The same amount of money (P) is also deposited in another bank which pays the same annual interestrate (r) but provides simple interest for the amount deposited. How do the amounts vary with time in years?

Solution

When a person lends money, the borrower usually pays an interest to the lender. The amount of simple interest paid each year is a fixed percentage of the amount that is borrowed initially. Simple interest is determined by multiplying the interest rate by the principal by the number of periods. The simple interest formula is,

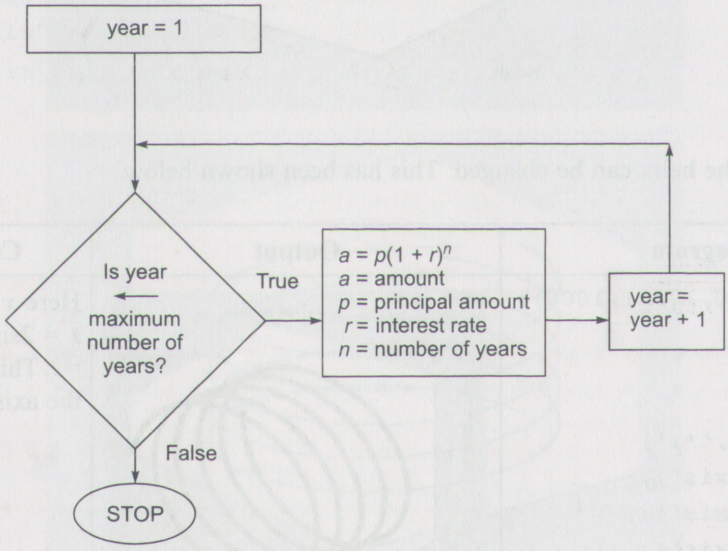
$$\text{Simple Interest } (I) = \text{Principal } (P) \times \text{Rate } (r) \times \text{Time } (n)$$

P is the principal amount or the loan amount
r is the interest rate
n is the duration of the loan

In compound interest the total amount A after a time of n years when interest is re-invested is then

$$A = P(1 + r)^n$$

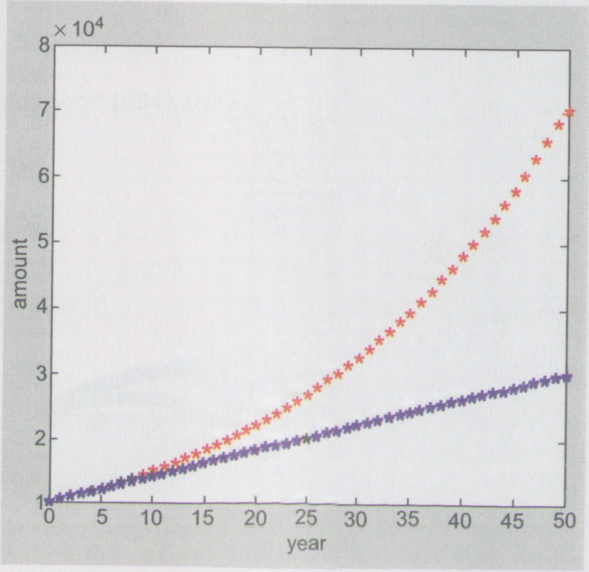
Here P is the principal amount and r is the interest rate.
Flow chart showing explaining the compound interest problem.



The MATLAB program to generate the amount in both of simple interest and compound interest is given below.

```
principalamt=10000.0;
interestrate=0.04;
for year=0:1:50
amount=principalamt*power((1+interestrate),year);
plot(year,amount,'r*')
hold on
end
xlabel('year')
ylabel('amount')

for year=0:1:50
amountsimple= principalamt+principalamt*interestrate*year;
```

```
plot(year,amountsimple,'b*')
hold on
end
```



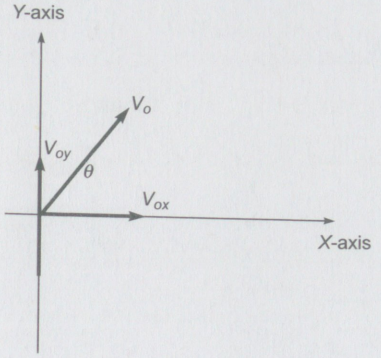
The variation of amount with time in years is shown above. The blue line shows the curve for simple interest whereas the red curve shows the curve for compound interest.

24. Projectile motion of a cricket ball. There a batsman hits a ball. The boundary is 82 m away. Does the ball clear the boundary or it drops inside the boundary? Projectile is a kind of motion in which an object which is known as the projectile (the cricket ball in this case) is thrown obliquely at an angle from the earth's surface and it moves in a curved path under the action of the gravitational force. Here the batsman hits the cricket ball at an angle θ and velocity V_o. The problem is to find the trajectory of the plot and decide the distance ball has covered to know if the hit is a SIX! or not.

Solution

$$v_{ox} = v_o \cos(\theta)$$

$$v_{oy} = v_o \sin(\theta) - gt$$



Displacement of the ball in x and y direction are given as,

$$x = v_o \cos(\theta).t$$

$$y = v_o t \sin(\theta) - \frac{1}{2} g t^2$$

Trajectory of the ball is given as,

$$y = (\tan\theta).x - \frac{g}{2v_o^2 \cos^2(\theta)}.x^2$$

```
h=input('enter the height of the batsman:')
theta1=input('enter the value of the angle of hit:')
vo=input('enter the velocity of the hit:')
t=(2*vo*sin(theta1))/9.8;
for xrange = 0:1:400
y= h+xrange* tan(theta1*pi/180) -
((9.8*xrange*xrange)/(2*vo*vo*(cos(theta1*pi/180)*cos(theta1*pi/180))));
    plot(xrange,y,'ro')
    hold on
end
grid
axis([0 200 0 200])
plot(82,0,'g*')
xlabel('Horizontal Distance')
ylabel('Height')
```

enter the height of the batsman:1.4

h =

1.4000

enter the value of the angle of hit:30

theta1 =

30

enter the velocity of the hit:40

vo =

40

enter the height of the batsman:1.4

h =

1.4000

enter the value of the angle of hit:20

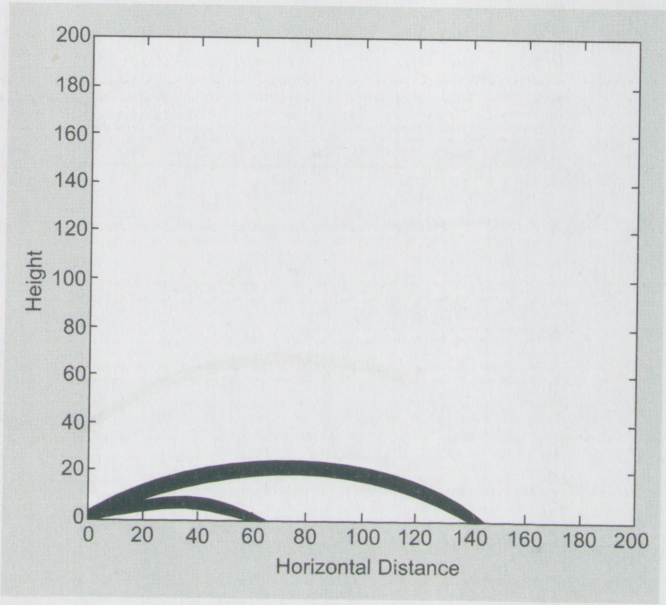
theta1 =

20

enter the velocity of the hit:30

vo =

30




```
h=input('enter the height of the batsman:')
theta1=input('enter the value of the angle of hit:')
vo=input('enter the velocity of the hit:')
t=(2*vo*sin(theta1))/9.8;
for xrange = 0:1:400
y= h+xrange* tan(theta1*pi/180) -
((9.8*xrange*xrange)/(2*vo*vo*(cos(theta1*pi/180)*cos(theta1*pi/180))));
    plot(xrange,y,'ro')
    hold on
end
grid
axis([-100 200 -200 200])
plot(82,0,'g*')
xlabel('Horizontal Distance')
ylabel('Height')
```

enter the height of the batsman:1.4

h =

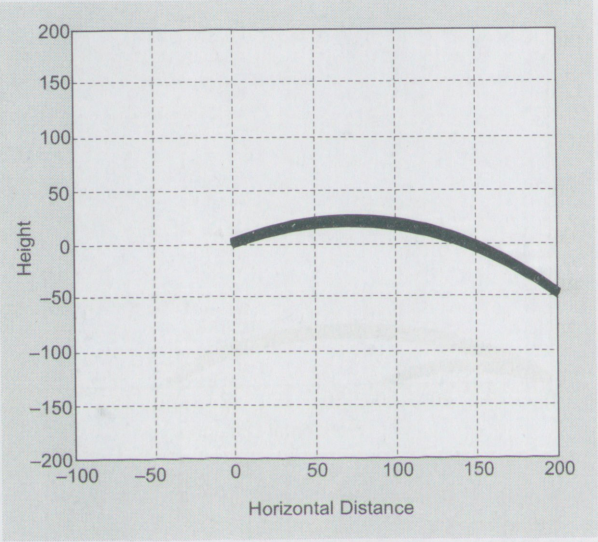
1.4000

enter the value of the angle of hit:30

theta1 =

30

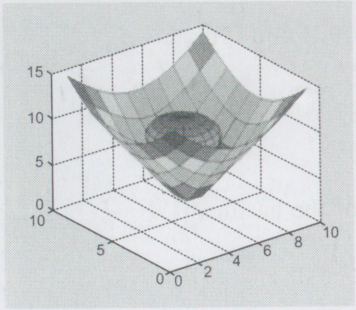
```
enter the velocity of the hit:40
vo =
    40
```



25. Consider a vase having equation $z^2 = x^2 + y^2$. Consider a spherical fruit having a radius 2 and center coordinates (4, 4, 10). Make the vase opaque and transparent by changing the value of alpha. Find out the visibility of the fruit when the vase is changed from opaque to transparent. Consider the equation of the sphere as $(x - 4)^2 + (y - 4)^2 + (z - 10)^2 = 2^2$ having a radius of 2 and center coordinates (4, 4, 10).

Solution

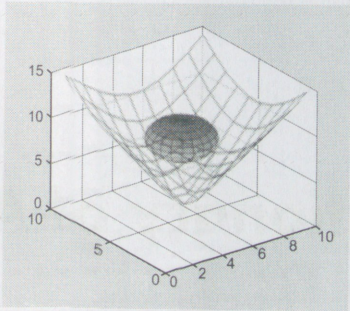
MATLAB Program	Output	Explanation

`x = linspace(-10,10,10);`
`y = linspace(-10,10,10);`
`[X,Y] = meshgrid(x,y);`
`Z = sqrt(X.*X+Y.*Y);`
`surf(Z)`
`alpha(1)`
`hold on`
`[u,v] =`
`meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);`
`a=4+2*cos(v).*sin(u);`
`b=4+2*sin(v).*sin(u);`
`c=10+2*cos(u);`
`surf(a,b,c);`
		`alpha(1)`
		So the vase is opaque.

Contd.

Contd.

```
x = linspace(-10,10,10);
y = linspace(-10,10,10);
[X,Y] = meshgrid(x,y);
Z = sqrt(X.*X+Y.*Y);
surf(Z)
alpha(0)
hold on
[u,v] =
meshgrid((0:0.1:1)*pi,(0:0.1:2)*pi);
a=4+2*cos(v).*sin(u);
b=4+2*sin(v).*sin(u);
c=10+2*cos(u);
surf(a,b,c);
```



`alpha(0)`
So the vase is transparent.

11
CHAPTER

Dictionary of Additional Terms

*"A **wiki** is a website that enables the user to add, remove or edit the content of the website using the web browser itself. This is possible due to the wiki software which is present in the server that hosts the website. The content of the website is mainly created by the visitors of the website. It is a collaborative effort of the user of the website. The wiki software was developed by Ward Cunningham. WikiWikiWeb was the first wiki website started by Ward Cunningham in 1994. He installed the wiki software in the domain c2.com in 1995. The word wiki is originally a Hawaiian word which means fast or quick. Cunnighan learnt the word when a staff at the Honolulu Airport told him to take the Wiki Wiki Shuttle to go from one terminal to the other. As Cunningham wanted to name the website as quik he used the word wiki. The word wiki was listed in the Oxford English Dictionary in 2007. Wikipedia, an encyclopedia whose content can be edited by the user, is the most well known wiki that most us have come across. A few other wikis are, AboutUs.org, Geo-wiki, LyricWiki, PlanetMath, Wikiquote and several others."*

.m, .fig, .mat and .mex

.fig - MATLAB figure file

.m - MATLAB function, script or class file

.mat - MATLAB binary file for storing variables

.mex - MATLAB executable file. It is platform specific, .mexmac for Mac operating system and .mexglx for Linux operating system.

