Computer Applications Workbook

Department of Physics, Visva-Bharati, Santiniketan 731235

About This Workbook

Undergraduate and postgraduate courses in science typically have some course on computer applications which typically include elements of programming languages (FORTRAN, C, C++, Java, Python, HTML, ...), graphical presentation and statistical treatment of experimental data, numerical methods and preparation of scientific documents.

There are many excellent books available for teaching programming languages. Similarly, there are books for teaching numerical methods. But while teaching the postgraduate students of Physics the course on Computer Applications it was felt that none of these books serve the students of science well. Primarily because the excellent books on languages are structured for more serious practitioners of the art and the science of programming. While for the students of science it is *one of the subjects* and more importantly their main objective is to *learn* the basics and immediately use them for getting their *calculations* done. They can not afford to devote a lot of attention and effort to learn the language in a more exhaustive way and *then* apply it to develop codes - at least during their course work.

A *workbook* approach has been adopted in this text so that the students learn while they take one *small step* at a time and take *giant leaps* when they feel more confident. Starting with simple programs the basics of the language and the art of writing a good code have been presented. A very brief introduction to managing file systems and the basics of Linux is included with the motto *know only what you need to know*. There are many free online resources and forums to offer help to the interested.

It is assumed that the basic *Linux* platform (Red Hat, Debian, Fedora, Ubuntu, ... of different vintages) is available with the language compilers. No special package, library or software is needed for most of this workbook. In case of any problem the local Linux *guru/druid/wizard* should be able to help you out. They are usually recognized by at least one of the following features - unkempt hair, ponytail, crumpled dress and pithy sense of humuor.

In this workbook nothing has been included on *text editors*. Depending on the version you have you are spoilt for choice of text editors in Linux - *vi, vim, gvim, emacs, xemacs, pico, gedit, ...* What text editor you will use depends on your personal taste and who is around to initiate or help you. It is assumed that *you shall use and learn* and most of these editors have built-in help.

Any suggestion regarding the content or organization in the text within the aim and scope of this workbook is welcome and is earnestly sought for both from the students and the Linux *guru/druid/wizard*s who have devoted their lives to spread the Linux virus all over the world.

Contents

Chapter 1

Introduction to Computers

1.1 Basics of Computers

A computer does only what you tell it to do, not what you want it to do.

You will realize the meaning of this simple statement slowly - after making many mistakes and fixing numerous *bugs*. As this course progresses you will make all sorts of mistakes - simple, obvious, subtle, gross. You will get results which are wrong and you did not want or expect. Slowly and painfully you will realize the profoundness of the statement above. But don't despair, at the end you will hopefully learn a few things about computers and computing.

1.1.1 What is a computer?

A computer is a machine that executes a set of instructions on a set of data.

The ability to store and execute lists of instructions called *programs* makes computers extremely versatile and distinguishes them from calculators. Computers with capability and complexity ranging from that of a personal digital assistant to a supercomputer are all able to perform the same computational tasks given enough time and storage capacity.

1.1.2 A very brief history of computers

One can use abacus or slide rules for addition, multiplications, etc. But they can not store a set of instructions or data to be excuted at a suitable time or sequence - they are not programmable devices.

Development of programmable devices has a long history. In 1801, *Joseph Marie Jacquard* made an improvement to the textile loom that used a series of punched paper cards as a template to allow his loom to weave intricate patterns automatically. The resulting *Jacquard Loom* was an important step in the development of computers because the use of punched cards to define woven patterns can be viewed as an early, albeit limited, form of programmability.

Large-scale automated data processing of punched cards was performed for the U.S. Census in 1890 by tabulating machines designed by *Herman Hollerith* and manufactured by the *Computing*

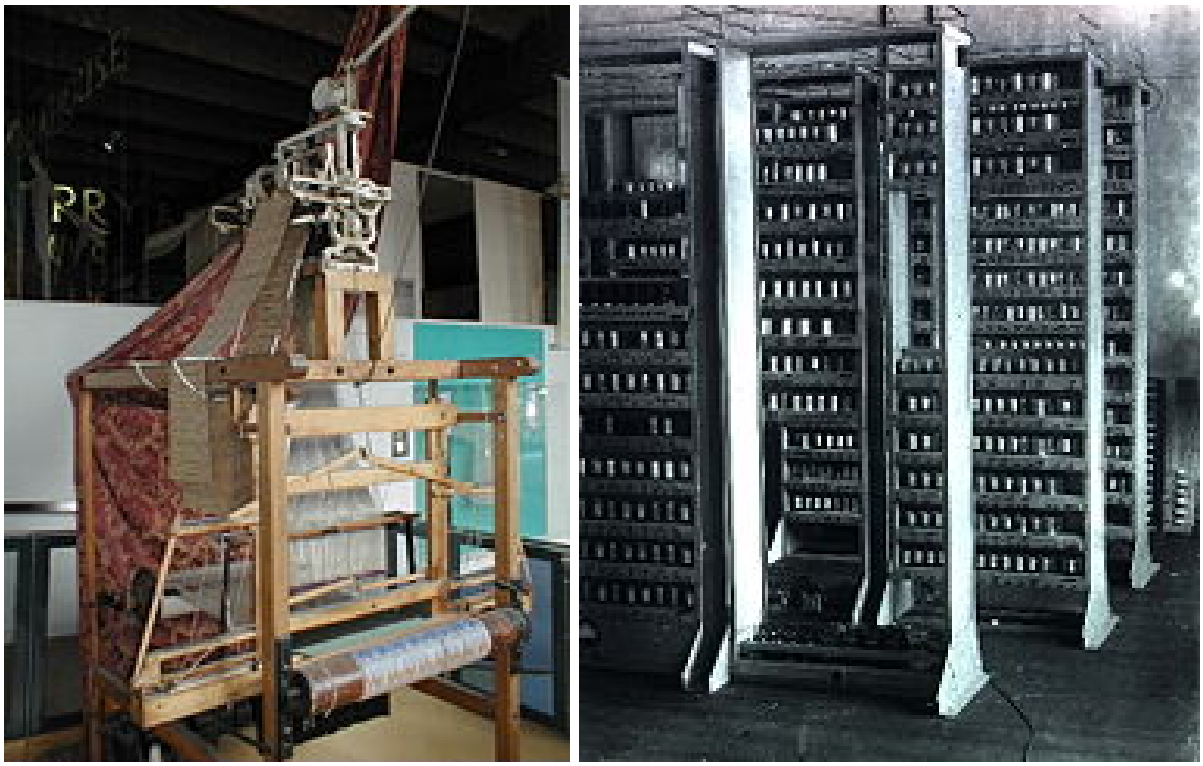


Figure 1.1: The Jacquard loom (left) was one of the first programmable devices. EDSAC was one of the first computers to implement the stored program (Von Neumann) architecture.

Tabulating Recording Corporation, which later became *IBM*. By the end of the 19th century a number of technologies that would later prove useful in the realization of practical computers had begun to appear: *the punched card, Boolean algebra, the vacuum tube (thermionic valve) and the teleprinter.*

The first devices that resembled modern computers date to the mid-20th century (around 1940 - 1945), although the computer concept and various machines similar to computers existed earlier.

1. The U.S. Army's Ballistics Research Laboratory's *ENIAC* (1946), which used decimal arithmetic and is sometimes called the first general purpose electronic computer. Initially, however, *ENIAC* had an inflexible architecture which essentially required rewiring to change its programming.

2. Several developers of ENIAC, recognizing its flaws, came up with a far more flexible and elegant design, which came to be known as the *stored program architecture* or *Von Neumann architecture*. This design was first formally described by John Von Neumann in the paper *First Draft of a Report on the EDVAC*, distributed in 1945.

3. The first such machine to be demonstrated working was the Manchester Small-Scale Experimental Machine (SSEM or "Baby").

4. EDSAC, completed a year after SSEM, was the first practical implementation of the stored program design.

5. EDVAC, the machine originally described by Von Neumann's paper completed but did not see full-time use for an additional two years.

Early electronic computers were the size of a large room, consuming as much power as several hundred modern personal computers. Modern computers are based on tiny integrated circuits and are millions to billions of times more capable while occupying a fraction of the space. Personal computers, in various forms, are icons of the Information Age and are what most people think of as *a computer*. However, the most common form of computer in use today is the embedded computer. Embedded computers are small, simple devices that are used to control other devices - for example, they may be found in machines ranging from fighter aircraft to industrial robots, digital cameras, and children's toys.

1.2 The Anatomy and Physiology of a Personal Computer

A general purpose personal computer, aka PC, has four main sections, which are interconnected by *busses* (made of groups of wires): the arithmetic and logic unit (ALU), the control unit, the memory, the input and output devices - collectively termed I/O.

1.2.1 The Central Processing Unit (CPU)

The control unit, ALU, registers, and basic I/O (and often other hardware closely linked with these) are collectively known as a *central processing unit (CPU)*. Early CPU's were composed of many separate components but since the mid-1970s CPU's have typically been constructed on a single integrated circuit called a *microprocessor*.

1.2.2 The Memory

In almost all modern computers, each memory cell is set up to store binary numbers in groups of eight bits (called a *byte*). Each *byte* is able to represent 256 different numbers; either from 0 to 255 or -128 to +127. To store larger numbers, several consecutive bytes may be used (typically, two, four or eight). When negative numbers are required, they are usually stored in *two's complement* notation. A computer can store any kind of information in memory as long as it can be somehow represented in numerical form. Modern computers have billions or even trillions of bytes of memory.

Logically a computer's memory is a list of cells into which numbers can be written or read from. Each cell has a numbered *address* and can store a single number. The computer can be instructed to [1]

```
put the number 123 into the cell numbered 1357
put the number 548 into the cell numbered 2468
add the number that is in cell 1357 to the number that is in cell 2468
put the answer into cell 1595
```

The information stored in memory may represent practically anything. Letters, numbers, even computer instructions can be placed into memory with equal ease. Since the CPU does not

[1] Could you guess the corresponding piece of computer code?

differentiate between different types of information, it is up to the software to give significance to what the memory sees as nothing but a series of numbers.

1.2.3 The Input and Output (IO) Devices

Input and output (I/O) devices are the means by which a computer receives information and instructions from the outside world and sends results back. Some I/O devices are part of the computer - keyboard, touch screen, mouse, etc. External devices that provide input or output to the computer are called peripherals. On a typical personal computer, peripherals include input devices like the keyboard and mouse, and output devices such as the display and printer. Hard disk drives, floppy disk drives and optical disc drives serve as both input and output devices. Computer networking is another form of I/O. Often, I/O devices are complex computers in their own right with their own CPU and memory. A graphics processing unit might contain fifty or more tiny computers that perform the calculations necessary to display 3D graphics. Modern desktop computers contain many smaller computers that assist the main CPU in performing I/O.

Consider a simple digital camera. Is it a computer? If so, what are the I/O devices for this computer?

1.3 Starting and Shutting Down a Computer

Most personal computers (which we will be using most of the time) need hardware intervention for start up. In plain speak you press the *start* button, usually located on the front face of the CPU. If your computer has more than one OS, you are asked to choose the OS you want to use for the session. Usually there is one default OS which is activated if you don't give your choice. Typically the following actions take place:

- the basic part of the *Operating System - OS* is activated
- hardware checks are done : state of the hard drive, keyboard, mouse, printer, network(if any) and suitable messages are flashed on the display
- the core services associated with the OS are activated
- depending on the OS, a graphic interface including pull-down/pop-up menus is activated; you may use the menu or the *shell* to start other services and utilities.

In normal situation shutting down is done using the menu provided in the graphic interface or using commands issued at the shell. In case the OS is not responding there are several options - you may have to press *Ctrl-Alt-Del* keys together. In extreme cases you need to press the *start* button for a few seconds to shut down. Please talk to the local system administrator to find out the details and the exact procedure he/she wants you to follow.

1.4 What is an Operating System?

An operating system is the software component of a computer system that is responsible for the

management and coordination of activities and the sharing of the resources of the computer.

You have heard of Windows, Linux, Mac, VMS... *Operating system* (OS) defines how the computer is organized and operates. A computer can have more than one OS. You may have to choose at the time of starting the computer what OS you may want to use.

The operating system acts as a host for *application programs* - internet browser (Netscape, Mozilla, Firefox), text editors - (vi, gvim, gedit, emacs, ...), pdf file viewer, mail program, etc that are run on the machine. As a host, one of the purposes of an operating system is to handle the details of the operations of the hardware. This relieves application programs from having to manage these details and makes it easier to develop applications. Almost all computers, including hand-held computers, desktop computers, supercomputers, and even modern video game consoles and 'not so smart' mobile phones, use an operating system of some type.