Arithmetic Logic Unit, ALU

The arithmetic logic unit or ALU performs calculation and makes decision.

Backslash character constants (also known as escape sequences)

Backslash character constants	Output
'\n' new line (line feed) the cursor position is on the new line	>> fprintf('MATLAB\n'); MATLAB >> >> fprintf('the colour of the sky is blue\n') the colour of the sky is blue >>

Contd.

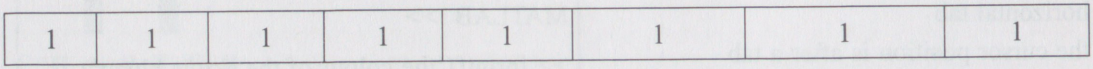
Contd.

'\t' horizontal tab the cursor position is after a tab	>> fprintf('MATLAB\t'); MATLAB >> >> fprintf('the colour of the sky is blue\t') the colour of the sky is blue >>
'\b' back space the cursor position is under the last element displayed	>> fprintf('MATLAB\b'); MATLA>> >> fprintf('the colour of the sky is blue\b') the colour of the sky is blue>>
'\r' carriage return the cursor position is under the first element displayed	>> fprintf('MATLAB\r'); MATLAB >> >> fprintf('the colour of the sky is blue\r') the colour of the sky is blue >>
'\"' double quote shows a double quote in the end	>> fprintf('MATLAB\"'); MATLAB">> >> fprintf('the colour of the sky is blue\"') the colour of the sky is blue">>
'\\' Backslash	>> fprintf('MATLAB\\') MATLAB\>> >> fprintf('the colour of the sky is blue\\') the colour of the sky is blue\>>
'%%' Percent character	>> fprintf('MATLAB%%') MATLAB%>> >> fprintf('the colour of the sky is blue%%') the colour of the sky is blue%>>

Bits and Bytes

A bit is the short for binary digit which is the smallest unit of information in a computer. This term was first used by John Tukey in 1964. John Tukey was a leading statistician and adviser to five American presidents. A single bit can hold only values 0 or 1. Large numbers can be obtained by combining consecutive bits into larger units. For example, a byte is composed of 8 consecutive bits. A byte is a sequence of bits and eight bits make one byte. All pieces of digital information are encoded in 0 or 1.

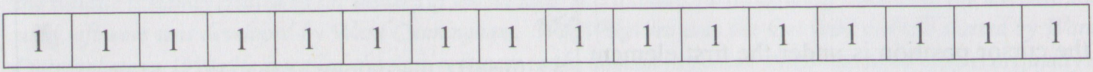
So if there are 8 bits in a byte the bits could all be filled with 0 or all could be filled with 1. If all are filled with 1 then it becomes,



$2^8 - 1 = 255$

and the decimal number for this is 255. This is the maximum representable number. So if there are 8 bits in a byte we can represent 0 to 255 using this byte. Note that a signed character has a range of –128 to 127 and unsigned characters have a range of 0 to 255. This is because the first bit is used for notifying the sign of the number and as result only 7 bits are left for storing the number making it $(2^7 - 1)$.

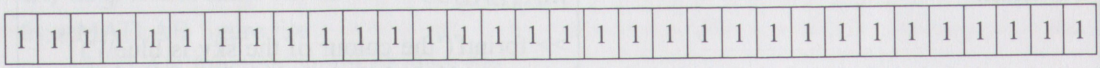
Similarly, if we have two bytes or 16 bits the maximum number that can be represented is,



$2^{16} - 1 = 65535$

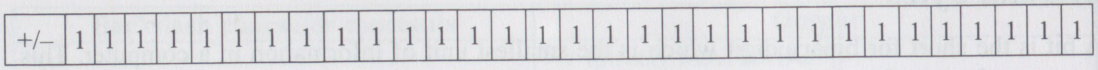
The maximum number that can be represented is 65535. For example, if the system we are working on has a variable is only 2 bytes or 8 bits long then the permissible limits of signed numbers is –32768 to +32767. Short integers are 2 bytes or 16 bits in size therefore the numbers in this format can vary from –32768 or 32767, this is the same range as that of integer variables.

The unsigned short integer has a range of 0 to 65535 and the unsigned long integer variable has the range of 0 to 4294967295.



$2^{32} - 1 = 4294967295$

If the integer has 32 bits in all then the maximum signed integer that can be represented by the 4 bytes is 2147483647. Signed long integers are 4 bytes or 32 bits in size and can therefore have a range of –2147438648 to 2147438647. This is also the size of the usual integer variable which can hold 4 byte size numbers.



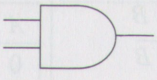
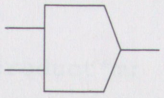
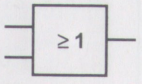
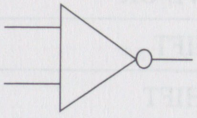
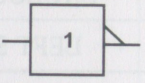
$2^{31} = 2147483648$

Boolean algebra

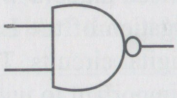
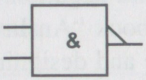
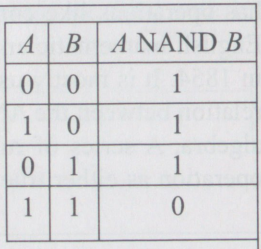
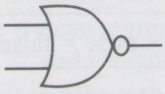
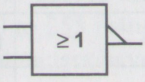
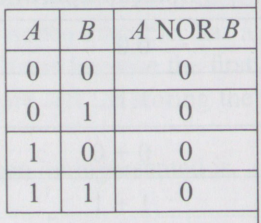
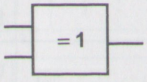
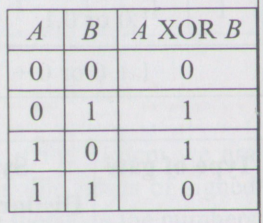
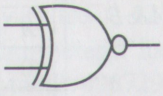
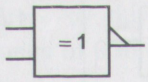
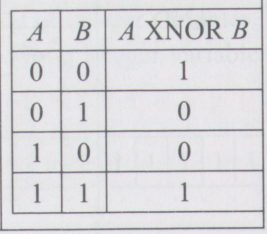
Boolean algebra is a subject of mathematics in which the values of variables are either true or false(truth values). True is denoted as 1 and false is denoted as 0. Unlike algebra Boolean algebra

has operations like conjunction, disjunction and negation. Boolean algebra was introduced by English mathematician, George Boole in his book "An Investigation of the Laws of Thought" in 1854. It is mostly used in computer science and designing digital circuits. The knowledge of relation between the AND gate, OR gate and NOT gate is very important to understand Boolean algebra. A series of rules as formulated by George Boole decides the outcome of a Boolean operation as either true or false.

Boolean operation	Result	Explanation
0.0	0	false × false = false
1.1	1	true × true = true
0 + 0	0	false + false = false
1 + 1	1	true + true = true
1.0 or 0.1	0	true × false or false × true = false
1 + 0 or 0 + 1	1	true + false or false + true = true

Type of gate	Symbol/ Distinctive shape	Rectangular shape	Boolean algebra between A and B	Truth table
AND			$A.B$ or $A\&B$	A | B | A AND B 0 | 0 | 0 1 | 0 | 0 0 | 1 | 0 1 | 1 | 1
OR			$A + B$	A | B | A OR B 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1
NOT			$\bar{A}$ or $\sim A$	A | NOT A 0 | 1 1 | 0

Contd.

NAND			$\overline{A.B}$ $A\|B$	
NOR			$\overline{A+B}$ $A-B$	
XOR			$A \oplus B$	
XNOR			$A \odot B$ $\overline{A \oplus B}$	

A	B	A NAND B
0	0	1
1	0	1
0	1	1
1	1	0

A	B	A NOR B
0	0	0
0	1	0
1	0	0
1	1	0

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

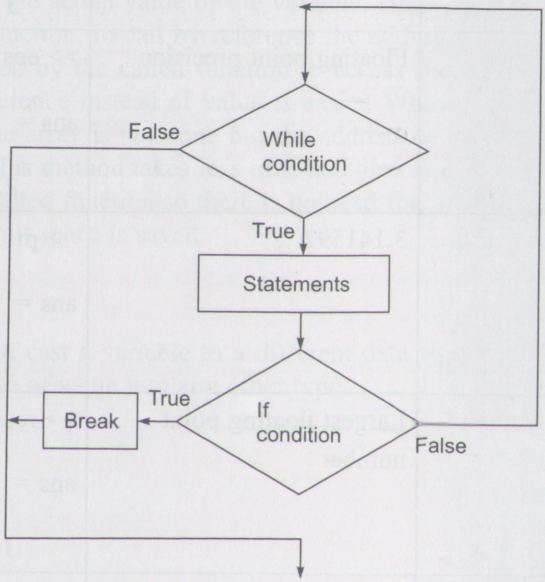
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Bitwise operator

&	AND
|	INCLUSIVE OR
^	EXCLUSIVE OR
<<	LEFT SHIFT
>>	RIGHT SHIFT
~	ONE'S COMPLIMENT

break

Write a program that determines whether a number n is prime, that is, whether it can be divided by any number except itself or 1 without bearing a remainder. To check whether a number is prime or not it is divided successively by all numbers from 2 to 1 less than itself. If the remainder of any of these divisions is zero, the number isn't a prime.



Program	Output

`n=input('enter the value of n:')`
`m=1;`
`product=1;`
`while m<=n`
`  product=product*m;`
`  m=m+1;`
`if product>=100`
`  disp('Sorry product is greater than 100!')`
`break`
`end`
`end`
`product`
	enter the value of n:5
	
	n =
	
	5
	
	Sorry product is greater than 100!
	
	product =
	
	120

Built-in MTALAB variables

The values assigned to the various built-in variables are listed in the Table below.

Name of the variable	Value	Note
ans	Value of expression	>> 2+1 ans = 3
eps	Floating point precision	>> eps ans = 2.2204e-016
pi	3.141592…	>> pi ans = 3.1416
realmax	Largest floating point number	>> realmax ans = 1.7977e+308
realmin	Smallest floating point number	>> realmin ans = 2.2251e-308
Inf	Infinity (∞), a number larger than realmax	>> 1/0 ans = Inf
NaN	Not a number	>> NaN ans = NaN

Call-by-value and Call-by-reference

A variable having a single value is sent to the variable of the called function by sending a copy of the value of the variable. The value is copied in a local variable in the function. This is known

as call by value. In call by value, the value that is stored in the memory is sent. When a calling function sends the value of the variable to the called function the value to the variable is copied to another variable name and the called function makes use of this value. The original variable in the calling function is not accessed by the called function.

On the other hand sending a variable in which the address of the variable of the calling function is sent to the called function is known as call by reference. Only the reference to the variable is sent and not the actual value of the variable. Here the pointer to the data is copied and sent to the called function. In call by reference the address where the variable is stored is sent. This address is used by the called function to access the variable. In the case of arrays sending an array by reference instead of value is easier. When the variable is an array then a copy of the values of the array is not done but the address of the array in the memory is sent to the called function. This method takes less time and also needs less space. As the address of the array is sent to the called function so there is no need for making a copy of the array in the called function. As a result space is saved.

cast

cast is a function used to cast a variable to a different data type. Using the cast operator it is possible to force any type of value into any other type.

For example,

```
>> c = int64(22);
>> d = cast(c,'uint32');
>> class(d)
ans =
uint32
```

The above command will cast the variable c to d having class unit32.