Operating systems offer a number of services to application programs and users. Applications access these services through application programming interfaces (*API*) or system calls. By invoking these interfaces, the application can request a service from the operating system, pass parameters, and receive the results of the operation. Users may also interact with the operating system by typing commands or using a graphical user interface (*GUI*, commonly pronounced *gooey*). For hand-held and desktop computers, the GUI is generally considered part of the operating system. For large multi-user systems, the GUI is generally implemented as an application program that runs outside the operating system.

Common contemporary operating systems include *Microsoft Windows*, *Mac OS X*, *Linux* and *Solaris*. Microsoft Windows has a significant majority of market share in the personal computers - desktop and notebook computer markets, while servers generally run on Linux or other Unix-like systems. Embedded device markets are split amongst several operating systems.

1.5 Computer Security

Computers store wide variety of sensitive data - your bank accounts, your research findings, defence data, financial data of companies, etc. It is important that such data are not accessed by unauthorized persons or computers. Hence is the importance of computer security. These days comuters storing such data are connected via networks (internet or intranet) and denial of physical access to the computer does not guarranty security of data.

Security of a computer and more importantly the data stored in it depends on a number of technologies working properly. A modern operating system provides access to a number of resources, which are available to software running on the system, and to external devices like networks via the kernel. The operating system must be capable of distinguishing between requests which should be allowed to be processed, and others which should not be. While some systems may simply distinguish between *privileged* and *non-privileged*, systems commonly have a form of requester identity, such as a *username*. To establish identity there may be a process of authentication. Often a username must be quoted, and each username may have a corresponding *password*. Other methods of authentication, such as *magnetic cards* or *biometric data* (fingerprint, iris scan, voice, etc) may be used instead.

1.5.1 Logging In

Most modern OS provide a graphical interface to give your username and then asks for your password. The passwords are kept in a file which is encrypted. Usually even the system administrator (*root*) will not know your password, although he/she may reset a user's password if necessary, particularly if the user forgets his/her password. For remote login typically a shell prompt is provided for giving username and password.

1.5.2 Changing Password

The security of a computer system depends crucially on the password for ordinary users and the *system administrator*. Usually at the time of creating an account, the system administrator (*root* for the Linux/Unix systems) gives the user a password. The user is supposed to change it immediately using the *passwd* command which asks you to provide the old password and then type the new password twice. If you think your password has been guessed/cracked by somebody else, then also you are supposed to change the password. All large systems require the users to change passwords regularly. If you are the *system administrator*, or, *root* for your system, it is good practice not to use the system regularly as *root* - this way it is more likely that you will compromise the security of the system. It is advisable that you create an ordinary user account with a password different from the *root* password and use the computer as an ordinary user. The *root* has certain privileges which should be used only when system administration related tasks have to be performed. *You are not expected to let others know your password, even less the root password for your system.*

Chapter 2

Introduction to Linux

2.1 Operating System: Linux

We will learn `C++` using the `Linux` operating system - open source, quasi-free version of Unix. Unix and C were developed in the 1970's at Bell Labs, USA. Unix was more suitable for the fast 'RISC processors' in early 1990's. In 1991, Linus Torvalds developed a free, open source version of UNIX called Linux. Many *flavours* and versions of Linux are available - *Red Hat, Fedora, Ubuntu, Scientific Linux* - are some of them. With more people using and contributing to the development of and applications based on Linux, its appearence and user interface are becoming more *friendly* for *ordinary* users.

Unix operates at two levels:

- interaction between operating system and computer (the `kernel`),
- interaction between operating system and user (the `shell`).

Several shells (i.e. command sets) are available: `sh, csh, tcsh, bash, ...` - we shall be using `bash`.

2.2 File Structure: Files and Directories

A *file* is an individual item without any substructure (from the OS point of view). A file does not contain another file within it.
A *directory* is an item which may contain files and other directories.

File/directory names are case sensitive: `thesis` $\neq$ `Thesis`. When you start a *shell*, *terminal*, etc. usually you are in your *home directory*. Typically an user with the username [1] *buddhu* has a home directory `/home/buddhu`. Note the following command and its output (`$` means a shell prompt):

```
$ pwd
/home/buddhu
$
```

[1] From now on the *username* will be used to denote a user.

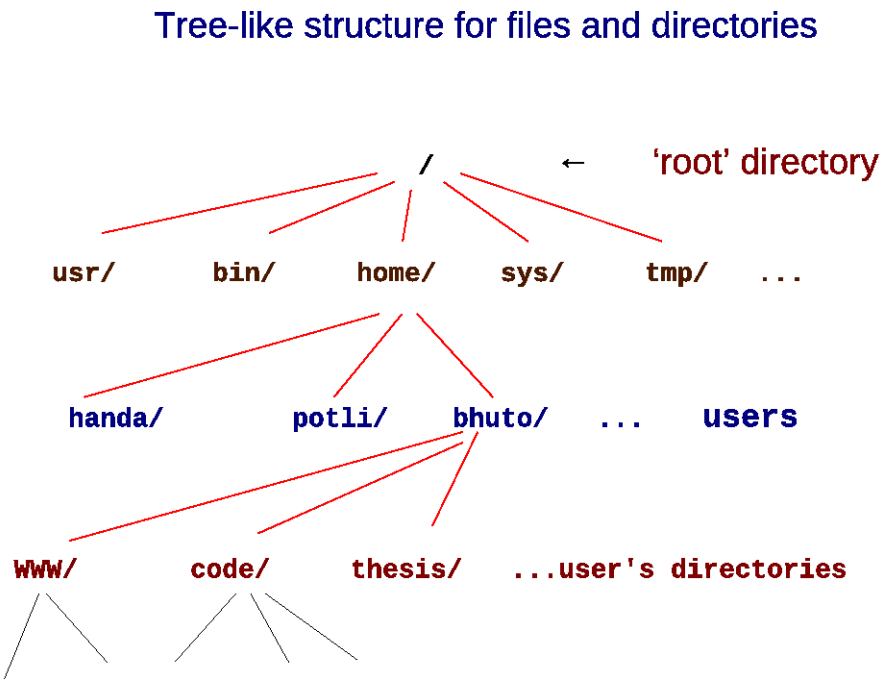


Figure 2.1: Typical directory structure in Linux.

Command `pwd` means `print working directory`.
User *buddhu* is responsible for the files and direcories in the area `/home/buddhu`. It is advisable to create seperate directories for different purposes. Some typical areas could be `Cprograms`, `Photos`, `notes`, `miscellaneous`, ... This is upto the individual user how he/she will organize his/her home area. Let's create a directory:

```
$ pwd
/home/buddhu
$ls
miscellaneous photos xxx xyz ....
$ mkdir CProgram
$ ls
CProgram miscellaneous photos xxx xyz ....
$ cd CProgram
$ pwd
/home/buddhu/CProgram
```

Try this out:

```
$ echo 'This is not a blank file, but a junk one.' > junk1.txt
$ less junk1.txt
```

`less` is used for a quick look into the contents of a text file.

Let us execute some more commands and watch carefully what happens when you use the `ls` command:

```
$ cp junk1.txt junk2.txt
$ ls
$ mv junk2.txt faltu.txt
$ ls
$ less faltu.txt
$ echo 'This is another dumb file.' > faltu-2.txt
$ ls
$ rm faltu.txt
$ ls
$ cp -p faltu-2.txt
$ ls
$ cp junk1.txt junk3.txt
$ ls
$ cp junk1.txt junk4.txt
$ ls
$ rm junk1.txt
$ ls
$ rm junk1.txt junk2.txt junk3.txt junk4.txt
$ ls
```

It is strongly advised not to use the `rm` command with * option at this stage. Instead of `rm junk1.txt junk2.txt junk3.txt junk4.txt` we could have used `rm junk*` with the same effect. But a casual typing mistake like `rm junk *` would delete all the files in that directory.

Use the following commands and examine their results carefully:

```
$ ls
$ ls -a
$ ls -l
$ ls -t
$ ls -lt
$ ls -ltr
```

In a certain directory the last command produced the following output. What are the information you get from this regarding the files and the directory?

```
-rw-r--r--    1 buddhu   users        11977 Jul 25 19:43 LeastSquare.eps
-rw-r--r--    1 buddhu   users        14346 Aug  9 08:11 JacquardLoom.jpg
-rw-r--r--    1 buddhu   users       269647 Aug  9 08:11 JacquardLoom.eps
-rw-r--r--    1 buddhu   users        17397 Aug  9 08:27 EDSAC.jpg
-rw-r--r--    1 buddhu   users       302611 Aug  9 08:36 EDSAC.eps
-rw-r--r--    1 buddhu   users         8993 Aug  9 08:40 PunchedFortranCard.jpg
-rw-r--r--    1 buddhu   users       268558 Aug  9 08:40 PunchedFortranCard.eps
-rw-r--r--    1 buddhu   users      1328218 Aug  9 09:19 Internet_map.jpg
-rw-r--r--    1 buddhu   users      9972695 Aug  9 09:20 Internet_map.eps
```

```
-rw-r--r--    1 buddhu   users        99493 Aug 10 07:00 MScII-comp-jul08.tex~
-rw-r--r--    1 buddhu   users           21 Aug 11 07:05 junk
-rw-r--r--    1 buddhu   users         5072 Aug 11 07:17 MScII-comp-jul08.toc
-rw-r--r--    1 buddhu   users     11238874 Aug 11 07:17 MScII-comp-jul08.ps
-rw-r--r--    1 buddhu   users        99811 Aug 11 07:19 MScII-comp-jul08.tex
```

What will be the result of the following commands issued while in he same directory?

```
$ ls
$ ls -a
$ ls *
$ ls *MSc*
$ ls MSc*
$ ls MS*
$ ls -ltr *.eps
$ ls *ps
```

2.3 Copying, Moving and Deleting Files and Directories

We want to create a directory structure and play with files and directories. Execute the following commands and note the results.

```
$ pwd
/home/buddhu
$ ls
.. .. .. test1.txt test2.txt .. ..
$ mkdir junk1
$ ls
.. .. junk1 .. test1.txt test2.txt .. ..
$ mkdir junk2 junk3
$ ls
.. .. junk1 junk2 junk3 .. test1.txt test2.txt .. ..
$ ls junk1

$ ls junk2

$ ls junk3

$ mkdir junk1/faltu junk1/baje junk1/notun
$ ls
.. .. junk1 junk2 junk3 .. test1.txt test2.txt .. ..
$ls junk1
baje faltu notun
$ mkdir junk1/faltu/pqrs junk1/faltu/abcd
$ mkdir junk2/gablu junk2/hablu junk2/kablu
$ ls junk2
gablu hablu kablu
```

```
$ mkdir junk3/galpo junk3/jokes junk3/messages
$ ls junk3
galpo jokes messages
$ mkdir junk2/hablu/junk2
ls junk2/hablu
$ pwd
/home/buddhu
```

Now try to do the following while staying put in /home/buddhu:
(use relative path and absolute path)
put a copy of test1.txt under /home/buddhu/junk2/gablu
put a copy of test1.txt under /home/buddhu/junk2/gablu with name pqr.txt
put a copy of test2.txt under /home/buddhu/junk2/gablu with name xyz.txt
....

Also do the same while in the directory home/buddhu/junk2/hablu/junk2.
Use absolute and relative paths for the following:
`cp, mv, rm, diff, ls, mkdir`

2.4 More Operations with Files

- Using the file manager for viewing files.
- Viewing text files : `less, more, cat`.
- Printing files `lpr`.
- File attributes, changing file attributes: `chmod`.
- File manipulation : `cat`.
- Use of following : `tee, pipe, grep, *` along with `ls, cat, wc`.
- Moving files and directories around `mv, ftp`.
- Remote Access : `telnet`.
- Execution : running a job in the foreground, background.
- redirecting output of a job to file
- System monitoring `ps, top`

2.5 Moving Around

It is important to move about in the directory structure. Consider the directory structure we have created in the above chapter.
`cd directory`
`cd directory/directory`

`cd directory/..../directory`
will take you to the appropriate directory. The following command

```
$ cd
```

will take you back to the home directory (`/home/buddhu` in this case) wherever you might be in the directory structure.

```
$ cd ..
```

will take one level up from your current position in the directory structure. Similarly

```
$ cd ../..
$ cd ../../..
$
```

will take you *two* and *three* levels up in the directory structure. You can go up any number of levels in the directory structure in this way - provided you are within your home area or, you have the appropriate permissions.
Remember going up in the directory structure is unambiguous, but not going down!