Program	Output	Comment
>> a = single(10); >> b = cast(a,'double'); >> class(b)	ans = double	Here a single precision number is cast into a double precision number. >> whos a Name Size Bytes Class Attributes a 1 × 1 4 single >> whos b Name Size Bytes Class Attributes b 1 × 1 8 double

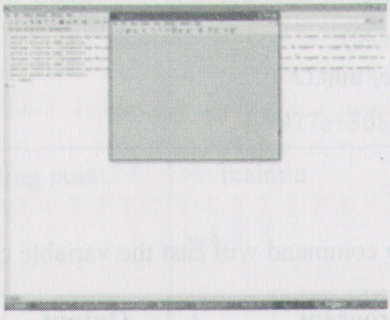
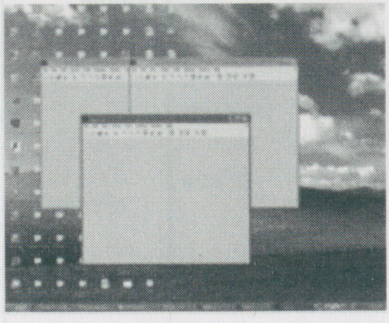
Thus, it is also possible to cast the result of an expression into a desired type of data.

Charles Babbage

Charles Babbage (26th December 1791–18th October 1871) was a mathematician, philosopher and also a mechanical engineer. He is the creator of programmable computer. He is considered father of computing. He invented the first mechanical computer. The original machine created by Babbage is now in the London Science Museum. Babbage on July 14th 1822 presented a paper entitled "Note on the application of machinery to the computation of astronomical and mathematical tables" to the Royal Astronomical Society. He called this the "difference machine." The machine was able to calculate polynomials by using a numerical method known as the differences method. In 1823 the Society approved the idea presented by Charles Babbage and the government granted him £1500 to build the machined he designed.

close

close deletes the current figure or the specified figure(s).
close name deletes the figure with the specified name.
close all deletes all figures whose handles are not hidden.

MATALB Command	Explanation
>> figure >> close	 One figure window is opened. The figure window is closed.
>> figure >> figure >> figure >> close all	 Three figure windows are opened. All three figure windows are closed.

Colormap

Visualization of scientific data can be done efficiently by mapping scalar values to colours. In this process scalar fields are viewed by coloring surfaces and volumes. Colormaps are also known as color tables. A colormap is an *m*-by-3 matrix of real numbers lying between 0.0 and 1.0. These numbers indicate the intensity of the particular color.

	R	G	B
1			
2			
3			
4			
...			
...			
k			

To create your own colormap we need to define a 2-dimensional array of dimension $n \times 3$, where n represents the number of colors. The second dimension represents the RGB values. Here a colormap has been created. The RGB array is,

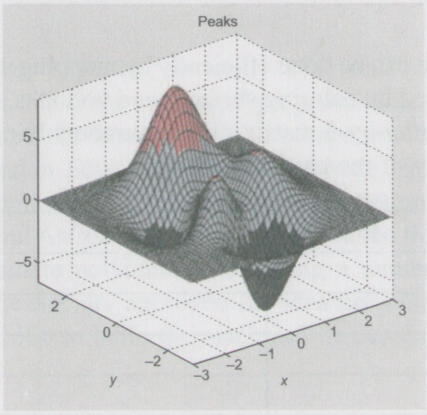
0.1	0.2	0.3
0.6	0.7	0.8
0.9	0.2	0.5

The RGB array is entered into matrix A,

```
>> A

A =

    0.1000    0.2000    0.3000
    0.6000    0.7000    0.8000
    0.9000    0.2000    0.5000
```

Here we plot the peaks function to using the defined colormap.

```
>> A=[0.1 0.2 0.3;0.6 0.7 0.8;0.9 0.2 0.5];
>> peaks
z = 3*(1–x).^2.*exp(–(x.^2) - (y+1).^2) ...
–10*(x/5 – x.^3 – y.^5).*exp(–x.^2–y.^2) ...
–1/3*exp(–(x+1).^2 – y.^2)
>> colormap(A)
```



Each row is an RGB vector that defines one color. The value in each cell defines the (k,:) = [r(k) g(k) b(k)]) which specifies the intensity of red, green and blue. K is the row and (k,:) = [r(k) g(k) b(k)]) is the *k*th row of the colormap or the *k*th color

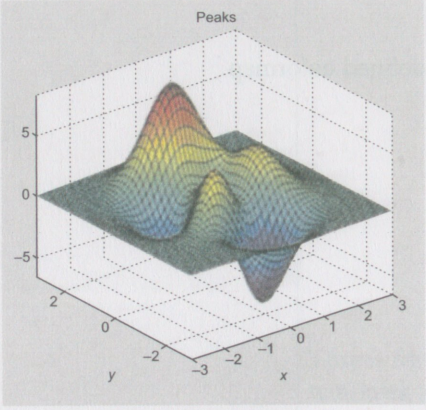
Thus, a colormap can be defined by the user.

colormap('default') sets the current colormap to the default colormap.

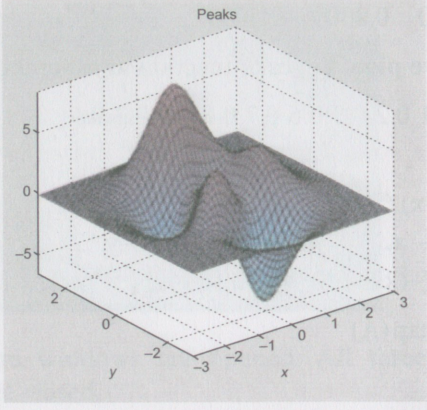
colormap(cool) sets the colormap to the matrix cool.

cmap = colormap retrieves the current colormap.

```
>> peaks

z =  3*(1–x).^2.*exp(–(x.^2) – (y+1).^2) ...
  – 10*(x/5 – x.^3 – y.^5).*exp(–x.^2-y.^2) ...
  – 1/3*exp(–(x + 1).^2 – y.^2)

>> colormap(jet)
>> colormap(cool)
>> view(2)
>> colormap(cool)
```

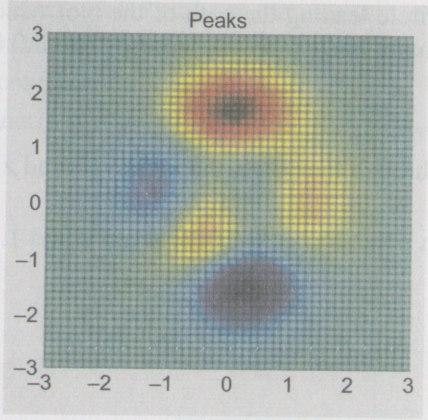


colormap(jet)

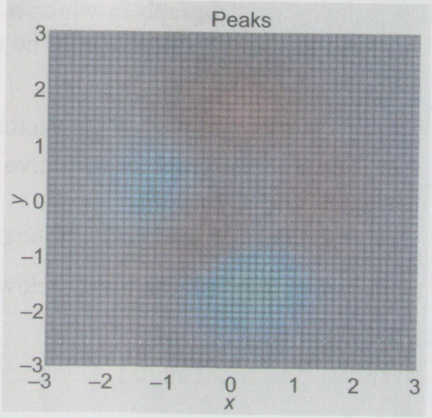


colormap(cool)

view (2) gives a 2-dimensional view of the plot.

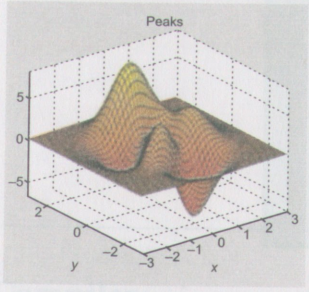


colormap (jet)

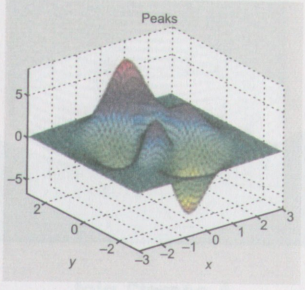


colormap (cool)

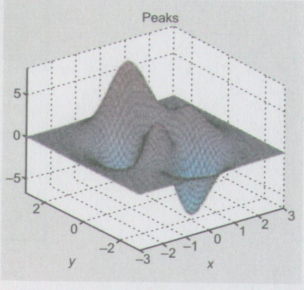
The various colormaps that are available in MATLAB are Jet, HSV, Hot, Cool, Spring, Summer, Autumn, Winter, Gray, Bone, Copper, Pink and Lines. Few of the colormaps are shown below.



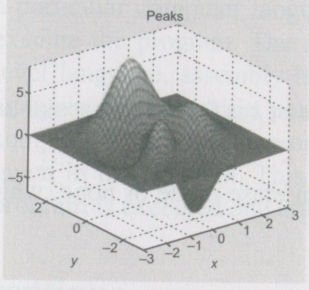
>> colormap (autumn)



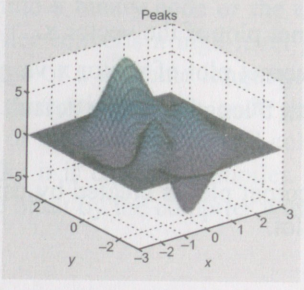
>> colormap (HSV)



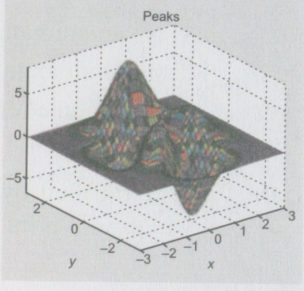
>> colormap (cool)



>> colormap (bone)



>> colormap (winter)



>> colormap (lines)

Comet Plot

A comet plot is an animated graph in which a circle leading the path of the plot which stands for the comet head traces the data points. The comet body follows the head. The tail is a solid line that traces the function.

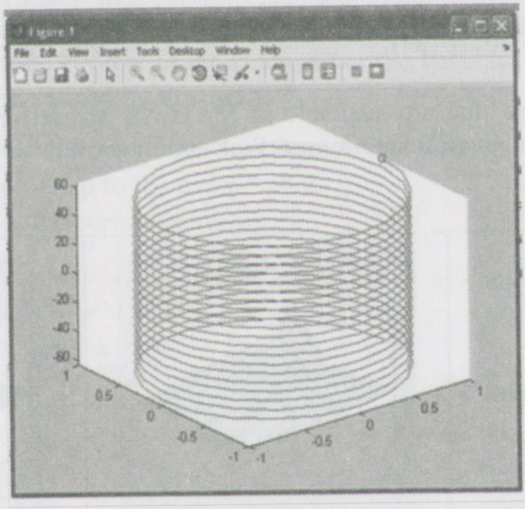
comet3 (z) plots a 3-D comet plot of the vector z.

comet3 (x,y,z) plots a comet graph of the curve through the points the x (i) , y (i) and z (i) points defined by the equation of the path.

comet3 (x,y,z,p) specifies a comet body of length p*length (y). p lies between 0 and 1.

An example of a comet3 plot is given below.

```
angle = -20*pi:0.1*pi:20*pi;
comet3 (cos (angle),sin (angle),angle)
```



A comet3 plot

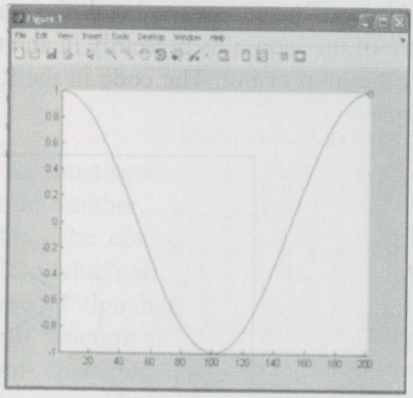
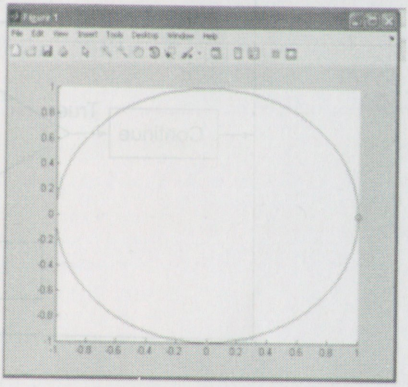
For 2-dimensional plots,

comet (x) displays an animated comet plot of the vector x.

comet (x, y) displays an animated comet plot of vector x vs. y.

comet (x, y, t) uses a comet of length t*length (y). The default value of t is 0.10. t is a scalar and lies between 0 and 1.

Here, that the trace left by the comet plot is created by using an erase mode of none, which implies that you cannot print the plot.

Program	Output
>> t=0:0.01*pi:2*pi; >> x=cos(t); >> comet(x)	
>> t=0:0.01*pi:2*pi; >> x=cos(t); >> y=sin(t); >> comet(x,y)	

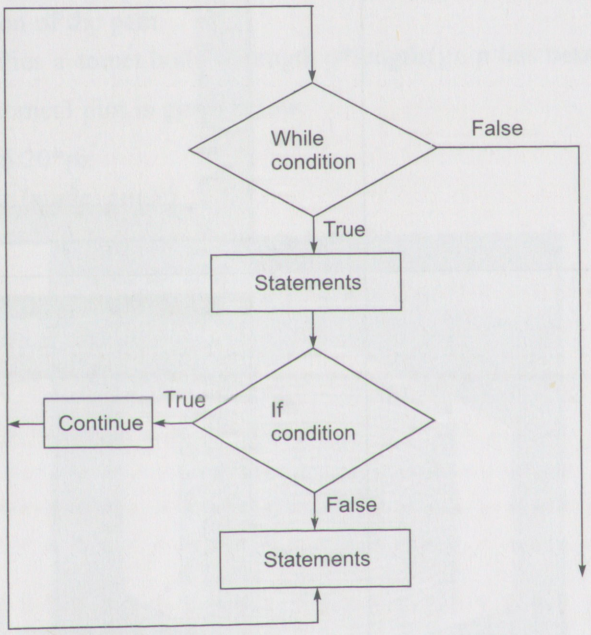
Compiler

A compiler is a computer program or a set of programs that converts the source code written in a particular computer language into a binary code or the object code. This is done in order to execute the program. The compiler converts a high-level programming language into a lower level language, assembly language or a machine language. A compiler does tasks like lexical analysis, preprocessing, parsing, semantic analysis, code generation and code optimization. The compiler looks into the program or the source code and collects and recognizes the instructions given in the program. This is why it is called a compiler. Each line of the source code or the program is executed one after the other by the compiler without looking into the entire program.

continue

MATLAB provides "break" and "continue" keywords to allow tighter loop control. The continue statement is used to pass control to the next iteration of a for or while loop. In MATLAB the

continue statement works somewhat like the break statement. Instaed of forcing termination of the loop the continue statement forces the next iteration of the loop to take place, skipping any code in between. In contrsast to break the continue statement causes the program to return back to the start of the loop it is presently in and to recheck the condition to see if it should continue executing the loop or not. The code in the loop after the continue statement is not executed in the same pass.



An example using continue statement is given below.

```
for k=0:1:10
if (k^3<1000)
continue
end
    disp('the value of k is')
    k
    disp('the value of k^3 is')
    val=k^3
end
```

the value of k is

k =

10

the value of k^3 is

```
val =
     1000
```

Conversion character

When we want to display or print an output then we must specify the type of data we want to display. The data could be an integer, floating point number, character, string, octal number, hexadecimal number or any other type of variable. The conversion character in the sprintf command specifies the type of data and the format in which we want to display the data. The conversion character which consists of a % sign and an alphabet which stands for the variable type we want to display or input. We use the sprintf function to display an output. Depending on the type of output we have to use a conversion character. The conversion character gets replaced by the value that we want to display. The program below displays the use of different conversion characters.

```
>> sprintf('%d\n %f\n %c\n %s\n %o\n %x\n %X\n %e\n %E\n',22,3.1415,'c','a
bc',111,111,111,500.555,500.555)
ans =
22
3.141500
c
abc
157
6f
6F
5.005550e+002
5.005550E+002
>> sprintf('%i\n %g\n %u\n',-22,1.1234,333)
ans =
-22
1.1234
333
>> sprintf('%d\n %f\n %c\n %s\n %o\n %x\n %X\n %e\n %E\n %g\n %i\n %u\n',22,3.1415,'c
','abc',111,111,111,500.555,500.555,3.3333333,-28,222)
ans =
22
3.141500
```

```
c
abc
157
6f
6F
5.005550e+002
5.005550E+002
3.33333
-28
 222
>> sprintf('%G\n %g\n',111.222,111.222)

ans =
    111.222
    111.222
```

The list of various conversion characters

Conversion characters	Variable type	An Example of Output
%c	Single character	'c'
%d	Decimal notation, signed integer	22
%e	Exponential notation using a lowercase e floating point number with exponent using small e	5.005550e+002
%E	Exponential notation using an uppercase E floating point number with exponent using capital E	5.005550E+002
%f	Fixed-point notation, floating point number without exponent	3.141500
%g	The more compact of %e or %f, Insignificant zeros are not printed. floating type number using wither e-type or f-type conversion, depending on which is shorter	1.1234
%i	signed integer	–22
%o	Octal notation (unsigned), octal integer with no leading 0	157

Contd.

Contd.

%s	String of characters	'abc'
%u	Decimal notation (unsigned), unsigned integer	333
%x	Hexadecimal notation (using lowercase letters a-f) unsigned hexadecimal integer using lowercase letters a-f with no leading 0 or X	6f
%X	Hexadecimal notation (using uppercase letters A-F) unsigned hexadecimal integer using uppercase letters A-F with no leading 0 or X	6F
%G	Same as %g, but using an uppercase E	111.222

Cubic Equations

The method proposed by Girolamo Cardano for solving cubic equations is given below.

$$ax^3 + bx^2 + cx + d = 0$$

This is a method which gives all the roots when all the three roots are real. Cubic equations have to be solved in several steps. First we find variable 'f':

$$f = \frac{(3c/a) - (b^2/a^2)}{3}$$

Next we find g:

$$g = \frac{(2b^3/a^3) - (9bc/a^2) + (27d/a)}{27}$$

Then we find h:

$$h = (g^2/4) + (f^3/27)$$

If $h > 0$, there is only 1 real root and is solved by another method. For the special case where $f = 0$, $g = 0$ and $h = 0$, all 3 roots are real and equal. This method is applicable for all three roots being real.

When $h <= 0$, as is the case here, all 3 roots are real and we proceed as follows:

Considering that all 3 roots are real

$$i = ((g^2/4) - h)^{1/2}$$

$$j = (i)^{1/2}$$

The following trigonometric calculations are in radians

k = arc cosine $(-(g/2i))$
$L = j * -1$
M = cosine $(K/3)$
N = (Square Root of 3) * sine $(K/3)$
$P = (b/3a) * -1$
$x_1 = 2j *$ cosine $(k/3) - (b/3a)$
$x_2 = L * (M + N) + P$
$x_3 = L * (M - N) + P$

x_1, x_2 and x_3 are the three roots.

The MATLAB program which implements the method proposed above is given below.

```
% Cubic equation of the form: ax^3+bx^2+cx+d=0
% Here a,b,c and d are the coefficients of the cubic equation.
a= input('enter the value of a:');
b= input('enter the value of b:');
c= input('enter the value of c:');
d= input('enter the value of d:');
f=((3*c/a)-(b^2/a^2))/3;
g=(((2*b^3)/(a^3))-((9*b*c)/(a^2))+((27*d)/a))/27;
h=((g^2)/4)+((f^3)/27);
if(h>0)
disp('there is only one real root');
else
i=sqrt(((g^2)/4)-h);
j=power(i,1/3);
k=acos(-g/(2*i));
l=j*-1;
m=cos(k/3);
n=sqrt(3)*sin(k/3);
p=(b/(3*a))*-1;
disp('the three roots are:')
x1=2*j*cos(k/3)-(b/(3*a))
x2=l*(n+m)+p
x3=l*(m-n)+p
end
```

enter the value of a:1
enter the value of b:–6
enter the value of c:11
enter the value of d:–6

the three roots are:

```
x1 =

     3

x2 =

     1

x3 =

    2.0000
```

disp ()

disp (a) displays an array. The array name is not printed. A could also be a text string. In this case the text string is displayed. The disp function accepts only one input. The disp function does not display empty arrays. A few examples are illustrated below,

```
>> c=[1 2 3;4 5 6];
>> disp(c)
   1   2   3
   4   5   6
>> disp('this is the correct answer');
this is the correct answer
>> d=[];
>> disp(d)
>> disp('');
```

Determinant, det()

Determinants are very useful for finding solution to a system of liner equations. It has been shown in Cramer's' rule that the solution a nonhomogeneous system of linear equations has a unique solution only if the determinant of the system's matrix is nonzero. If the determinant of a matrix is 0 such a matrix is called a singular matrix. In order to have a unique solution the matrix has to be nonsingular. Determinants are defined only for square matrices.

The determinant of the matrix,

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

is,

$1x(2x3 - 2x3) - 1(2x3 - 2x3) + 1(2x3 - 3x2)$

$= 0$

det(a) returns the determinant of a. For example,

```
>> a=[1 1 1;2 2 2;3 3 3];
>> det(a)

ans =

     0
```

Equality operator (==)and assignment operator (=)

==tests the relationship if the first operand is equal to the second operand. A command using == operator could be,

```
If(a==100)
```

The above statement test if the value of the left operand is equal to that of the light operand. Here the above statement checks if a is equivalent to 100. On the other hand,

```
a=100;
```

assigns the value 100 to the variable a == is used to compare primitive data types like character, integer or floating point etc. whereas = operator is used to assign values to a variable. The output of if(a==100) is true if the data types are equal. == operator can also be used for arrays and strings but in this case the addresses of elements of the arrays and strings are compared. Their location in the memory are compared and not their values. Here the difference in the use of == and = in an if loop is shown below.

Program	Output	Comment

a=50;
if a==100
disp('You win!');
else
disp('You loose!');
end
	You loose!
	>>
		Here a has been initialized to 50 by the statement a=50 so the condition if(a==100) is not true and hence "You loose!" is the output.

Floating point number

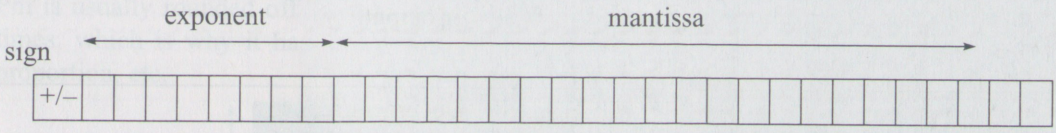
4 bytes are used to store a floating point number. As 1 byte is made of 8 bits so 4 bytes have 32 bits. In a floating point number there are two parts one is the exponent and the other mantissa.

$$\text{Floating point number} = \text{mantissa} \times (\text{base})^{(\text{exponent})}$$

For example, in a number line $2.134567 = 2134567 \times 10^{-6}$ the mantissa is 2124567 and the exponent is 10^{-6}.

Floating point number	Mantissa	Exponent
2.134567 =	2124567 ×	10^{-6}

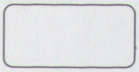
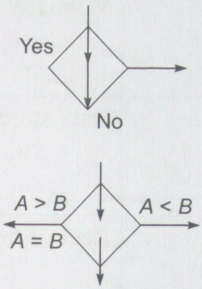
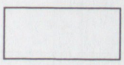
Similarly, a floating point number stored in C also has two components mantissa and exponent. Any floating point number is stored as a binary number consisting of 1 and 0 in an array of maximum 32 bits.



Here 1 bit is reserved for sign of the number, 8 bits for the exponent and 23 bits of the mantissa.
The maximum value of a floating point variable is $3.402823466 \times (10)^{38}$
The minimum value of a floating point variable is $1.175494351 \times (10)^{-38}$

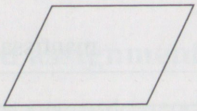
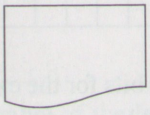
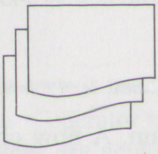
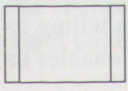
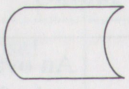
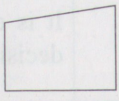
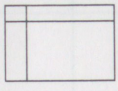
Flow chart

Flow chart mainly helps in designing a program. A flow chart is a diagram that tells us the sequential steps that has to be undertaken in a process. It in fact represents the algorithm or the process itself. The boxes are connected by arrows telling us which step to execute next. A flow chart helps us to visualize the entire process. This enables us to understand the process and write the code for executing the process.