Watch the action of the commands (ask the nearest Linux/Unix literate for explanations should you need any).

```
$ pwd
/home/buddhu
$ cd junk1
$ pwd
/home/buddhu/junk1
$ ls
baje faltu notun
$ cd ..
$ pwd
/home/buddhu
$ cd junk2
$ pwd
/home/buddhu/junk2
$ ls
gablu hablu kablu
$ cd ../junk3
$ pwd
/home/buddhu/junk3
$ cd
$ pwd
/home/buddhu
$ cd junk1/faltu/pqrs
$ pwd
```

pwd	Show present working directory
ls	List files in present working directory
ls -la	List files of present working directory with details
man ls	Show manual page for **ls**. Works for all commands.
man -k *keyword*	Search manual pages for info on "keyword".
cd	Change present working directory to home directory.
mkdir *foo*	Create subdirectory foo
cd *foo*	Change to subdirectory foo (go down in tree)
cd ..	Go up one level in tree
rmdir *foo*	Remove subdirectory foo (must be empty)
emacs *foo* &	Edit file foo with emacs (& to run in background)
more *foo*	Display file foo (space for next page)
less *foo*	Similar to **more *foo***, but able to back up (**q** to quit)
rm *foo*	Delete file foo
cp *foo bar*	Copy file foo to file bar, e.g.,
cp ~bhuto/*foo* ./	copies bhuto's file foo to my current directory
mv *foo bar*	Rename file foo to bar
lpr *foo*	Print file foo. Use **-P** to specify print queue, e.g.,
lpr -Plj1 *foo*	(site dependent).
ps	Show existing processes
kill *345*	Kill process 345 (**kill -9** as last resort)
./*foo*	Run executable program foo in current directory
***ctrl*-c**	Terminate currently executing program
chmod ug+x *foo*	Change access mode so user and group have privilege to execute foo (Check with **ls -la**)

Figure 2.2: This is a list of some commonly used commands in Linux.

```
/home/buddhu/junk1/faltu/pqrs
$ cd ../../../junk2/gablu
$ pwd
.......................
$ cd /home/buddhu/junk3/jokes
$pwd
.......................
$
```

What will be the results of the last two `pwd` commands?

Most Linux/Unix OS provides GUI-based file managers: you can use *drag & drop* for copying and moving files; create directories (often called *folders*). You may not need to use the `terminal`, or the `shell` at all. But it is better to have some idea of the basics.

Chapter 3

Writing a Computer Program

3.1 What Is a Computer Program?

This is an attempt to define a *computer program*, more often known as *code*:
A computer program is a set of *instructions* in a certain *computer language* to perform a set of *operations* on certain *inputs* to produce certain *outputs*.
A very simple example is a program that does the following: $c = a+b$, where a, b are two numbers - *inputs* - which are added to produce a third number - c, the *output*. We shall learn more while we go along.

3.2 Anatomy of a Computer Program

As has been mentioned just now, a *computer program* or *code* should have the following:

- provision for supplying the input(s)
- the operations on the input(s)
- making available the output(s) when the operations are over

In our example of the code $c = a+b$, we can have several options:

- have the values of the *inputs*[1] - a, b supplied inside the code: then the code is specific for these two values of a, b and you have to write a new code if you wish/need to use different values of a, b. That would be very dumb and you are allowed to do that only when you are a beginner.
- have the *inputs* supplied when the code is being executed (*running*) - from a terminal; this is interactive and may be used when the number of inputs is not very large and you will not make a mistake in typing them out at the terminal - good at the developing and testing stage of a code when you need to see how every line of the code is performing;

[1] called simply inputs from now on

- have the values supplied through a file: better and more accurate way of doing things, particularly when you have developed a steady version of the code and you need to supply a large number values as inputs - this makes the code run faster;
- in even more interesting cases, the inputs may come from device(s) attached to the computer, which provide the inputs when asked by the computer in course of running the code - bar code readers attached to a billing machines in stores are examples of *input devices*;
- the output may be shown on the screen;
- depending on the need it may be written out to a *file*;
- in some situations the output may be available to some devices attached to the computer which may be used as input for some other process - printers and displays attached to billing machines in stores are examples of such *output devices*.

We shall restrict ourselves to the cases where the inputs are supplied in the program itself, or given from the terminal, or supplied from an input file. Similarly, the output will shown on the screen or written on a file. The codes are written in *C* which is a *low level* language [2]. It is a very popular language for which some *compiler* [3] will be available on any Linux/Unix platform.

3.3 Writing a Code

Often you will be asked to write *codes* which will be much more complicated than addition or subtraction of numbers. In complicated problems writing the actual code is only a small part of the whole exercise.

- Logical formulation of the problem is the most important part. Once you formulate the logical structure of the problem, writing the code becomes easy.
- prepare a *flowchart* of the problem - symbolic representation of the soultion
- If the code is complicated break it up into smaller modules - functions, subroutines, etc., assigning limited task/functionality to each module:
  - writing the code becomes easier
  - it becomes easier to read and understand the code
  - mistakes are easy to locate and correct
  - if the code is being developed collaboratively, it is easy to share
- each module, or even a set of instructions within a module should be clearly documented within the code itself - each language provides ways of inserting *comment* lines within the code.

[2] Naively speaking a high level computer programming language sounds more like the English language, FORTRAN is an example. If it is less like English it is a lower level language.

[3] A *compiler* converts a code written in a *high level* or a *low level* language into the *executable* - a set of instructions in *machine language* which the computer actually implements. So, the compiler is specific for the language and the operating system.

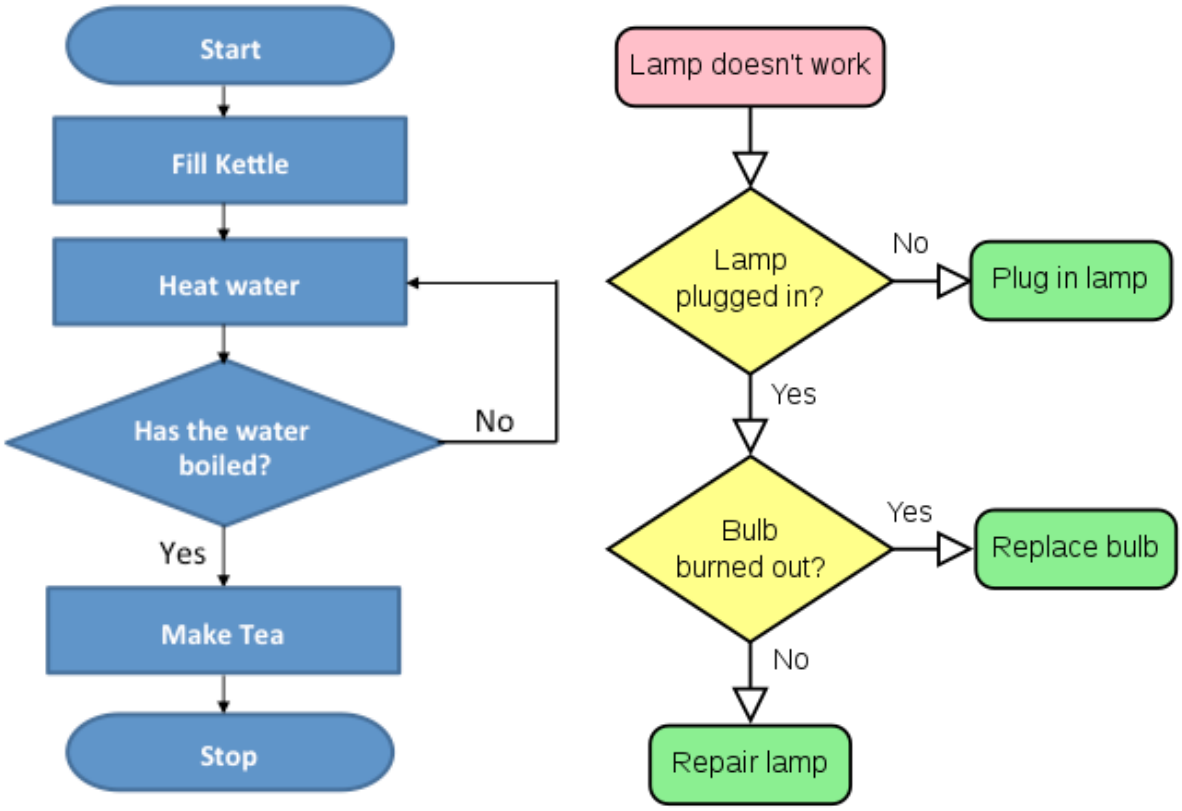


Figure 3.1: Boiling water for making a cup of tea.

3.3.1 Program Flowchart

Consider the process of boiling water for making a cup of tea or fixing a lamp. What we do instinctively may be described by the diagram in figure 3.1. Flowchart is a diagrammatic representation of an algorithm. Flowcharts are very helpful in writing program and explaining programs to others. It is also the simplest way to figure out the bugs in a program before carrying out, which saves a lot time, labor and money. A flowchart for the repeatitive loop is shown in figure 3.2 (left). It also shows a flowchart for generating the Fibonacci series upto 1000.

Different symbols are used for different states in flowchart, for example: Input/Output and decision making have different symbols. Figure 3.3 describes all the symbols that are used in making a flowchart.

3.4 The C++ Language

Language C developed (from B) around 1970 at Bell Labs and is used to create parts of the Unix OS. C++ is derived from C in early 1980's by Bjarne Stroustrup - it is "C with classes", i.e., user-defined data types that allow "Object Oriented Programming". We shall try to understand these terms as we get along with developing codes.

Like C, C++ is case sensitive - variable `a` not the same as `A`. Currently most widely used programming language in High Energy Physics and many other science/engineering fields.

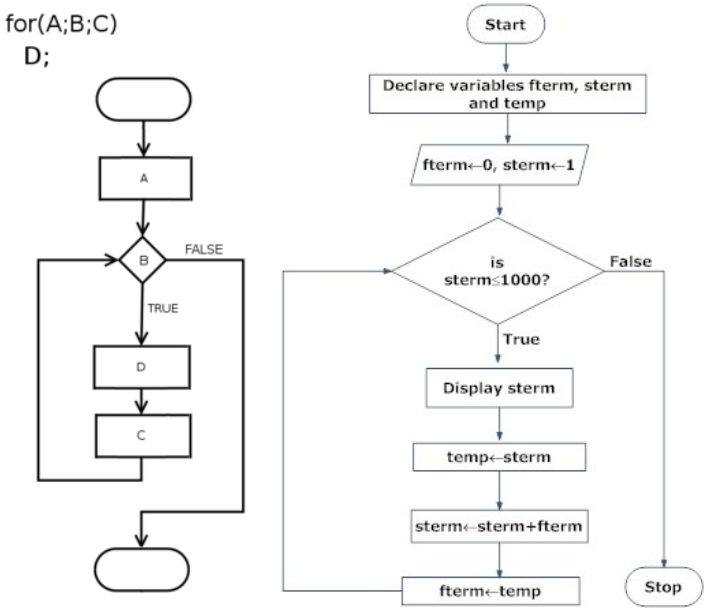


Figure 3.2: Flowcharts for the repetitive loop (left) and Fibonacci series upto 1000 (right) are shown.

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for airthmetic operations and data-manipulations.
	Desicion	Used to represent the operation in which there are two alternatives, true and false.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect flowchart portion on different page.
	Predefined Process/Function	Used to represent a group of statements performing one processing task.

Figure 3.3: Different symbols used for different states in a flowchart.

3.4.1 The First Program: Most Naive

Let us write our first program for adding two numbers. Use your favourite editor - `vi`, `vim`, `gvim`, `emacs`, `xemacs`, `pico`, `gedit`, `..` to create a text file which contains the code. The most naive version will have the following lines. Let's give the file the name *add.cpp*

```
// A very naive program to add two numbers --------------
#include<iostream>
using namespace std;
int main()
{
  /* A very naive
     program to
     add two numbers -------------- */
  /* This code is written by Manas Maity, 25/08/2015 */
  int a, b, c;
  a = 2;
  b = 3;
  c = a+b;
  cout << "a = " << a << ", b = " << b << ", c = a+b = " << c << "\n";
  return 0;
}
```

3.4.2 The Basics of a C++ Program

Now let's examine the code:

- The very first line
  `// A very naive program to add two numbers --------------`
  is a comment line. Within a line anything followed by `//` is a comment. Comments are not executable part of the code. Yet they are very important for documentation. Similarly
  `/* A very naive`
  `program to`
  `add two numbers -------------- */`
  is also a comment spanning several lines and so is the following:
  `/* This code is written by Manas Maity, 25/08/2015 */`
  You should include enough comments in your code to make it understandable by someone else (or by yourself, later). Each file should start with comments indicating author's name, main purpose of the code, required input, etc.