Name	Symbol	Function and comments
Start/End		An oval shape that represents the start or end of a process.
Decision making step		It is diamond in shape and represents a decision making step.
Processing step		It is rectangle in shape and it represents a process.

Contd.

Contd.

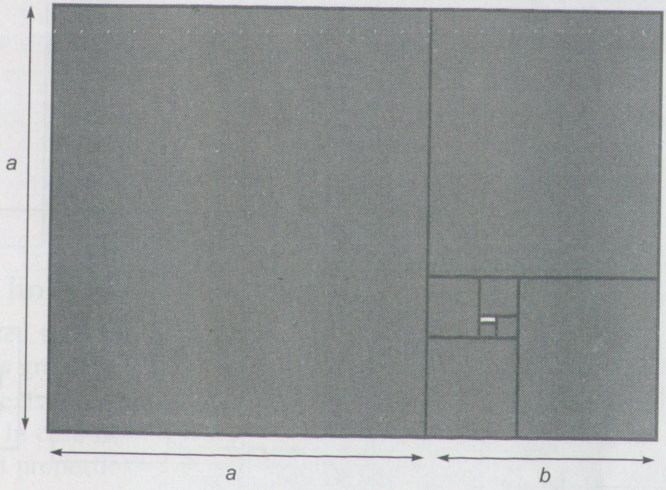
Flow of control or logic, the next step to be taken.		A solid arrow acts as a connector. It connects two steps. It shows the subsequent step that needs to be taken. It also shows the relationship between two representative shapes that it connects.
Data Input/Output		A parallelogram that represents data input or output.
Tagged Document	 or 	Document/report or multidocument. Document sign is used for hard copy input or output.
Predefined process, processes defined elsewhere		A predefined sub-process. The symbol for process ( ) is very similar to the predefined process ( )
Online storage, stored data		Input/output using online magnetic storage. A general data storage symbol.
Manual input		Manually input into the system
Internal storage		Data stored in the memory
Sequential access storage or magnetic tape		Reel of tape, magnetic tape for storage
Disk storage, Magnetic disk		A database

The Golden ratio, for loop

The Golden ratio is a special number found by dividing a line into two parts so that the longer part divided by the smaller part is also equal to the whole length divided by the longer part. It is often symbolized using phi, after the 21st letter of the Greek alphabet. In an equation form, it looks like this:

$$a/b = (a + b)/a = 1.6180339887498948420 \ldots$$

Phi is usually rounded off to 1.618. This number has been discovered and rediscovered many times, which is why it has so many names — the Golden mean, the Golden section, divine proportion, etc.



This is amazing ratio is also equal. The representation of the Golden ratio in terms of a nested radical is,

$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1}}}}}}}} \ldots$$

The representation of the Golden ratio in terms of continued fraction is,

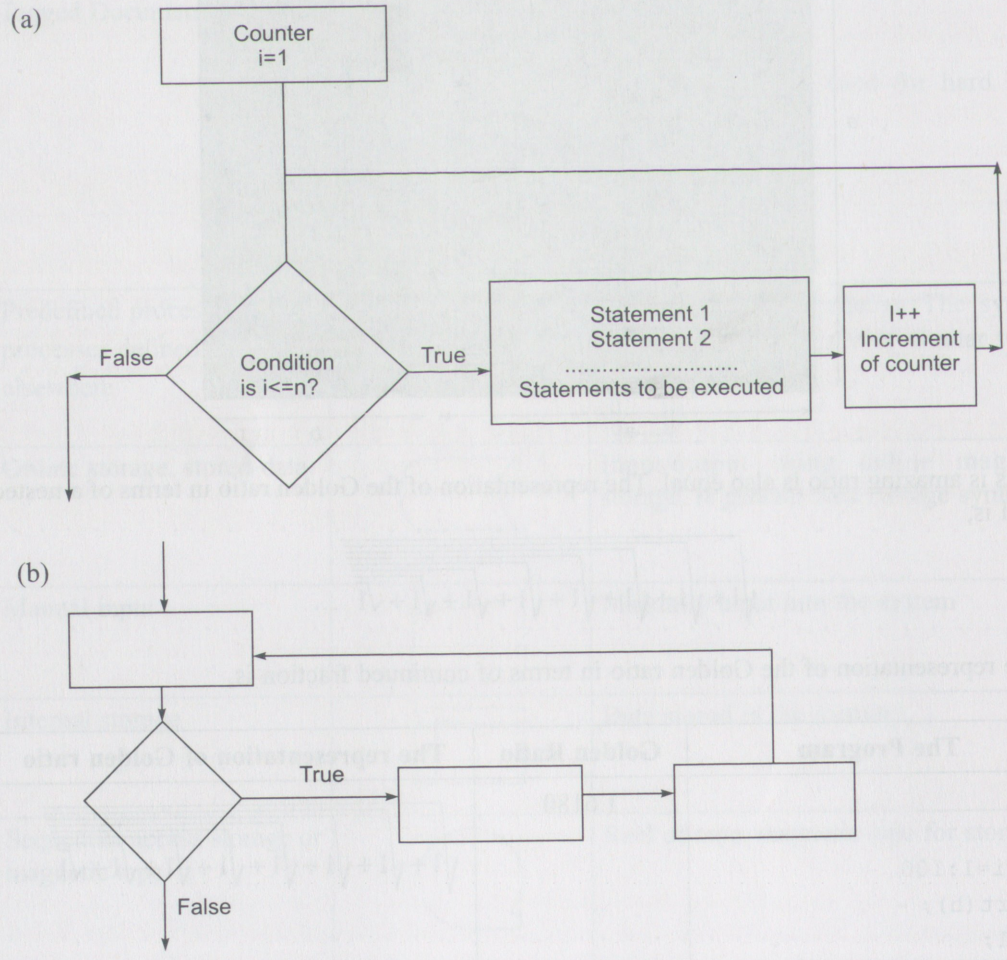
The Program	Golden Ratio	The representation of Golden ratio

h=1;
i=1;
for i=1:100
g=sqrt(h);
h=g+1;
end
disp(sqrt(h))
	1.6180	$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1}}}}}}} \ldots$

Contd.

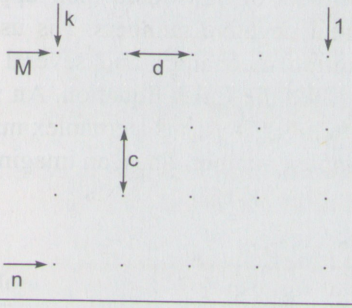
Contd.

a=2;
for i=1:100
c=(1/a);
d=c+1;
a=d;
end
disp(a)
	1.6180	$1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{\ddots}}}}}}$



The for loop structures

The various types of for loop are listed below.

Command	Points obtained	Comments
for i=m:1:n	.....	Here the increment is 1. A linear array of points is created.
for i=m:c:n		Here the increment is c. The points are sparse is the value of c is large. A linear array of points is created.
for i=m:c:n for j=k:d:l		Here two for loops are used and as a result a two-dimensional array is obtained.

hold, hold on, hold off

hold function is used to determine a new figure which is to be added to the already existing graph or replace the graph.

hold on retains the existing plot and certain axes properties so that subsequent plots are added to the existing graph. In case no axes are present before the hold on command then new axes are created with default properties. Axes properties may be changed to fit in new plots.

hold off on the other resets the already existing axes properties to their default before drawing new plots. hold off is by default.

Infinity

Infinity (∞) represents something which does not have limit. Infinity finds its application in mathematics and physics. The word "infinity" comes from the Latin word "infinitas" which means unboundedness. The reciprocal of an infinitesimal is infinity. Infinity is greater than any real number that can be imagined. Infinity is endless, something that never ends. Infinity is not a real number—it is a concept or an idea. It is something that cannot be measured.

A few examples of infinity are,

$$22/7 = 3.142857\ 142857\ 142857\ 142857\ 142857\ 1.....$$

This goes on and on infinitely and never ends. 22/7 is an approximation of π.

or $$1/3 = 0.3333333333.....$$

The number of points in a line or a circle is infinite.

Imaginary number

Rene Descartes used the term "imaginary number" originally and wrote about these numbers in La Géométrie to refer to the numbers that are today known as the complex numbers. The term

imaginary was meant to be derogatory. An imaginary number is a number that does not exist. i is called the imaginary unit.

$$i = \sqrt{-1} \quad \text{and}$$
$$i^2 = -1$$

i multiplied by a real number converts the number into an imaginary number. An imaginary number when squared is either 0 or negative. This is a nonexistent number because a number multiplied by itself cannot be negative. Initially, imaginary numbers were considered useless and derogatory. Later during the Enlightenment Era or the Age of Reason in the 17th and 18th century, mathematicians like Euler, Gauss and Wessel demonstrated their application and usefulness. Imaginary numbers are square root of real negative numbers. i is used in signal processing, electromagnetism, fluid dynamics, quantum mechanics and several other areas. Imaginary number is also used in several equations like the Euler Equation. An imaginary number when added to a real number gives a complex number. $a + ib$ is a complex number where a is the real part and b is the imaginary part of the complex number. Thus, an imaginary number is a complex number with real part equal to 0.

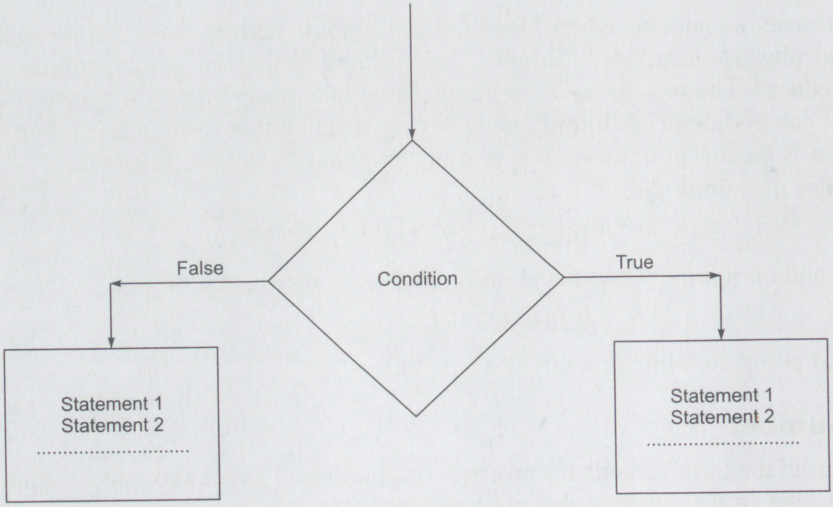
Identity matrix

An identity matrix of size n is a n by n square matrix whose diagonal elements are all 1 and other elements are 0.

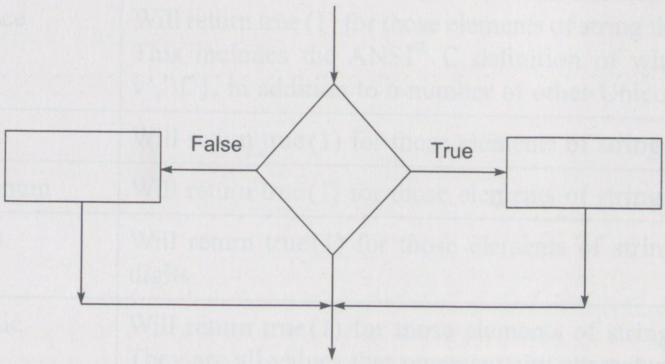
$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

if...else...statement

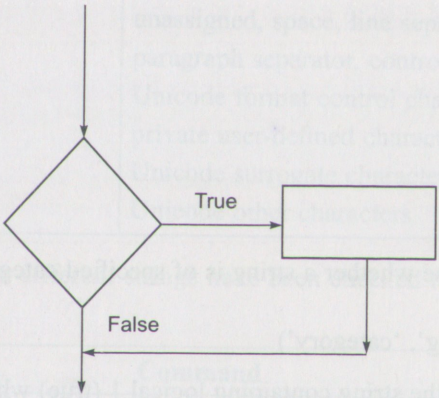
(a)



(b)



(c)



(a–b) the if/else structure and (c) the if structure

If the condition is true then the statements in the right hand box are executed. On the other hand it statement is false then the statements in the left hand box are executed.

input()

x=input('enter the unknown value');

This command displays the prompt on the screen and an input from the user is expected. The value entered is stored in the variable x. If the return key is entered without entering any value then the input function returns an empty matrix. If an erroneous data type is entered then an error message is shown and the prompt is redisplayed so that a correct value could be entered. A newline tabe ,'\', could also be used to enter value in a separate new line.

```
>> x=input('enter a')
enter a2
x =
     2
```

```
>> x=input('enter a')
enter a

x =

    []

enter a2

x =

     2

>> x=input('enter a\n')
enter a
2

x =

     2
```

isstrprop

isstrprop is a function that can be used to determine whether a string is of specified category or not. For example,

a = isstrprop ('string', 'category')

will return a logical array having the same size as the string containing logical 1 (true) when the elements of string belong to the specified category and logical 0 (false) when it does not belong to the specified category. The string can be a character array, cell array or any MATLAB numeric type. If the string is a cell array, then the return value is a cell array of the same shape as string. The choice of category input can be any of the type shown below.