- `#include<iostream>` simply tells the compiler that you need the *utilities* to *read* the input data (here from the *keyboard*) and *write* the output data to the screen. We shall use several such `#include` statements as we progress.

- `main()` tells the OS where the program starts

- `return` tells the OS where to *stop/exit* from the code.

- `a, b, c` are the *variables*. Variables are like containers to store different types of data - integer number (`2`), floating point number(`2.1`), characters (`d`), ... see figure 3.4 for basic data types used in `C++` .
- `C++` requires every variable be *declared* before it is used.
  `int a, b, c;`
  declares three variables of integer type. For a beginner it is preferable to declare the variables at the beginning of the `main` or the *subroutine* where it has been used.
- You must *initialize* (assigne actual value) the variables before they are used.
- `a = 2;` assigns the value `2` to the variable `a`. Similarly for the variable `b`.
- `c = a+b;` adds the values stored in `a` and `b` and assigns the sum, `5` in this casw to the variable `c`.
- Note that `c` has been assigned a value before it is used in the next statement.
- `cout` is the standard output stream in `C++` . This is made available through *standard library* - `using namespace std`
- `cout << "a = " << a << ", b = " << b << ", c = a+b = " << c << "\n";`
  displays the output on the terminal - `\n` tells that ytou want to start a new line after this. You can omit the `\n` and see the difference it makes.
- `=, +,` are operators. A list of operators are given in table 3.1

Now let's compile the code. Type the following command at the terminal

```
$ g++ add.cpp -o add.exe
```

This means you are asking the compiler (`gcc`) to compile the code `add.cpp` and produce the *executable* file `add.exe` which the OS can execute.

Now let's run the code - you will get the following on your terminal:

```
$ ./add.exe
a = 2, b = 3, c = a+b = 5
$
```

3.4.3 Data Types & Operators

Data types and operators in `C++` are shown in figure 3.4 and table 3.1 respectively. When an expression has multiple operators, the following rules apply:

1. `*` and `/` have precedence over `+` and `-` — `x*y + u/v` ≡ `(x*y) + (u/v)`
2. `*` and `/` have same precedence, carry out left to right — `x/y/u*v` ≡ `((x/y) / u) * v`
3. Similar for `+` and `-` — `x - y + z` ≡ `(x - y) + z`

Name	Description	Size*	Range*
`char`	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
`short int` (`short`)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
`int`	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`long int` (`long`)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean value. It can take one of two values: true or false.	1byte	true or false
`float`	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
`double`	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`long double`	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`wchar_t`	Wide character.	2 *or* 4 bytes	1 wide character

Figure 3.4: Basic data types used in `C++` .

`=`	assignment	`a = 5;` assigns the value 5 to the variable `a`
		`a = b;` assigns the current value of `b` to `a`
		`a = b = 5.1;` assigns the value `5.1` to `a` and `b`

Arithmatic Operators		
`+`	addition	`b+c`, `a+5`, `c+d+f`
`-`	substraction	`b-c`, `a-10.1`, `c+d-f`
`*`	multiplication	`b*c`, `a*5.2`, `(c+d)*f`
`/`	division	`b/c`, `a/5.2`, `(c*d+5.3)/f`
`<<`	insertion	`cout << a;` displays on the *standard output* (terminal) the content of `a` `cout << "I am happy.";` displays on the terminal `"I am happy."`

`>>`	extraction	`cin >> a;` extracts from the *standard input* the value to be assigned to `a`
		`cin >> a >> b;` the first value extracted is assigned to `a`
		and the next to `b`

Table 3.1: Some commonly used operators in `C++`

3.4.4 Less Naive

As you have seen, the code `add.cpp` can add only `2` and `3`. So, if you have to add say, `4` and `5` you have to edit the code and compile and then only it will work. Pretty inefficient even if you are adding very large numbers. Instead of supplying the *values* of `a` and `b` in the code itself we can supply them at the time of *running* the code: then the *code* becomes more general and more efficient:

```
// A less naive program to add two numbers --------------
#include<iostream>
using namespace std;
int main()
{
  int a, b, c;
  cout << "Give a, b\n";
  cin >> a >> b;
  c = a+b;
  cout << "a = " << a << ", b = " << b << ", c = a+b = " << c << "\n";
  return 0;
}
```

- `cout << "Give a, b \n";` is strictly not necessary, if you *remember* when to give which input from the terminal, but for most codes you may have to provide many inputs and it is not possible even for the *programmer* to remember the sequence in which they are needed. So, it is better to *print* on the screen which inputs are required and in what sequence.
- The code asks for `a`, `b` - the numbers to be added - *input*.
- The `cin` is used for reading *input(s)* from the *terminal* and assigning the inputs to the appropriate variables (from left to right).

When you *run the code* this is what happens:

```
$ ./add.exe
Give a, b
2 3
a = 2, b = 3, c = a+b = 5
$
```

3.4.5 Smart

Let's extend the code to add as many such pairs of numbers as you would like.

```
/* A smarter code for adding pairs of numbers.
   Manas Maity. 20/08/2015 */

#include<iostream>
```

```
using namespace std;

int main()
{
  float a, b, c;
  char ans[5];

  cout << "Want to add two numbers? (y/n)\n";
  cin >> ans;

  while(ans[0] == 'y' || ans[0] == 'Y')
    {
      cout << "Give a, b \n";
      cin >> a >>  b;
      c = a+b;
      cout << "a = " << a << " b = " << b << " c = a+b = " << c << "\n";

      cout << "Want to add two more numbers? (y/n)\n";
      cin >> ans;
    }

  return 0;
}
```

And this is how the code is executed.

```
$gcc add.c -o add.exe
$./add.exe
Want to add two numbers? (y/n)
y
Give a, b
2.1 3.4
a = 2.1, b = 3.4, c = a+b = 5.5
Want to add two more numbers? (y/n)
y
Give a, b
3.7 -4.1
a = 3.7, b = -4.1, c = a+b = -0.4
Want to add two more numbers? (y/n)
n
$
```

Note the new features in the code:

- `float a, b, c` - floating point numbers
- `char ans[5]` - we have used character variable - `ans` can hold at most a *string* of 5 characters.
- `ans[0]` represents the 0th element of the string `ans`;

- for repeating the same action we have used a `while` *loop*. The set of instructions contained between the curly brackets { ... }

```
{
  cout << "Give a, b \n";
  cin >> a >>  b;
  c = a+b;
  cout << "a = " << a << " b = " << b << " c = a+b = " << c << "\n";

  cout << "Want to add two more numbers? (y/n)\n";
  cin >> ans;
}
```

  is executed till the *condition* `(ans[0] == 'y' || ans[0] == 'Y')` is *true*.

- `ans[0] == 'y'` and `ans[0] == 'Y'` are two different conditions
- `(ans[0] == 'y' || ans[0] == 'Y')` means if either (or both) of the separate conditions is *true* then the whole condition is true. `||` is equivalent to the logical OR.
- `while` loop is used when you *a priori* don't know how many times you have to repeat a set of instructions.

3.5 Average of Numbers

The following is a very simple code to find the average of a few numbers. The numbers are to be given one at a time at the terminal. The outcome of the code is printed on the screen.

```
/* -------- this is a program to calculate average of numbers --------- */
/* -------------- the numbers are given at the terminal --------------- */
/* ---------------------- Manas Maity 21/08/2015 ---------------------- */
#include<iostream>
using namespace std;

int main()
{

  int i, ndata;
  float x, sum=0.0, average;

  cout << "How many numbers do you have? \n";
  cin >> ndata;

  cout << "Give the numbers \n";
  for(i=0; i<ndata; ++i)
    {
      cin >> x;
      sum=sum+x;
```

```
    }

  average = sum/ndata;
  cout << "The sum of numbers     = " << sum << "\n";
  cout << "The average of numbers = " << average << "\n";

  return 0;
}
```

Now it is not convenient to give the numbers at the terminal. So, next we modify the previous code to read the numbers (input) from a *file*, say `average.inp`. The outcome of the code is printed to a *file* `average.out`.

```
/* -------- this is a program to calculate average of numbers --------- */
/* -------------- the numbers are given at the terminal --------------- */
/* ---------------------- Manas Maity 21/08/2015 ---------------------- */
#include<iostream>
#include <fstream>
using namespace std;

int main()
{
  int i, ndata;
  float x, sum=0.0, average;
  ifstream inFile;
  ofstream outFile;
  inFile.open("average.inp");
  outFile.open("average.out");

  inFile >> ndata;

  for(i=0; i<ndata; ++i)
    {
      inFile >> x;
      sum=sum+x;
    }

  average = sum/ndata;
  outFile << "The sum of numbers     = " << sum << "\n"
     << "The average of numbers = " << average << "\n";

  inFile.close();
  outFile.close();
  return 0;
}
```

Note the new features in the code:

- `#include <fstream>` is needed for handling files

- `ifstream inFile;` an *object* `inFile` is declared and a physical file `average.inp` is associated with it. Throughout the code `average.inp` will be referred to by the object `inFile`. `average.inp` has to be an existing file.
- Same for `average.out` referred to by `outFile`. `average.out` will be created during the excution of the code or, if it already exists it will be overwritten.
- When the files are no more needed, they may be closed - but this is optional. The files are closed when execution of the program is complete.
- A more compact form of the same code is given below.

```
#include<iostream>
#include <fstream>
using namespace std;

int main()
{

  int i, ndata;
  float x, sum=0.0, average;
  ifstream inFile("average.inp");
  ofstream outFile("average.out");

  inFile >> ndata;

  for(i=0; i<ndata; ++i)
    {
      inFile >> x;
      sum=sum+x;
    }

  average = sum/ndata;
  outFile << "The sum of numbers     = " << sum << "\n"
      << "The average of numbers = " << average << "\n";

  return 0;
}
```

Note that `cin` and `cout` relates to the *standard input* and *standard output*, ie, the terminal.

Exercise: Expand this code to calculate the mean $\bar{x}$, r.m.s. $\overline{x^2}$ and variance $\overline{(\bar{x}-x)^2}$.

3.6 Evaluating the Exponential

3.6.1 The Ugly: Simple Minded

We end this part with an example of writing a better code - compact and efficient. Let's evaluate e^x.

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$

First we valuate the following series up to x^5. We can use the formula straight forward, and the program would look like:

```
/* -------------- simple program to calculate exp(x) ---------------- */
#include<iostream>
#include<math.h>
using namespace std;

int main()
{
  int i;
  float x, expx, term;

  /* ------------------- read in the value of x ---------------------- */
  cout << "Give the value of x \n";
  cin >> x;

  /* ------------------------ compute exp(x) ------------------------- */
  expx = 1.0 + x + pow(x,2)/2. + pow(x,3)/6. + pow(x,4)/24. + pow(x,5)/120.;

  /* ---------------- write the result on the terminal -------------- */
  cout << "The value of exp(" << x << ") = " << expx << "\n";

  return 0;
}

$ g++ Exponential-Bad.cpp -o Exponential-Bad.x
$ ./Exponential-Ugly.x
Give the value of x
1.0
My Calculation:   exp(1) = 2.71667
Library function: exp(1) = 2.71828
```

- `#include<math.h>` neded for mathematical functions
- `pow(x,2)` - a mathematical function is used to evaluate x^2.

3.6.2 The Bad: Sophisticated

The code above is clearly simple minded, and can't be done for large number of terms and the result will limit the accuracy of e^x. A better method would be to use a loop to implement the sum for N terms, where N is given:

```
/* --------------------- Exponential-Bad.cpp ----------------------- */
/* ------------------ program to calculate exp(x) -------------------- */
#include<iostream>
#include<math.h>
using namespace std;

int fact(int n)
  /* ------------------ calculates n! for integer n* ----------------- */
{
  int i, factor;
  factor = 1;
  if(n==0) return factor;

  for(i=1; i<=n; ++i)
    {
      factor = factor*i;
    }
  /* -------------- This is a diagonstic print statement ------------- */
  cout << n << "! = " << factor << "\n";
  return factor;
}

main()
{
  int i, N;
  float x, expx, term;

  /* -------------------- read in the value of N ---------------------- */
  cout << "How many terms do you want to add? \t";
  cin >> N;
  /* -------------------- read in the value of x ---------------------- */
  cout << "Give the value of x \n";
  cin >> x;

  /* ------------------------ compute exp(x) ------------------------- */
  /* -------------------------- initialize --------------------------- */
  expx = 0.0;

  /* ------------------------ start the loop ------------------------- */
  for(i=0; i<N; ++i)
    {
      expx = expx + pow(x,i)/fact(i);
    }
```

```
  /* ---------------- write the result on the terminal -------------- */
  cout << "My Calculation:   exp(" << x << ") = " << expx << "\n";
  /* ----------- compare the result with the standard result --------- */
  cout << "Library function: exp(" << x << ") = " << exp(x) << "\n";

  return 0;
}

$ g++ Exponential-Bad.cpp -o Exponential-Bad.x
$ ./Exponential-Bad.x
How many terms do you want to add?  10
Give the value of x
1.0
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
My Calculation:   exp(1) = 2.71828
Library function: exp(1) = 2.71828
```

- we are using a *function*! The following piece of code takes as *input* an integer (n in this case) and returns an integer ($factor$) as *output* as declared by `int fact(int n)`
- Typically when a piece of code is used repeatedly we separate it out as a function. Also, it makes a very large piece of code modular, and easy to handle, understand and debug.
- We have used a diagnostic print statement to check the output of the function `fact`. This may be commented out later.

```
int fact(int n)
/* ------------------------ This is a function ---------------------- */
/* ------------------ calculates n! for integer n ------------------ */
{
  int i, factor;
  factor = 1;
  if(n==0) return factor;

  for(i=1; i<=n; ++i)
    {
      factor = factor*i;
    }
  /* -------------- This is a diagonstic print statement ------------- */
  cout << n << "! = " << factor << "\n";
```

```
  return factor;
}
```

3.6.3 The Good: Sophisticated, Simple and Efficient

Note, in the previous code you had to write a function for calculating the factorial $i!$. As i increases both evaluating $i!$ and `pow(x,i)` take more and more computing time[4]. Also, $i!$ will give *integer overflow* for a rather modest value of i although the contribution of the ith term may be quite small (depending on the value of x).

We can do better using the recursive relation:

$$t_i = \frac{x}{i} t_{i-1}, \quad t_0 = 1.0$$

```
/* ---------------------- Exponential-Good.cpp ----------------------- */
/* ------------------ program to calculate exp(x) -------------------- */
/* -------------------- Manas Maity 25/08/2015 ----------------------- */
#include<iostream>
#include<math.h>
using namespace std;

main()
{
  /* ----------------- declare the variables first -------------------*/

  int i, N;
  double x, expx = 0.0, term;

  /* ------------------ read in the value of N, x -------------------- */
  cout << "How many terms do you want? \n";
  cin >> N;

  cout << "Give the value of x \n";
  cin >> x;

  /* ------------------------ compute exp(x) ------------------------- */
  /* ------------ the first term has to be done by hand ------------- */
  term = 1.0;
  expx = expx + term;

  for(i=1; i<N; ++i)
    {
      term = term*x/i;
      expx = expx + term;
    }
```

[4]This may not be obvious when you calculate the value for a single number but if it is used many times, say 10^5 times in a complex computation, you will see the difference

```
  /* ---------------- write the result on the terminal -------------- */
  cout << "My Calculation:   exp(" << x << ") = " << expx << "\n";
  /* ----------- compare the result with the standard result --------- */
  cout << "Library function: exp(" << x << ") = " << exp(x) << "\n";

  return 0;
}
```

Similar recursive relation (you have to be a bit more careful here though) has been used in the following code to implement the series for *sin(x)* and *cos(x)*.

```
/* ------------------------ SineCosine.cpp -------------------------- */
/* ------------------ program to calculate exp(x) -------------------- */
/* -------------------- Manas Maity 25/08/2015 ----------------------- */
#include<iostream>
#include<math.h>
using namespace std;

main()
{
  /* ------------ Compute sin(x) & cos(x) upto 10 terms ------------- */

  int i;
  float x, sineterm, costerm, sinex, cosx;


  /* ------------------- read in the value of x ---------------------- */
  cout << "Give the value of x in radian \t";
  cin >> x;

  /* ------------------- compute sin(x), cos(x) ---------------------- */

  cosx     = 0.0;
  sinex    = 0.0;
  costerm  = 1.0;
  sineterm = x;
  cosx     = cosx  + costerm;
  sinex    = sinex + sineterm;

  for(i=1; i<10; ++i)
    {
      costerm = -(costerm*x*x)/(2.0*i*(2.0*i-1));
      sineterm = -(sineterm*x*x)/(2.0*i*(2.0*i+1));
      cosx  = cosx  + costerm;
      sinex = sinex + sineterm;
    }

  cout << "The value of sin(" << x << ") = " << sinex << "\n";
```

```
  cout << "The value of cos(" << x << ") = " << cosx << "\n";

  return 0;
}
```

Now instead of evaluating upto a fixed number of terms, let us evaluate $\sin(x)$ and $\cos(x)$ upto a given accuracy, say 0.0001.

```
/* -------------------- SineCosine-Accuracy.cpp ---------------------- */
/* ------------------ program to calculate exp(x) -------------------- */
/* -------------------- Manas Maity 25/08/2015 ----------------------- */
#include<iostream>
#include<math.h>
using namespace std;

main()
{
  /* --------- Compute sin(x) & cos(x) upto a given accuracy --------- */
  int i;
  float x, sineterm, costerm, sinex=0.0, cosx=0.0, dx;


  /* ------------------- read in the value of x ---------------------- */
  cout << "Give the value of x in radian \t";
  cin >> x;
  cout << "What accuracy do you want?  \t";
  cin >> dx;

  /* ------------------- compute sin(x), cos(x) ---------------------- */
  costerm  = 1.0;
  sineterm = x;
  cosx     = cosx  + costerm;
  sinex    = sinex + sineterm;

  i = 1;
  while(fabs(sineterm) > dx)
    {
      sineterm = -(sineterm*x*x)/(2.0*i*(2.0*i+1));
      sinex = sinex + sineterm;
      i++;
    }

  i = 1;
  while(fabs(costerm) > dx)
    {
      costerm = -(costerm*x*x)/(2.0*i*(2.0*i-1));
      cosx  = cosx  + costerm;
      i++;
    }
```

```
  cout << "The value of sin(" << x << ") = " << sinex << "\n";
  cout << "The value of cos(" << x << ") = " << cosx << "\n";

  return 0;
}
```

Finally let us use the above to plot *sin(x)* or *cos(x)* vs *x*. Here, we need to evaluate *sine(x)* many times. So, we define a function and call it as and when we need it! Also, we need to write the values of *x, sin(x), cos(x)* to a *file* for convenience. Note we have defined `functions` and directly printed the values just by calling them in the *print* statement.

```
/* -------------------- SineCosine-Function.cpp ---------------------- */
/* ------------------ program to calculate exp(x) -------------------- */
/* -------------------- Manas Maity 25/08/2015 ----------------------- */
#include<iostream>
#include<math.h>
#include <fstream>

using namespace std;

float sine(float x, float dx)
/* function to calculate sin(x) */
{
  int i;
  float sineterm, sinex;

  sinex    = 0.0;
  sineterm = x;
  sinex    = sinex + sineterm;

  i = 1;
  while(fabs(sineterm) > dx)
    {
      sineterm = -(sineterm*x*x)/(2.0*i*(2.0*i+1));
      sinex = sinex + sineterm;
      i++;
    }

  return sinex;
}

float cosine(float x, float dx)
/* function to calculate cos(x) */
{
  int i;
  float costerm, cosx;

  cosx     = 0.0;
  costerm  = 1.0;
```

```
  cosx     = cosx  + costerm;

  i = 1;
  while(fabs(costerm) > dx)
    {
      costerm = -(costerm*x*x)/(2.0*i*(2.0*i-1));
      cosx  = cosx  + costerm;
      i++;
    }
  return cosx;
}

main()
{
  /* ---- Compute the values of sin(x), cos(x) upto given accuracy ---- */
  int i, nDivision;
  float xLow, xHigh, xDiv, x, precision;

  /* ----------------- Write the output to a file -------------------- */
  ofstream outFile("SineCosine-Plot.txt");

  /* --------------------- read in the values ------------------------ */
  cout << "Give the low and high values of x, # of divisions\t";
  cin >> xLow >> xHigh >>  nDivision;
  cout << "What accuracy do you want \t";
  cin >> precision;

  /* ------ compute sin(x), cos(x) and write to a file nicely -------- */
  xDiv = (xHigh - xLow)/nDivision;
  for(i=0; i<=nDivision; ++i)
    {
      x = xLow + i*xDiv;
      outFile << i << "  " <<  x << "  " << sine(x, precision) << "  " <<
cosine(x, precision) << "\n";
    }

  return 0;
}
```

Plot the data in the file `sinusoid-plot.txt` using `gnuplot`. See Chapter 5 in this workbook or `www.gnuplot.info` for more on gnuplot.

```
$ gnuplot

G N U P L O T
Version 4.6 patchlevel 4    last modified 2013-10-02
Build System: Linux x86_64

Copyright (C) 1986-1993, 1998, 2004, 2007-2013
```

```
Thomas Williams, Colin Kelley and many others

gnuplot home:     http://www.gnuplot.info
faq, bugs, etc:   type "help FAQ"
immediate help:   type "help"  (plot window: hit 'h')

Terminal type set to 'wxt'
gnuplot> set grid
gnuplot> plot "SineCosine-Plot.txt" u 2:3 w p
gnuplot> plot "SineCosine-Plot.txt" u 2:4 w lp
gnuplot> q
```

3.7 Pointers

Pointers are an important feature of the C and C++ language. Let's start with a simple code to introduce the pointers. Subsequent examples will elaborate features and uses of pointers.

```
#include <iostream>

using namespace std;

int main ()
{
   int  var = 20;   // actual variable declaration.
   int  *ip;        // pointer for an integer variable

   ip = &var;       // store address of var in pointer variable

   cout << "Value stored in 'var' :        ";
   cout << var << "\n";

   // print the address stored in ip pointer variable
   cout << "Address stored in ip variable: ";
   cout << ip << "\n";

   // access the value at the address available in pointer
   cout << "Value of *ip variable:         ";
   cout << *ip << "\n";

   return 0;
}
```

Pointers can be used for arrays of all sorts, structures, ...

```
% ------------------------------------------------------------------------
#include <iostream>
```

```
using namespace std;
const int MAX = 3;

int main ()
{
   double  var[MAX] = {10.1, 100.1, 200.2};
   double * ptr;

   // let us have array address in pointer.
   ptr = var;    //   ptr = &var[0]; // These two are equivalent statements

   cout << "In usual order --------------------\n";
   for (int i = 0; i < MAX; i++)
   {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;

      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;

      // point to the next location
      ptr++;
   }

   cout << "Now in reverse order --------------\n";
   ptr = &var[MAX-1];
   for (int i = MAX-1; i >= 0; i--)
   {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;

      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;

      // point to the previous location
      ptr--;
   }

   return 0;
}
% ------------------------------------------------------------------------
```

3.8 Simple Problems

1. Write a program which reads two integers. If the first integer is less than the second, print the message *UP*. If the second is less than the first, print the message *DOWN*. If the numbers are equal, print the message *EQUAL*. If there is an error reading the data, print the message *ERROR*.

2. Write a code to solve the equation $1 - e^{-x} = x/5$.
Hints: write a code to evaluate the values of both sides of the equation and see for what value of x the difference of these two is smaller than a small number, say .001.

3. (a) Write a program to read in N numbers from the terminal and compute the average, variance, maximum, and minimum values.
   (b) Modify the above code to read the numbers from a file where N is unknown.

4. Evaluate the following series up to 10 terms.
   (a) $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$
   (b) $cosx = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + ...$
   (c) $sinx = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} + ...$

5. Repeat the above problems till the result converges within .0001.

6. Evaluate sin(x) using the above formula for 100 equally spaced values of x ($0.0 < x < 2\pi$) and plot using *gnuplot*.

7. Read an integer value, representing the month. Print out the name of that month.

8. Read integer values from a file, representing dates and write the corresponding dates to a file. For example *20032008* represents *March 20, 2008*.

9. Write a program to read a *float* representing the temperature in °C, and print as a *float* the equivalent temperature in °F. Print your results such as 100.0 °C converts to 212.0 °F.
Modify the code to plot temperatures in °F -vs °C and find the temperature at which both scales give the same reading.