Category	Condition for logical 1(True)
alpha	Will return true (1) for those elements of string that are alphabetic
digit	Will return true (1) for those elements of string that are numeric digits
lower	Will return true (1) for those elements of string that are lowercase letters
print	Will return true (1) for those elements of string that are graphic characters, plus char (32)
punct	Will return true (1) for those elements of string that are punctuation characters
cntrl	Will return true (1) for those elements of string that are control characters (for example, char (0 :20))

Contd.

wspace	Will return true (1) for those elements of string that are white-space characters. This includes the ANSI® C definition of white space, {' ','\t','\n','\r','\v','\f'}, in addition to a number of other Unicode characters.
upper	Will return true (1) for those elements of string that are uppercase letters
alphanum	Will return true (1) for those elements of string that are alphanumeric
xdigit	Will return true (1) for those elements of string that are valid hexadecimal digits
graphic	Will return true (1) for those elements of string that are graphic characters. They are all values that represent any characters except for the following: unassigned, space, line separator, paragraph separator, control characters, Unicode format control characters, private user-defined characters, Unicode surrogate characters, Unicode other characters

Here different strings have been checked to know if they belong to the a particular category or not.

Command	Output
>> punctornot = isstrprop('.', 'punct')	punctornot = 1
>> punctornot = isstrprop('"', 'punct')	punctornot = 1
>> punctornot = isstrprop('[', 'punct')	punctornot = 1
>> upperornot = isstrprop('A', 'upper')	upperornot = 1
>> upperornot = isstrprop('a', 'upper')	upperornot = 0
>> digitornot = isstrprop('1', 'digit')	digitornot = 1

Contd.

Contd.

>> digitornot = isstrprop('a', 'digit')	digitornot = 0
>> alphaornot = isstrprop('a', 'alpha')	alphaornot = 1
>> alphaornot = isstrprop('8', 'alpha')	alphaornot = 0

Linear Search

Linear search compares each element of an array with the search key. Since the elements of the array are not ordered so the probability of finding the search key in the first element or the last element of the array is the same. The linear search method is suited for small arrays and unsorted arrays. It is not efficient for large arrays. For unsorted arrays the binary search method is found to be more appropriate.

Loop Counter

Loop counter is the term used to refer to the variable that keeps a count on the number of iterations of a loop. The loop counter is a variable that is initialized and then incremented or decremented. The loop counter value changes with every iteration. The loop counter is also used to decide if the loop should be exited or not. First the name of the counter variable is assigned along with its initial value. The increment or decrement value by which the control variable is increased or decreased every time the loop is executed is set. A condition which tests for the final value of the counter variable is also set. This determines whether the loop should be continued or exited.

Memory

The command memory gives the information about how much memory is available and how much the MATLAB software is currently using. For example, in the system that is being used,

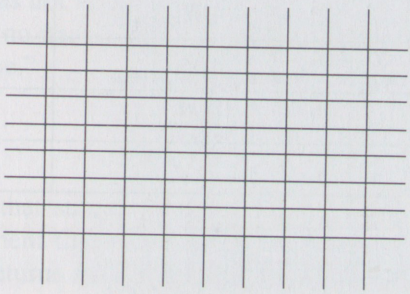
```
>> memory
Maximum possible array:              754 MB (7.902e+008 bytes) *
Memory available for all arrays:     1445 MB (1.515e+009 bytes) **
Memory used by MATLAB:               270 MB (2.828e+008 bytes)
Physical Memory (RAM):               3567 MB (3.741e+009 bytes)
```

* Limited by contiguous virtual address space available.

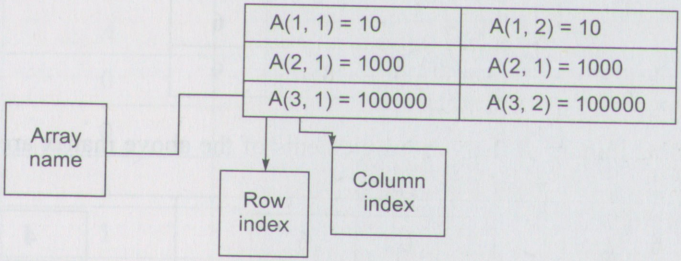
** Limited by virtual address space available.

Memory Map

A 2-dimensional array is shown below:



Each array cell is designated and address which contains the name of the array and the row and column number.



Arrangement of the array in the memory of the computer,

a[0][0]	a[0][1]	a[1][0]	a[1][1]	a[2][0]	a[2][1]
10	100	1000	10000	100000	1000000
1002	1004	1006	1008	1010	1012

The memory in our computer does not contain rows and columns. Regardless of the fact whether the array is 2-dimensional or 1-dimensional all elements of the array are arranged in a linear continuous manner.

The entire memory is divided into a number of cells depending on the size of the memory (for example 256 MB or 1 GB). Each memory cell has a unique address. The address is an integer. The address is always increasing when an array is stored in the memory.

Memory capacities are measured in k bytes (or kilobytes). Though k implies 1000 but the term kB means $2^{10} = 1024$ bytes. So a 64 kB machine has $64 \times 1024 = 65536$ bytes storage capacity.

Name(symbol)	Value	Binary usage
Kilobyte(kB)	10^3	2^{10}
Megabyte(MB)	10^6	2^{20}
Gigabyte(GB)	10^9	2^{30}
Terabyte(TB)	10^{12}	2^{40}
Petabyte(PB)	10^{15}	2^{50}

Minor

Let us consider matrix,

1	2	3
4	5	6
7	8	9

The corresponding minors of the various elements of the above matrix are:

5 6 8 9	4 6 7 9	4 5 7 8
2 3 8 9	1 3 7 9	1 2 7 8
2 3 5 6	1 3 4 6	1 2 4 5

The cofactor of $a(i, j)$ element of the matrix a is $(-1)^{(i+j)}M_{ij}$. Here M_{ij} is the minor of $a(i, j)$ element of matrix a. If a is a n by n matrix then the dimension of M_{ij} is $(n-1)$ by $(n-1)$.

Null

The dictionary meaning of the word null is amounting to nothing, absent or nonexistent. A variable having no value is called null. In fact null value is the only possible value of the variable type null. Having a value 0 is different from having a value null since 0 is the value

of the variable and null is absence of any value. In a Boolean test however both null and 0 are considered false. A null character on the other hand is a character which has all its bits set to zero. It has a numeric value 0 and is used to represent the end of a string character. The null character helps us determine the size of a sting. A variable is considered null when, it has been assigned the constant null, it has not been set any value, or it has been unset. It should be noted that the null type variable has no name and is impossible to declare a variable of the null type or to cast a variable to null type."

NAND

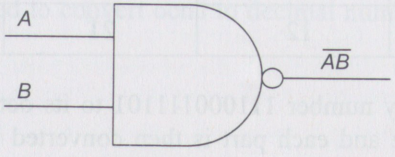
NAND is a Boolean operator that stands for NOT AND or Negated AND. A NAND between two words will return a document that fails to contain both the words. In digital electronics a NAND is a logic gate which returns false if all its input are true.

The truth table of NAND is given below.

Input		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

The output is 1 when either of A or B is 1 or if neither of A or B is 1. The NAN gate and NAND gates are said to be universal gates since a combination of them can accomplish any of the basic operations. The NAND gates functions like an AND gate followed by NOT.

The NAND gate symbol is shown below.



Octal system

Octal is a number system that uses only digits 0,1,2,3,4,5,6 and 7. There is no 8 in the octal system so after 7 comes 10 and similarly after 17 comes 20.

For example, if we want to convert an octal number say 1234 to decimal number then,

8^3	8^2	8^1	8^0
1	2	3	4
1×8^3	2×8^2	3×8^1	4×8^0
512	128	24	4
512 + 128 + 24 + 4 =			668

In octal number system the base or radix is 8. The octal numbers are 0 to 7. All numbers are expressed as powers of 8. An octal digit is equivalent to 3 bits. To convert a binary number to octal number we can split the binary number into groups of 3 binary bits and each group is then converted into its equivalent octal number. 1 octal digit is equivalent to 3 bits.

Decimal	Octal	Decimal	Octal
0	0	11	13
1	1	12	14
2	2	13	15
3	3	14	16
4	4	15	17
5	5	16	20
6	6	17	21
7	7	18	22
8	10	19	23
9	11	20	24
10	12	21	25

If we want to convert a binary number 111000111101 to its octal equivalent then the number is broken into groups of three and each part is then converted to its octal equivalent. This is shown below.

111	000	111	101
7	0	7	5

When 7075 is converted to equivalent binary number it gives 111000111101.

Three examples showing how a decimal number can be converted to an octal number are given below.

Divisor	Dividend	Quotient
8	30	3
	24	
Remainder	6	
Octal equivalent of 30 is 36	$3 \times 8^1 + 6 \times 8^0 = 30$	

Divisor	Dividend	Quotient
8	50	6
	48	
Remainder	2	
Octal equivalent of 50 is 62	$6 \times 8^1 + 2 \times 8^0 = 50$	

Another method by which this can be done is shown below. This method is appropriate for numbers having 3 or more digits.

Decimal Number, Dividend	Divisor or base	Quotient	Remainder
100	8	12	4
12	8	1	4
1	8	0	1
Octal equivalent of decimal number 100 is 144	$1 \times 8^2 + 4 \times 8^1 + 4 \times 8^0 =$		

oct2dec function can be used to convert octal to decimal numbers. For example,

```
>> oct2dec(1234)

ans =

    668

>> oct2dec(16)

ans =

    14
```

A MATLAB program to convert an decimal number to octal number is given below.

```
n = input('enter an octal number:');
b=num2str(n);
```

```
d = 0;
c = length(b);
for i = 1 : length(b)
d = d + str2num(b(i)) * 8^(c-1);
c = c-1;
end
d
```

enter an octal number:222

d =

146

$$2 \times 8^0 + 2 \times 8^1 + 2 \times 8^2 = 2 + 16 + 128 = 146$$

OOP

OOP stands for object oriented programming. OOP is a programming philosophy. OOP refers to a method of programming that is based on objects. OOP was not present in old languages like FORTRAN and Pascal. It is based on objects and not just functions and procedures. OOP is a type of programming which uses objects and these objects have data fields. C/C++, PHP and Java are examples of object oriented programming languages. An OOP can be visualized as a collection of interacting objects whereas in conventional programming a program performs a list of tasks called subroutines. The objects are organized into classes and each object has a structure similar to the other objects in the class. OOP helps programmers structure and organize programs as a particular object could be modified and changed without disturbing the other aspects of the program. Thus, it is much easier to make changes to a program in object oriented programming. Large programs became more manageable using OOP. Languages with OOP features are, C++, C#, COBOL, Cobra, Fortran 2003, FreeBASIC, Java, LISP, MATLAB, Object Pascal, Perl since v5, PHP5, JavaScript, JScript, Python, Simula, Smalltalk, Squirrel, Visual Basic, Visual FoxPro, Visual Prolog and several other languages."

Overwrite

In computer terminology overwriting refers to replacing the old data with a new data. Overwriting could refer to both text and files. Overwrite replaces existing characters as we type the new text. It is also possible to insert text between two words without overwriting. This can be done using the INS key or the Insert key. If we use the insert option then it is possible to insert text between two existing words without deleting the words on the right. This is also the default option in word processing programs. On the other hand if insert option is not used then the words on the right will be overwritten. The term overwrite can also be used for files. If a document with the same file name as an existing file is saved we would be asked if we would like to overwrite the existing file. If we choose yes then the old file will be replaced with the new file having the same name.