10. Evaluate the Fibonacci series up to 10 terms. $F(i) = 1, 1, 2, 3, 5, 8....$ and ratio of $\frac{F(i+1)}{F(i)}$ as a function of i.

11. Read the roll number, name, marks in Bengali, English, Maths, Physics, Chemistry and Biology of 10 students from a file. If a student scores less than 40 in any subject he/she is considered *Failed*, otherwise *Passed*. Create a file to write their roll number, subject marks, grand total and status (*Passed*, or *Failed*).

12. (a) Write a program to read in the dimensions of two matrices A and B and check if the following operations are possible A+B, A-B, AB.
   (b) Read the elements of two matrices A, B from a file and write C in matrix form, $C = A+B$.
   (c) Read the elements of two matrices A, B from a file and write C in matrix form, $C = AB$.

13. Write a program to read the elements of a matrix A and write A and A^T in suitable format.

14. Read the elements of three matrices, A, B, C. Compute $D = AB$, $E = BC$, $F = DC$ $G = AE$. Write F and G and check if they are equal. *You are checking the Associative Law of matrix multiplication.*

15. Read the elements of three matrices, A, B, C. Compute $D = A+B$, $E = B+C$, $F = D+C$ $G = A+E$. Write F and G and check if they are equal. *You are checking the Associative Law of matrix addition.*

16. Read the elements of a square matrix and check if it is symmetric or antisymmetric.

17. For individual incomes up to *Rs. 1,00,000* there is no tax, for next *Rs. 50,000* the rate is *10%*, next *Rs. 50,000* the rate is *20%* and above that the rate is *30%*. Write a program to

    - calculate the tax of one person at a time interactively.
    - calculate taxes of persons using a file as input

18. Read a text from a file, count the number of *a*, *e*, *i*, *o* & *u* and write the result in a different file.

19. In a coin toss experiment, the results *HEAD* and *TAIL* are equally likely. Generate random numbers to simulate an experiment of 100 tosses and find the fraction of times you get *HEAD* and plot the result of 100 such experiments using *gnuplot*.

20. Use random numbers to find out the area of a circle.

21. Write a program to read the name, age, educational qualification, and annual income of 2000 people from a file and generate the data for a histogram; plot using *gnuplot*. Also, find the *mean* and *variance* of the income.

22. Prove the laws of reflection using Fermat's Principle of propagation of light. [Take a starting point A and an end point B for the ray of light. Consider a point O on a straight line and let the ray of light take the path $\vec{AO}, \vec{OB}$. A, O, B should be on the same plane (say xy). Calculate the path length $l = |\vec{AO}| + |\vec{OB}|$. Now vary the position of O on the straight line so that l is minimum and check that it satisfies the laws of reflection.]

23. Write a code for evaluating

$$e_N^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^N}{N!}$$

    - Vary N and plot the difference $e_N^x - e^x$ for a fixed value of x where e^x is evaluated using the appropriate math library function.
    - Also check the same for fixed N varying x.

24. Write a code to solve the following equation graphically:

$$1 - e^{-x} = \frac{x}{4}$$

25. - Find the Poisson probabilities for various integer values n for a given *mean* or *expectation* value $\lambda = 3$ and plot them using `Gnuplot` (λ may be any +ve number).

$$P(n|\lambda) = \frac{\lambda^n e^{-\lambda}}{n!}$$

- Plot $P(n|\lambda)$ for $n = 3,\ 4,\ 5$ varying λ.

26. For an observed count n in an experiment which follows Poisson distribution the *likelihood* of the *true* value of the *expectation* is λ is given by

$$\mathcal{L}(\lambda|n) = \frac{\lambda^n e^{-\lambda}}{n!}$$

Calculate numerically λ_U such that for $n = 0$

$$\int_0^{\lambda_u} \mathcal{L}(\lambda|0)d\lambda = .95$$

27. Consider a wave packet which is a superposition of three free particle wave functions.

$$\psi(x) = A\left(\frac{1}{2}e^{i4x} + e^{i5x} + \frac{1}{2}e^{i6x}\right) = Ae^{i5x}\left(\frac{1}{2}e^{-ix} + 1 + \frac{1}{2}e^{ix}\right) = Ae^{i5x}\,(1 + cosx)$$

- Plot the real part of $\psi(x)$ .vs. x.
- Plot $|\psi(x)|^2$ .vs. x.

28. The wave function of a harmonic oscillator is given by

$$\psi_n(\xi) = \left(\frac{m\omega}{\pi\hbar}\right)^{1/4}\left(\frac{1}{2^n n!}\right)^{1/2} H_n(\xi)e^{-\xi^2/2}$$

For an oscillator with $m\omega = 16\pi\hbar$

- Plot $\psi_0(\xi)$ $.vs.$ ξ for $|\xi| \leq 1$, $H_0(\xi) = 1$
- Plot $\psi_1(\xi)$ $.vs.$ ξ for $|\xi| \leq 3$, $H_1(\xi) = 2\xi$
- Plot $\psi_2(\xi)$ $.vs.$ ξ for $|\xi| \leq 5$, $H_2(\xi) = 4\xi^2 - 2$

Chapter 4

Numerical Methods With C++

4.1 Least Square Fitting

4.1.1 What is ...

Least Square Fitting (LSF) is a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets ("the residuals") of the points from the curve. The sum of the squares of the offsets is used instead of the offset absolute values because this allows the residuals to be treated as a continuous differentiable quantity. However, because squares of the offsets are used, outlying points can have a disproportionate effect on the fit, a property which may or may not be desirable depending on the problem at hand.

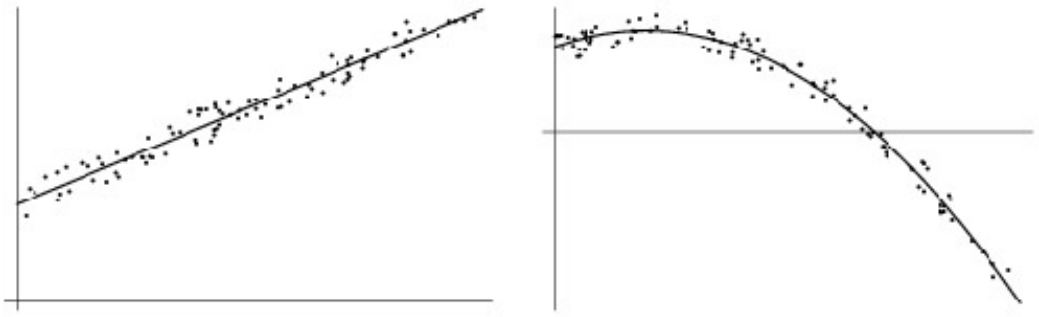


Figure 4.1: Examples of fitted lines through data points.

In practice, the vertical offsets from a line (polynomial, surface, hyperplane, etc.) are almost always minimized instead of the perpendicular offsets. This provides a fitting function for the independent variable X that estimates y for a given x (most often what an experimenter wants), allows uncertainties of the data points along the x- and y-axes to be incorporated simply, and also provides a much simpler analytic form for the fitting parameters than would be obtained using a fit based on perpendicular offsets. In addition, the fitting technique can be easily generalized from a best-fit line to a best-fit polynomial when sums of vertical distances are used. In any case, for a reasonable number of noisy data points, the difference between vertical and perpendicular fits is quite small. Vertical least squares fitting proceeds by finding the sum of the squares of the vertical

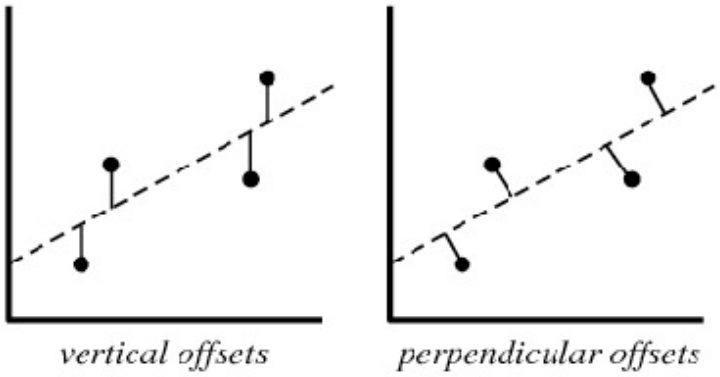


Figure 4.2: Different definitions of the deviations from the hypothesis.

deviations R^2 of a set of N data points $(x_i,\ y_i),\ i = 1, N$ from a function $f(x, a_1, a_2, ..., a_n)$.

$$R^2 = \sum_{i=1}^{N} [y_i - f(x_i, a, b, ..)]^2 \tag{4.1}$$

Note that this procedure does not minimize the actual deviations from the line (which would be measured perpendicular to the given function). In addition, although the unsquared sum of distances might seem a more appropriate quantity to minimize, use of the absolute value results in discontinuous derivatives which cannot be treated analytically. The square deviations from each point are therefore summed, and the resulting residual is then minimized to find the best fit line. This procedure results in outlying points being given disproportionately large weighting.

The condition for R^2 to be a minimum, ie, the best fit is that

$$\frac{\partial R^2}{\partial a_i} = 0 \tag{4.2}$$

4.1.2 Fit to a Straight Line

Consider fitting N data points to straight line:

$$y_i = mx_i + c \tag{4.3}$$

$$\begin{aligned} R^2 &= \sum_{i=1}^{N} [y_i - mx_i - c]^2 \\ &= \sum_{i=1}^{N} \left[y_i^2 + m^2 x_i^2 + c^2 - 2mx_i y_i - 2cy_i + 2mcx_i\right] \\ &= N\left[\langle y^2\rangle + m^2\langle x^2\rangle + c^2 - 2m\langle xy\rangle - 2c\langle y\rangle + 2mc\langle x\rangle\right] \end{aligned} \tag{4.4}$$

$$\begin{aligned} \frac{\partial R^2}{\partial c} &= 0 = 2c - 2\langle y\rangle + 2m\langle x\rangle \\ c &= \langle y\rangle - m\langle x\rangle \end{aligned} \tag{4.5}$$

$$\begin{aligned} \frac{\partial R^2}{\partial m} &= 0 = 2m\langle x^2\rangle - 2\langle xy\rangle - 2c\langle x\rangle \\ m &= \frac{\langle xy\rangle - \langle x\rangle\langle y\rangle}{\langle x^2\rangle - \langle x\rangle^2} \end{aligned} \tag{4.6}$$

Equations 4.6, 4.5 may be used to find m and c for the best fit of the data to a straight line.

4.1.3 How to ....

- As is evident you have to find the following averages

$$\langle x\rangle,\ \langle x^2\rangle,\ \langle y\rangle,\ \langle xy\rangle$$

- You have already seen a simple code to calculate averages. So, you are ready to write the complete code!
- It is better to have the data points $(x_i,\ y_i)$ in a file and read one data point at a time.

4.1.4 Suggested Exercises

1. Fit data points to the function $f(x) = ax^2 + b$.
2. Fit data points to the function $f(x) = ax^3 + b$.
3. Fit data points to the function $f(x) = ae^{\lambda x}$.

Hint: Use appropriate change of variable to obtain a relation of the form $f(z) = mz + c$.

4.2 Numerical integration

The formula used for Simpson's One-third rule

$$\int_a^b ydx \ = \ \frac{h}{3}\left[y_0 + 4\left(y_1 + y_3 + ... + y_{n-1}\right) + 2\left(y_2 + y_4 + ... + y_{n-2}\right) + y_n\right] \tag{4.7}$$

where $y_i = f(x_i)$ and $a = x_0 < x_1 < x_2 ... < x_n = b$ and $n = 2m$ is an even number.

4.2.1 How to ....

- Find *two* series sums, one for the *odd* terms and the other for the *even* terms.
- Note the two series have different number of terms.
- In the *first attempt* use *two* different loops for the two different series.
- Next try to use a *single* loop to sum the two different series. You may rewrite the formula as

$$\int_a^b ydx = \frac{h}{3}\left[y_0 + 4\left(y_1 + y_3 + ... + y_{n-1}\right) + 2\left(y_2 + y_4 + ... + y_{n-2} + y_n\right) - y_n\right] \tag{4.8}$$

4.2.2 Suggested Exercises

1. Do the integration for a simple function, evaluate the integration numerically for different values of n, compare with the analytical result and plot the error versus n.

2. Use the **Trapezoidal Rule** for the same integration.

$$\int_a^b ydx = \frac{h}{2}\left[y_0 + 2\left(y_1 + y_2 + ... + y_{n-1}\right) + y_n\right] \tag{4.9}$$

   Compare the error in Trapezoidal Rule method with the error in Simpson's One-third rule as a function of n.

4.3 Solution of Linear Equations

Consider a system of linear equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n &= b_1 && (4.10)\\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n &= b_2 && (4.11)\\ &..... && (4.12)\\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n &= b_n && (4.13) \end{aligned}$$

Make sure that the equations are independent.

This may be written as a matrix equqtion $AX = B$ where $A,\ X,\ B$ are the appropriate matrices. We start with a set of guess values of the solutions $\{x_{i,0}\}$.

Then we can write the next iteration for $\{x_i\}$ as

$$x_{1,1} = \frac{1}{a_{11}}\left[b_1 - a_{12}x_{2,0}.... - a_{1n}x_{n,0}\right] = \frac{1}{a_{11}}\left[b_1 - \sum_{i=2}^{n} a_{1i}x_{i,0}\right] \quad (4.14)$$

Next we use $x_{1,1}$ for finding $x_{2,1}$.

$$\begin{aligned} x_{2,1} &= \frac{1}{a_{22}}\left[b_2 - a_{21}x_{1,1} - a_{23}x_{3,0}.... - a_{2n}x_{n,0}\right]\\ &= \frac{1}{a_{22}}\left[b_1 - a_{21}x_{1,1} - \sum_{i=3}^{n} a_{2i}x_{i,0}\right] && (4.15) \end{aligned}$$

In general we can write for 1^{st} iteration of x_k $(x_{k,1})$

$$x_{k,1} = \frac{1}{a_{kk}}\left[b_k - \sum_{i=1}^{k-1} a_{ki}x_{i,1} - \sum_{i=k+1}^{n} a_{ki}x_{i,0}\right] \quad (4.16)$$

and for the r^{th} iteration of x_k $(x_{k,r})$

$$x_{k,r} = \frac{1}{a_{kk}}\left[b_k - \sum_{i=1}^{k-1} a_{ki}x_{i,r} - \sum_{i=k+1}^{n} a_{ki}x_{i,r-1}\right] \quad (4.17)$$

We can take the guess values $x_{i,0} = 0$ and stop the iteration when the diffference between two iterations become smaller than a certain number (ϵ).

$$\sum_{i=1}^{n} \left(x_{i,r} - x_{i,r-1}\right)^2 \leq \epsilon \quad (4.18)$$

4.3.1 How to?

- declare the arrays A (two-dimensional), B (one-dimensional), X (two-dimensional)
- read the elements of A, B.
- set the guess values $x_{i,0}$
- evaluate the terms $x_{k,r}$ - loop inside loop inside loop

4.4 Interpolation

We sometimes know the value of a function $f(x)$ or measurements at a set of points x_0, x_1, x_2, ... x_n (say, with $x_0 < x_1 < x_2 < .... < x_n$), but we don't have an analytic expression for f(x) that lets us calculate its value at an arbitrary point. For example, the $f(x_i)$'s might result from some physical measurement or from long numerical calculation that cannot be cast into a simple functional form. Often the x_i's are equally spaced, but not necessarily.

The task now is to estimate $f(x)$ for arbitrary x by, in some sense, drawing a smooth curve through (and perhaps beyond) the $\{x_i\}$. If the desired $x \in \{x_0, x_n\}$ the problem is called *interpolation*; if $x \notin \{x_0, x_n\}$, it is called *extrapolation*, which is considerably more hazardous (as stock-market analysts or economists can attest).

We assume that the given data represent some underlying function which is not analytically derivable, but has a certain degree of smoothness for the function interpolated, but within this framework of presumption, deviations from smoothness can be detected. Conceptually, the interpolation proceeds in two stages:

1. Fit an interpolating function to the data points provided.
2. Evaluate that interpolating function at the target point x.

The number of points (minus one) used in an interpolation scheme is called the order of the interpolation. Increasing the order does not necessarily increase the accuracy, especially in polynomial interpolation. If the added points are distant from the point of interest x, the resulting higher-order polynomial, with its additional constrained points, tends to oscillate wildly between the tabulated values. This oscillation may have no relation at all to the behavior of the 'true' function. (See **Numerical Recipes in C**)

4.5 Newton's Interpolation: Data at Regular Intervals

In the following we shall deal with data at regular intervals: we are given a set of $n+1$ data points, ($x_i = x_0 + ih$)

$$(x_0, y_0),\ (x_1, y_1),\ (x_2, y_2), ...\ (x_n, y_n)$$

. We have to find $y_n(x)$, a polynomial of degree n which approximately represents the underlying functional form such that y and $y_n(x)$ agree at the tabulated points.

First let's construct the *forward difference table*: the *forward differences* are defined as follows:

$$\begin{aligned}
\Delta^1 y_r &= y_{r+1} - y_r && (4.19)\\
\Delta^2 y_r &= \Delta^1 y_{r+1} - \Delta^1 y_r = y_{r+2} - 2y_{r+1} + y_r && (4.20)\\
\Delta^3 y_r &= \Delta^2 y_{r+1} - \Delta^2 y_r = y_{r+3} - 3y_{r+2} + 3y_{r+1} - y_r && (4.21)\\
..... &= ..... \\
..... &= ..... \\
\Delta^p y_r &= \Delta^{p-1} y_{r+1} - \Delta^{p-1} y_r = y_{r+p} - {}^pC_1.y_{r+p-1} + {}^pC_2.y_{r+p-2} .... (-1)^p y_r && (4.22)
\end{aligned}$$

is the general form for the difference. The difference table for *six* data points will look as follows:

x_0	y_0	$\Delta^1 y_0$	$\Delta^2 y_0$	$\Delta^3 y_0$	$\Delta^4 y_0$	$\Delta^5 y_0$
x_1	y_1	$\Delta^1 y_1$	$\Delta^2 y_1$	$\Delta^3 y_1$	$\Delta^4 y_1$	
x_2	y_2	$\Delta^1 y_2$	$\Delta^2 y_2$	$\Delta^3 y_2$		
x_3	y_3	$\Delta^1 y_3$	$\Delta^2 y_3$			
x_4	y_4	$\Delta^1 y_4$				
x_5	y_5					

Let us write $y_n(x)$ in the following way:

$$y_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + ... + a_n(x - x_0)(x - x_1)...(x - x_n) \quad (4.23)$$

Since the polynomial has to match with the data set at (x_0, y_0), (x_1, y_1), ... (x_n, y_n) it may be shown that

$$a_0 = y_0 \quad (4.24)$$

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta^1 y_0}{h^1.1!} \quad (4.25)$$

$$a_2 = \frac{\Delta^2 y_0}{h^2.2!} \quad (4.26)$$

$$a_n = \frac{\Delta^n y_0}{h^n.n!} \quad (4.27)$$

For an arbitrary $x = x_0 + p.h$ we may write equation 4.23 as

$$y_n(x) = y_0 + \frac{p}{1!}\Delta^1 y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0 + ... + \frac{p(p-1)(p-2)...(p-n+1)}{n!}\Delta^n y_0 \quad (4.28)$$

4.5.0.1 How to ....

1. Read the given data points, preferably from an input file.
2. Construct the difference table: the code is incredibly simple and easy to implement, use the loops carefully
3. Calculate the approximate y for the given x: use the recursion relation for the successive terms for making the code compact and the computing faster.

4.5.1 Lagrange' Method: Polynomial Interpolation

Newton's method is applicable where data at equal intervals is provided. Polynomial Interpolation using Lagrange's formula is applicable also for data at unequal intervals:

$$P(x) = \frac{(x - x_1)(x - x_2)...(x - x_n)}{(x_0 - x_1)(x_0 - x_2)...(x_0 - x_n)}.y_0 + \frac{(x - x_0)(x - x_2)...(x - x_n)}{(x_1 - x_0)(x_1 - x_2)...(x_1 - x_n)}.y_1 + ... + \frac{(x - x_0)(x - x_2)...(x - x_n)}{(x_n - x_0)(x_n - x_1)...(x_n - x_{n-1})}.y_n \quad (4.29)$$

Here each term is a polynomial of degree n-1. All terms are designed to be zero except the ith which is equal to y_i.

4.6 Finding Roots of an Equation

4.6.1 Newton Raphson Method: Real Roots

Let us consider an equation of $f(x) = 0$ with real roots (X_i). We use a guess value for the solution (x_0) and iteratively arrive at one of the solutions. First we draw a tangent at the guess value and find where it intersects the x axis (x_1). This is used as the new guess value. The process is iterated till $|x_{i+1} - x_i| \leq \delta$ where δ is a given small number depending on the desired accuracy of the solution.

Consider a function $f(x)$ and its derivative is $f^{'}(x)$. The tangent at x_0 gives:

$$\begin{aligned} f(x_0) &= f^{'}(x_0).x_0 + c \\ c &= f(x_0) - f^{'}(x_0).x_0 \end{aligned} \tag{4.30}$$

Now suppose the tangent at x_0 intersects the *x-axis* at x_1. Hence, using equation 4.30 we get

$$\begin{aligned} 0 &= f^{'}(x_0).x_1 + c = f^{'}(x_0).x_1 + f(x_0) - f^{'}(x_0).x_0 \\ &= f^{'}(x_0)(x_1 - x_0) + f(x_0) \\ \Rightarrow x_1 &= x_0 - \frac{f(x_0)}{f^{'}(x_0)} = x_0 + \Delta x \end{aligned} \tag{4.31}$$

We can write in general

$$x_{i+1} = x_i - \frac{f(x_i)}{f^{'}(x_i)} \tag{4.32}$$

This method depends on the choice of x_0 - if $f^{'}(x_0) = 0$ then the method fails. Also, if there are multiple roots, the guess value of x determines which root will be found.

4.6.2 How to ....

Plot the function using a graphical application like gnuplot and find out the approximate values of X_i. Then use this method to find the X_i's more accurately. The iteration is easy to implement: use `do - while` with the condition for accuarcy, ie, $|x_{i+1} - x_i| \leq \delta$.

4.6.3 Newton Raphson Method: Complex Roots

Consider $f(z) = 0$ where $z = x+iy$.

$$\begin{aligned} f(z) = g(x,y) + ih(x,y) &= 0 \\ \Rightarrow g(x,y) = h(x,y) &= 0 \end{aligned}$$

Now proceeding similarly with a guess value $z_0 = x_0 + iy_0$ we may write

$$\begin{aligned} g(x_0 + \Delta x, y_0 + \Delta y) = g(x_0, y_0) + \frac{\partial g}{\partial x}|_{x_0,y_0}.\Delta x + \frac{\partial g}{\partial y}|_{x_0,y_0}.\Delta y &= 0 \\ h(x_0 + \Delta x, y_0 + \Delta y) = h(x_0, y_0) + \frac{\partial h}{\partial x}|_{x_0,y_0}.\Delta x + \frac{\partial h}{\partial y}|_{x_0,y_0}.\Delta y &= 0 \end{aligned}$$

Which may be written more compactly as

$$g_0 + g_x.\Delta x + g_y.\Delta y = 0 \quad (4.33)$$
$$h_0 + h_x.\Delta x + h_y.\Delta y = 0 \quad (4.34)$$

where,

$$g_0 = g(x_0, y_0), \quad g_x = \frac{\partial g}{\partial x}|_{x_0,y_0}, \quad g_y = \frac{\partial g}{\partial y}|_{x_0,y_0}, \quad h_0 = h(x_0, y_0), \quad h_x = \frac{\partial h}{\partial x}|_{x_0,y_0}, \quad h_y = \frac{\partial h}{\partial y}|_{x_0,y_0}$$

Solving equations 4.33, 4.34 we get

$$\Delta x = \frac{g_y h_0 - g_0 h_y}{g_x h_y - g_y h_x} \quad (4.35)$$
$$\Delta y = \frac{g_0 h_1 - g_1 h_0}{g_x h_y - g_y h_x} \quad (4.36)$$
$$x_1 = x_0 + \Delta x \quad (4.37)$$
$$y_1 = y_0 + \Delta y \quad (4.38)$$

This may be iterated till $|\Delta x| \leq \delta, \quad |\Delta y| \leq \delta$.

4.6.4 Bi-section Method: Real Roots

In this method we need two guess values x_1 and x_2 such that $f(x_1) > 0$ and $f(x_2) < 0$. So the interval $\{x_1, x_2\}$ contain at least one root X_i of the equation $f(x) = 0$. We approach X_i iteratively.