Punched card

A punched card, also known as IBM card or Hollerith card, is a card containing digital information. It is an early method by which data was stored. They are made of stiff paper having thickness

same as that of greeting cards. They have holes punched in them in definite positions depending on the data. These cards have been used since 1725 in textile handlooms. In 1832 the punch cards were later used to store information by Semen Korsakov. Herman Hollerith (29.2.1860–17.11.1929) in 1890 developed a method by which machines could record information in the punch cards. This method was later used for the census in USA. Hollerith was an American statistician and inventor who created the mechanical tabulator using the punch cards. This machine could tabulate statistics collected from millions of data like the census data. This reduced the time taken to complete the census. The data collected during the census was collected in the punched cards. In early 1900s punch cards have been used for storing and retrieving data. They could be seen even in the early 1980s but today they are obsolete and very hard to find. Since 1960s they began to be replaced by other storage devices. Hollerith founded the Tabulating Machine Company, which later merged to become IBM in 1924. He is known as the father of modern automatic computation.

Prime numbers

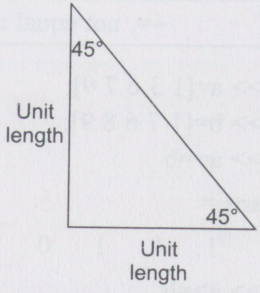
A prime number is an integer or natural number greater than 1 and is divisible only by 1 and itself. Its only divisors are 1 and itself. If a number is not a prime number it is called composite number. The property of number of being prime is called primality. Mainly to test if a number is prime or not it is often tested to see if the number is a multiple of an integer between 2 and $\sqrt{n}$. Here a program has been written to test if a number is prime or not.

Pythagoras's constant

$\sqrt{2}$ is known as the Pythagoras's constant. $\sqrt{2}$ is famous because it is probably one of the first irrational numbers that had been discovered.

$$\sqrt{2} = 1.41421356237309504880168872420969807856967187537694 80.....$$

According to Greek philosopher Aristotle (384–322 BC) it was the Pythagoreans around 430 BC who showed that $\sqrt{2}$ is irrational, which is also the length of the diagonal of a unit square. $\sqrt{2}$ is the length of the hypotenuse of triangle with sides having length of 1. This is an isosceles right triangle. Since $\sqrt{2}$ is irrational it cannot be exactly expressed by a ratio of two integers, a/b.



According to the legends when Pythagorean philosopher Hippasus found out the irrationality of $\sqrt{2}$ is he was travelling on sea and when he told others about the discovery, he was thrown overboard by the angry Pythagoreans because at that time all their system was based on integers and fractions of integers.

The Babylonians found that $\sqrt{2}$ is approximately,

$$\sqrt{2} \approx 1 + (24/60) + (51/60^2) + (10/60^3) = 1.41421296296296.....$$

$$1+1/(1 + \sqrt{2}) \approx 1.41421356237309504880168872420 97.....$$

RAM, random access memory

RAM or random access memory is a computer memory that can be accessed randomly. This implies that any byte of the RAM can be accessed without touching the preceding bytes whereas in a hard disk, CD, DVD or the primitive drum disks data could be read and written in a predetermined order. This is due to the difference in design of the two memories. The first RAM was developed in 1951. A RAM allows us to access data directly in any random order. RAM is a common type of memory found in computers and devices like the printer. There are two types of RAM, namely SRAM and DARAM. SRAM stands for static random access memory and DRAM stands for dynamic random access memory. SRAM is faster compared to DRAM because DRAM needs to be refreshed several times per second. SRAM does not need to be refreshed. Both the RAMs are volatile and they lose their data when the power is turned off. The technology used in the two RAMs is different. DRAM is more commonly used although it is slower compared to SRAM because it is cheaper.

Relational Operators

Relational operators can perform element by element comparison between arrays. They return a logical array whose dimension is the same as that of the arrays being compared. The elements of the array are 1 when the relation is true and the elements are set to 0 when it is not true. The operators ><<= >= compare only the real parts of the operands whereas the operators == and ~= compare both the real and imaginary parts.

>, greater than	a>b
< , less than	a<b
<=, less than equal to	a<=b
>=, greater than equal to	a>=b
==, equal to	a==b
~=, not equal to	a~=b

```
>> a=[1 3 6 7 9];
>> b=[1 7 6 8 9];
>> a==b
ans =
    1    0    1    0    1
>> a>=b
ans =
    1    0    1    0    1
>> a~=b
ans =
    0    1    0    1    0
```

Scope of a variable

The scope of variable is where a name can be accessed. For example, an auto variable has a local variable scope. This implies that it can only be accessed and changed in the block or function in which it is declared.

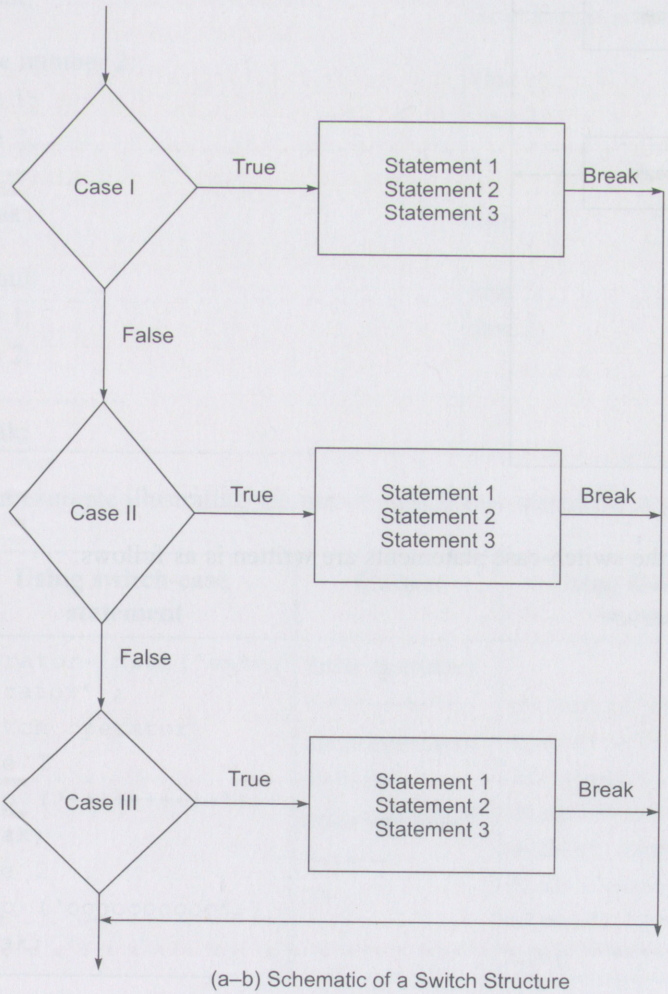
Storage duration of a variable

Storage duration is the time between the creation of a variable and its destruction. For example, an auto variable disappears when it goes out of scope. The memory space in which it was stored is free once an auto variable is out of scope.

switch

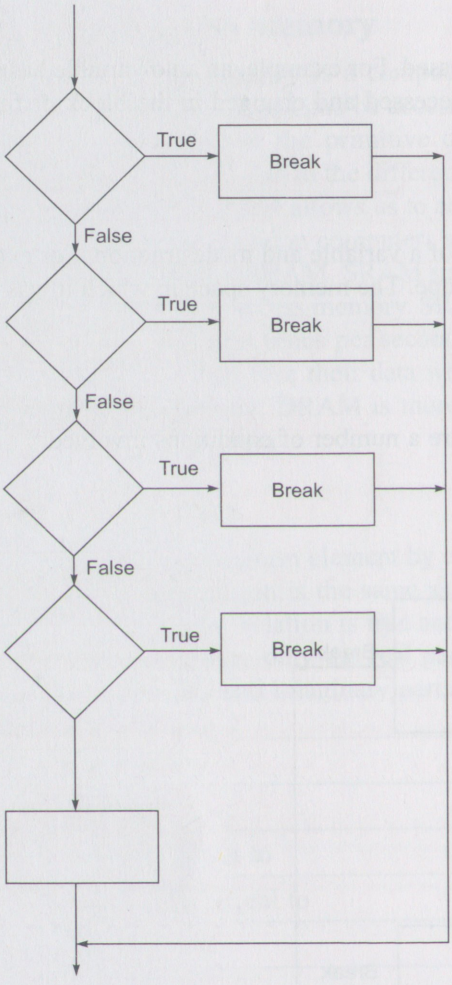
The switch-case statements are used when there are a number of conditions involved.

(a)



(a–b) Schematic of a Switch Structure

(b)



The standard form in which the switch-case statements are written is as follows:

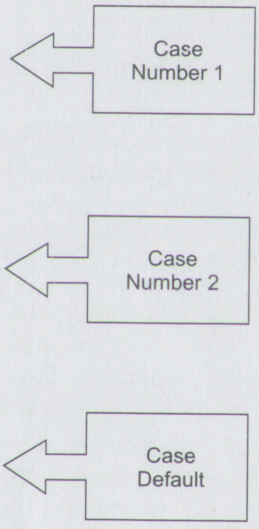
```
switch (expression)
case number1:
line 1;
line 2;
……..
break;
case number 2:
line 1;
line 2;
……
break;
```

```
default:
line 1;
line 2;
.....
break;
```

The switch-case statement is very similar to if-else if-else statement.

```
switch (expression)
case number1:
line 1;
line 2;
.....
break;

case number 2:
line 1;
line 2;
.....
break;

default:
line 1;
line 2;
.....
break;
```

```
if(expression)

line 1;
line 2;
.....

else if(expression)

line 1;
line 2;
.....

else

line 1;
line 2;
.....
```



An example illustrating the use of switch-case statement is given below.

Using switch-case statement	Output	Using if-elseif-else statement	Output

`operator=input('enter operator');`
`switch operator`
`case 1`
`disp ('++++++++++');`
`break;`
`case 2`
`disp ('oooooooooo');`
`break;`
	enter operator1
	++++++++++
	enter operator4
	!!!!!!!!!!
	enter operator5
	--------- 	
	>>
		`c=input('enter operator');`
		`if (c==1)`
		`disp ('++++++++++');`
		`elseif (c==2)`
		`disp ('oooooooooo');`
		`elseif (c==3)`
		`disp ('**********');`
			enter operator1
			++++++++++
			enter operator4
			!!!!!!!!!!
			enter operator5
			--------- 	
			>>

Contd.

Contd.

```
case 3
disp ('**********');
break;
case 4
disp('!!!!!!!!!!');
break;
otherwise
disp('----------');
break;
end
```

```
elseif (c==4)
disp('!!!!!!!!!!');
else
disp('----------');
break;
end
```

Both the methods give the same result.

Syntax

Syntax is a Greek word meaning *arrangement*, *together* or an *ordering*. In English language syntax is the way in which words are put together to form phrases, clauses or sentences. From linguistics point of view, it is the study of principles, processes and rules that govern the formation of grammatical sentences in a particular language using words.

From computers point of view, in programming language the term syntax refers to the rules and structural patterns that govern the use of appropriate words and symbols, in the particular computer language, for issuing commands or writing codes. Syntax is also used to refer to the rules that govern the behavior of mathematical systems.

Trojan

Trojan is a dangerous virus that is loaded in a computer through a host program. It is named after the wooden horse with which the Greeks infiltrated Troy. A Trojan or sometimes known as Trojan horse unlike the worm is a non-self-replicating type of virus or malware. Trojans spread when the user interacts with it. Opening and downloading a file from the Internet or opening an e-mail attachment from the Internet can cause Trojans to spread in a computer system. It appears to perform a desirable function and seems to do no harm but instead it drops a payload which is malicious to the computer. Computer users usually get tricked by the Trojans. It may even allow an invisible backdoor into the target computer and thus allow illegal access to the system. Trojans may steal information from the target computer and may cause damage to the computer system. Actually the Trojan Horse is a tale from the Trojan War. The Greeks built a huge wooden horse and hid warriors inside it. When the Greeks pretended to have lost the war the Trojans took with them the large wooden horse as a trophy. At night the Greek warriors hiding in the horse came out and opened the gates for the rest of the Greek army to enter the city of Troy. They destroyed the city of Troy and ended the war.

Unix,UNIX

"UNIX is basically a simple operating system, but you have to be a genius to understand the simplicity." ...Dennis Ritchie

Unix is a operating system that was originally developed by Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, Michael Lesk, Joe Ossanna in the year 1969. Unix has been under development ever since its inception. Operating system or OS for short is a suite of programs that makes a computer work. It was originally written in Assembly language and was later written in C in 1973. This led to the further development of the language and it could also be used in other operating systems. In the following year in 1974 it was licensed for use by univerisites like the University of Illinois at Urbana Champaign. The UNIX trademark is owned by the Open Group. There are many different versions of Unix. Some of them are Sun Solaris, GNU/Linux and Mac OS X. A Unix terminal window starts with a % prompt. It indicates that it is now ready for input of commands. To put it in brief the Unix operating system has three parts, the kernel, the shell and the programs.

"I think the major good idea in Unix was its clean and simple interface: open, close, read, and write."...Ken Thompson.

Virtual Reality (VR)

Virtual Reality (VR) has been widely used for entertainment or gaming purpose but it is also used in military and medical fields. It is widely used in training programs. VR is a term used for computer simulated environments. It simulates the physical presence in real and imaginary world. The simulated environment is so similar to the real world that it gives us a lifelike experience. Virtual reality creates a computer generated world or environment which people can explore or interact with. The dictionary meaning of the term virtual means something that exists or results in essence or effect but not in actual fact, form or name. It is something that is temporarily simulated or extended by computer software. It can also mean something that is created or simulated and therefore virtual reality is best described as an illusion of something real and mostly created by computer system. A virtual reality world can be entered by wearing special glasses, helmets and sensors to touch and move virtual objects attached to the computer that is running the virtual reality program. Any movement of the user also moves the virtual world accordingly in real-time. VR mostly includes 3-dimensional life-sized images. It is capable of tracking the movement of the user's head and eye and it accordingly adjusts the images on the monitor to reflect the changes in perspective.

Workstation

Workstation is like a stop or station where work comes and stops to get on the computer and reach the destination or get completed just as at a railway station passengers come and stop to board a train and reach their destination. This sounds like a good analogy! The term workstation refers to a high-end computer that does a specific task for the user. It is computer that has been designed to do a specific job. Workstations generally have a higher performance than desktop computers with respect to processor speed, memory size, graphics, etc. They have multitasking capability. They are generally used for visualization of data in 3D, designing of machines, simulation of problems,

animation, solution of complex mathematical problems and their plots, etc. Workstations are generally connected to each other or networked although they may be standalone as in the case of the computer I am using to write my book. Thus in a network the workstations can communicate with each other making it possible to do a large integrated job. The entire work can be split into pieces and solved by different users on different workstations and finally can be joined to present a complete solution. Workstations built by different companies are available in the market for example, DELL, HP, Apple, etc. The term workstation has also been used to refer to mainframe computers. In general usage the word mainframe means "big iron". It is used to distinguish high-end machines from the ones that are less powerful computers. The most common operating system for workstations is UNIX and Windows NT.

Worm

A worm is a malicious computer software that replicates itself. It does not alter any files. Worms can cause problems by replicating itself and consuming memory space of the hard disk. As a result the computer might become very slow and ultimately crash. It affects the accessing of files in the hard disk and new file cannot be created or saved. A worm is a computer program which uses the computer network to replicate and spread itself. It relies on the security failures or the failure the antivirus to enter the computer. Worms often cause some harm to the computer network. Unlike a computer virus it does not attach to an existing program. Some worms may also be harmless or even useful. The XSS worms have been written to research how worms spread and how they affect the behavior of the user.

Zebibyte

A zebibyte is a data storage unit which equals to 270 bytes.

$$270 = 1{,}180{,}591{,}620{,}717{,}411{,}303{,}424$$

A zebibyte is closely related to a zettabyte on the other hand is 1021 bytes and is approximately, 1,000,000,000,000,000,000,000 bytes in size.

The symbol for zebibytes is ZiB. Zebibyte is a binary multiple of byte. It is also a unit of storage of data.

Name of unit and symbol	Value in decimal	Binary value
Kilobyte (kB)	10^3	2^{10}
Megabyte (MB)	10^6	2^{20}
Gigabyte (GB)	10^9	2^{30}
Terabyte (TB)	10^{12}	2^{40}
Petabyte (PB)	10^{15}	2^{50}
Exabyte (EB)	10^{18}	2^{60}
Zettabyte (ZB)	10^{21}	2^{70}
Yottabyte (YB)	10^{24}	2^{80}

Index

A

Absolute value 50
Algorithm 10
Animation 197
Arithmetic Logic Unit (ALU) 358
Arrays 40
ASCII 51
Assembly language 12
assignment operator (=) 378
Audio data 207
Axis, types of 137

B

Backslash character constants 357
Bar chart 307
Barnsley's fern problem 106
Bisection method 276
Bits and bytes 358
Bitwise operator 361
Boolean algebra 359
Bounding phase method 253
boxplot 312
Bracketing methods 249
break 362
Brightness 121
Buckyball 192
Built-in MATLAB variables 363

C

Call-by-reference 363
Call-by-value 363
camlight 174
camup 164
cast 364
Charles Babbage 365
Children 130
close 365
colordef 101
Colormaps 169, 366
Colour specification 125
Comet plot 369
Command 29
Compiler 35, 37, 38, 371
Computing 1
Cone 109
continue 371
Contour plots 111
Contrast 120
Control-Alt-Delete or Ctrl-Alt-Del 14
Conversion character 373, 374
Conway's game of life 25
Cross product 116
Cruller 26
Cube 117
Cubic equations 375
Cubic fit 237
Cursor 14

D

Depth of field 124
Determinat, det() 377
Digital image 119
disp () 377
Dot product 116
drawnow 201
.m, .fig, .mat and .mex 358

E

earthmap 26
EBCDIC 51
Eigenvalue 79
Eigenvector 79
eps 55
Equality operator (==) 378

erf (x), error function 293
erfc(x), complimentary error function 293
Exclusive or 301
Exhaustive search method 249

F
FaceAlpha 132
Fibonacci search method 265
fifteen 26
figure 99
figure('color',c) 161
First-order equations 245
Fliplr 88
Floating point number 378
Floating point number 47
Flowchart 379
Fourth Order Runge-Kutta method 242

G
getframe 205
ginput 96
Golden Master 13
Golden ratio 381
Golden section method 269
Gradient-based methods 272
gtext 189

H
h=gtext ('string') 189
Hadamard matrix 55
handle 203
Help 22
Hibernate 14
High level language 12
Histogram 311
hold, hold on, hold off 383

I
Identity matrix 82, 384
if...else...statement 384
Image reading 160
Image rotation 156
image (c) 129
imageext 27
Imaginary number 53, 383
Infinity (∞) 53, 383
Inline M-files 246
input() 385
Integer representation 44
Interpolation 227
Interpreter 38
Intersection 301
Interval halving method 261
isstrprop 386

K
Kernel 10

L
Lagrange interpolation 228
Least square method 230
Lighting 139
Linear equations 216
Linear Search 388
Logical operators &, |, ~and xor 67
Loop counter 388
Lower triangle 58
Lower triangular matrix 57

M
makevase 27
Mantissa 54
Material 143
MATLAB 17, 90
  array operations 41
  built-in demos 23
  colours in 85, 95
  making movies with 204
  variable in 49
MATLAB compiler 38
MATLAB® Mobile™ 29
  date, calendar 33
  default layout 31
  pause 31
  web browser 30
Matrices 40
Matrix
  transpose of 58, 77
Mean 285
Median 286

Megapixel 129
Memory 388
Memory map 389
Mesh plot 102
meshgrid 61
Microprocessor 7
Minor 390
Mode 289
Motherboard 9

N
NAND 391
Newton-Raphson method 239, 273
Non-linear equations 223
Normal distribution 293
Normalization 298
Null 390
Numerical integration 210

O
Octal system 391
Ones' compliment 66
OOP 394
Ordinary differential equations 242
Output 52, 225
Overwrite 394

P
Palindrome 81
Palindromic number 81
Paraboloid 110
pcolor 158
pi 60
Pie chart 305
Pixel 127
Plane 113, 115
Plane plotting 116
plot and plot3 98
Plotting a line 90, 93
Polyfit 235
Polynomial roots 239
Polyval 235
Prime numbers 395
Product of two matrices 71
Programming languages 11
Punched card 394
Pythagoras's constant 395

R
Rack unit 15
RAM 396
Random access memory (RAM) 7
Random numbers 313
Range 285
Real number 50
Region elimination method 259
Relational operators 68, 396
Reversing a number 81
RGB image 172
rotate 167
Rotation matrix 85

S
Scatter plot 100
Script files 21
Secant method 280
Server 9
Shading 132
Simpson's 1/3 rule 213
Single and double 46
Sound 206
Sparse matrix 83
Square matrix 74
Standard deviation 291
Statistical functions 285
Statistics 285
Steps to Execute a Program 16
Subtraction 301
surf plot 104
Surface normal (surfnorm) 115
Surfc plot 104
switch 397
Syntax 15, 400

T
Temporary variable 70
text 188
The ':' operator 69
Trace 87
Translation matrix 87

Transparency 124
Trojan 15, 400
truss 28
Two-dimensional interpolation 220

U
Union 301
UNIX 401
Upper triangle 58
Upper triangular matrix 57

V
Variable storage 23
Variance 291
view 150
Virtual reality 401

W
Workstation 401
Worm 402

X
x86 8

Z
Zebibyte 402

Advanced Guide to MATLAB

A sequel to *Understanding MATLAB: A Textbook for Beginners,* this book is an advanced-level guide to MATLAB with a number of applications in science and engineering. There are a number of solved problems from all important areas of science and engineering to give enough practice to the reader. MATLAB integrates computation, visualization and programming in a very user-friendly and easy-to-use environment.

This book is mainly aimed at those who already known the basics of the language and would like to use MATLAB to solve advanced engineering problems. Here, in this book, the authors have tried to illustrate the advanced features of MATLAB.

Salient Features:

- Provides a brief discussion on the history and evolution of MATLAB
- Discusses the MATLAB compiler and its working
- Detailed treatment of numbers, matrices and matrix operations
- Dedicated chapters on plotting, image processing, and animations, sounds and movies
- Comprehensive coverage of numerical methods
- A separate dictionary of additional terms at the end of the book
- A separate chapter on solved problems
- Step-by-step discussion of solved problems
- Abundant sample codes, algorithms, and flow charts

S. N. Alam is an Assistant Professor, Metallurgical and Materials Engineering, National Institute of Technology, Rourkela, Odisha. He received his B.Tech in 1997 from the Indian Institute of Technology, Kharagpur in Metallurgical and Materials Engineering and his M.S. in Photonics and Electronic Materials in the year 2001 from the Chemical and Nuclear Engineering Department of the University of Massachusetts, Lowell, USA. He obtained his Ph.D. in the year 2010 in Metallurgical and Materials Engineering from the Indian Institute of Technology, Kharagpur. He has also taught at BITS-Pilani.

Sanjib Islam is a senior technology leader. He has been providing IT, project management, business process, and infrastructure solutions to companies, (Upland Software, Accenture, Fidelity Investments, Lucent Technologies) for over the past 20 years. He earned his M.S. in Engineering from Case Western Reserve University, USA and B. Tech (Hons) in Engineering from IIT, Kharagpur. He is a SUN Certified Enterprise Architect. He also got his MBA from the University of Massachusetts, Lowell, USA.

S. K. Patel is an Associate Professor, Mechanical Engineering Department, National Institute of Technology, Rourkela, Odisha. He graduated in Mechanical Engineering from the National Institute of Technology, Rourkela in the year 1985. Initially, he served with Hindustan Aeronautics Limited, Koraput Division, Sunabeda for 5 years. He obtained his M. Tech degree from the Indian Institute of Technology Madras, Chennai and Ph.D. degree from the Indian Institute of Technology, Kharagpur. His broad areas of interest are statistical quality control, optimization and industrial engineering and management.

dreamtech PRESS

₹ 665/-

978-93-89520-29-3

9 789389 520293