1. take $x' = (x_1 + x_2)/2$ and compute $f(x')$
2. if $f(x') < 0$, replace x_1 by x'
3. if $f(x') > 0$, replace x_2 by x'
4. repeat till $|f(x')| > \delta$ where δ controls how close x' is to X_i

4.6.5 How to ....

- Select two guess values x_1 and x_2: use a `do, while` loop till $f(x_1) * f(x_2) < 0.0$ condition is met
- use another `do, while` loop to find X_i

4.7 Solution of Ordinary Differential Equations: Runge-Kutta Methods

4.7.1 First Order Differential Equation:

Given a first order differential equation and the initial condition,

$$\frac{dy}{dx} = f(x, y), \; y(x_0) = y_0$$

we may numerically obtain the values of $y(x)$ for $x \neq x_0$.

Second Order Runge-Kutta Formula

$$y(x_1) = y_0 + \frac{h}{2}\left[f(x_0, y_0) + f(x_0 + h, y_0 + hf(x_0, y_0))\right] \tag{4.39}$$

This may be alternately written as

$$y(x_1) = y_0 + \frac{1}{2}(k_1 + k_2) \tag{4.40}$$
$$k_1 = h.f(x_0, y_0) \tag{4.41}$$
$$k_2 = h.f(x_0 + h, y_0 + k_1) \tag{4.42}$$

Do you see any connection with numerical integration in this formula, or, the one below?

Fourth Order Runge-Kutta Formula

$$k_1 = f(x_0, y_0) \tag{4.43}$$
$$k_2 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right) \tag{4.44}$$
$$k_3 = f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}\right) \tag{4.45}$$
$$k_4 = f(x_0 + h, y_0 + k_3) \tag{4.46}$$
$$y(x_1) = y_0 + \frac{h}{6}(k_1 + k_2 + k_3 + k_4) \tag{4.47}$$

4.7.2 How to ....

It is better to subdivide the interval $x - x_0$ into a large number of equal sub-intervals and make use of the formula iteratively.

4.7.3 Suggested exercises

Use the Runge-Kutta Methods (4th order formula) to numerically find the solutions for the following:

$$\frac{dy}{dx} = 1 + y^2,\ y(0) = 0$$
$$\frac{dy}{dx} = y - x,\ y(0) = 2$$
$$10\frac{dy}{dx} = x^2 + y^2,\ y(0) = 0$$
$$\frac{dy}{dx} = \frac{y - x}{y + x}, y(0) = 1$$

4.7.4 Coupled First Order Differential Equations:

Given two first order differential equations and their initial conditions,

$$\frac{dx}{dt} = f(x, y, t), \; x(t_0) = x_0, \frac{dy}{dt} = g(x, y, t), \; y(t_0) = y_0$$

we may numerically obtain the values of $x(t)$, $y(t)$ for $t \neq t_0$.

Fourth Order Runge-Kutta Formula

$$\begin{aligned}
k_1 &= f(x_0, y_0, t_0) && (4.48)\\
m_1 &= g(x_0, y_0, t_0) && (4.49)\\
k_2 &= f\left(x_0 + \frac{k_1}{2}, y_0 + \frac{m_1}{2}, t_0 + \frac{h}{2}\right) && (4.50)\\
m_2 &= g\left(x_0 + \frac{k_1}{2}, y_0 + \frac{m_1}{2}, t_0 + \frac{h}{2}\right) && (4.51)\\
k_3 &= f\left(x_0 + \frac{k_2}{2}, y_0 + \frac{m_2}{2}, t_0 + \frac{h}{2}\right) && (4.52)\\
m_3 &= g\left(x_0 + \frac{k_2}{2}, y_0 + \frac{m_2}{2}, t_0 + \frac{h}{2}\right) && (4.53)\\
k_4 &= f(x_0 + k_3, y_0 + m_3, t_0 + h) && (4.54)\\
m_4 &= g(x_0 + k_3, y_0 + m_3, t_0 + h) && (4.55)\\
x(t_0 + h) &= x_0 + \frac{h}{6}(k_1 + k_2 + k_3 + k_4) && (4.56)\\
y(t_0 + h) &= y_0 + \frac{h}{6}(m_1 + m_2 + m_3 + m_4) && (4.57)
\end{aligned}$$

4.8 Use of Random numbers

4.8.1 What is ..

At the microscopic level, at the quantum level physical processes happen randomly - outcome of a coin toss, decay of a radioactive nucleus, emission of a photon by an atom, measured value of energy of a particle, etc. Although, at the macroscopic level they follow statistical rules according to the physical process governing such actions. We may associate *random numbers* with the outcome of such processes. As is true for any statistical process the nature of the random number may only be understood only when we make large number of observations.

The distribution of random numbers may be uniform, Gaussian, Poisson, etc. Random numbers are heavily used in computation and simulation of physical phenomena to predict the outcome of physical prcesses particulalry where it is difficult, if not impossible, to use analytical methods.

4.8.2 How to ..

True random numbers, in principle can be obtained from physical processes which are random in nature. This is not possible with computers, which use well-defined processes. Computers use

sophisticate mathematical algorithms to generate (very) long sequence of numbers which have (almost) all the proerties of true random numbers. Such numbers are called *pseudo-random numbers*. Most programming languages have ready-to-use pseudo-random number generators which may be called in a program.

4.8.3 Example

Following is a simple program to test the uniformity of such a generator which generates random numbers uniformly distributed between `0, 1`.

- *Random numbers between 0,1 have been generated*
- *numbers falling into each of the hundred equal bins 0.00-0.01, 0.01-0.02, ... 0.99-1.00 have been counted :* N_i *is the number of times the generated random number falls in the* ith *bin*
- *fractions* $f_i = N_i/\Sigma_i N_i$ *are obtained and plotted againts* i

```
/* ------------------------------------------------------------------------ */
/* ----------------------- random number generation ----------------------- */
/* ------------------------------------------------------------------------ */
#include<stdio.h>

main()
{
  float x, y, nbin[100], fraction[100];
  int i, j, nterm;
  FILE *outfile;

  outfile  = fopen("random-histo.out", "wt");

  printf("How many terms do you want    ");
  scanf("%d", &nterm);

  for(i=0; i<100; ++i) nbin[i] = 0.0;

  for(i=0; i<nterm; ++i)
    {
      /* ------- generate random numbers and put them in bins -------
 ------- rand()%1000 generates integers between 0 1000 ------ */
      x = rand()%1000;
      y = x/1000.0;

      for(j=0; j<100; ++j) if(y >= j*.01 && y<(j+1)*.01) ++nbin[j];
    }

  /* ------------------ write the output to a file ------------------ */
  fprintf(outfile, "#\nbin    fraction\n");
  for(i=0; i<100; ++i)
```

```
    {
      fraction[i] = nbin[i]/nterm;
      fprintf(outfile, "%d\t%f\n", i, fraction[i]);
    }

  return 0;
}
```

We run the program to generate $10^3,\ 10^4,\ 10^5,\ 10^6, 10^3$ events. The histograms made from these runs have been plotted in the figure below. See that the fluctuations in f_i are reduced as the number of generated random numbers increases.

The output files for the runs are stored in files `random-1K.txt, random-10K.txt, random-100K.txt, random-1000K.txt` and plotted with the following set of commands in `gnuplot` to create `random.jpg`.

```
set term jpeg landscape monochrome
set out ‘‘random.jpg’’
set xrange [0.002:100.0]
set yrange [0.002:0.03]
plot "random-1K.txt" u 1:2 w lp,\
     "random-10K.txt" u 1:2 w lp,\
     "random-100K.txt" u 1:2 w lp,\
     "random-1000K.txt" u 1:2 w lp
```

Alternately, you can write the commands in a file, say, `random-plot.gnu` and execute the command

```
$ gnuplot random-plot.gnu
```

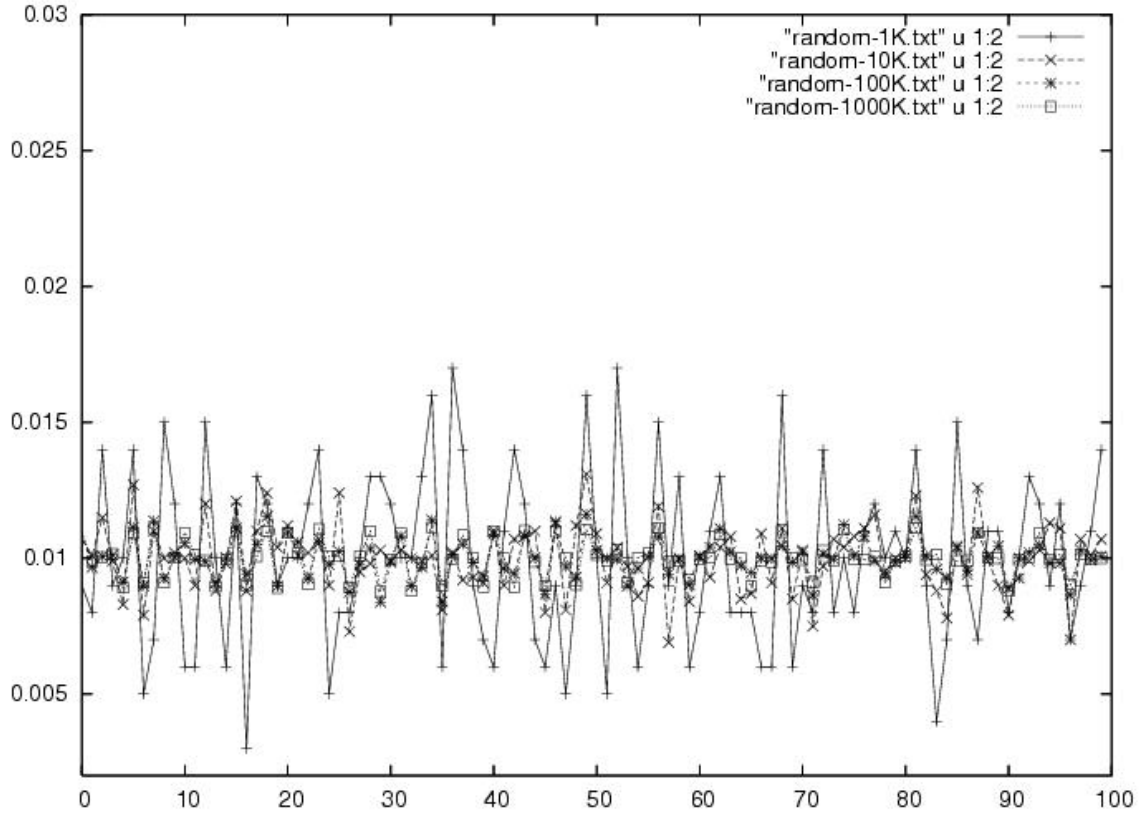


Figure 4.3: Test of uniformity of a flat/uniform random number generator.

4.9 Solutions to problems of Numerical Analysis

4.9.1 Least Square fitting

The following code will do Least Square Fit to a straight line. The output will be written on a file `LeastSquare.out`. It will also create `LeastSquare.gnu` which is optional. The last few lines are optional. We shall use *system* call to invoke `gnuplot` and plot the data written to the file `LeastSquare.out`.

`gnuplot` is a popular plotting package available on most Linux platforms. Simple instructions of using `gnuplot` has been described later in the book.

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>

/* ----------- this is a least square fit to a straight line ---------- */
/* ----------- max. number of data points to be fitted = 10 ----------- */
main()
{
```

```
int npoint, ip;
FILE *infile, *outfile, *plotfile;
char input[80];
float x[10], y[10], xavg = 0.0, yavg = 0.0, xyavg = 0.0, xxavg = 0.0;
float slope, constant;

/* open the input & output files ------------------------------ */
infile   = fopen("LeastSquare.inp", "rt");
outfile  = fopen("LeastSquare.out", "wt");
plotfile = fopen("LeastSquare.gnu", "wt");

/* read the number of points to be fitted from a file ---------- */
fgets(input,80,infile);
sscanf(input,"%d", &npoint);

printf("\nNumber of points to be fitted ( max. = 10) \t%d\n", npoint);

/* read the data points and form the different averages -------- */
for(ip = 0; ip < npoint; ++ip)
  {
    fgets(input,80,infile);
    sscanf(input,"%f%f", &x[ip], &y[ip]);
    xavg   = xavg  + x[ip];
    yavg   = yavg  + y[ip];
    xyavg  = xyavg + x[ip]*y[ip];
    xxavg  = xxavg + x[ip]*x[ip];
  }

xavg   = xavg/npoint;
yavg   = yavg/npoint;
xyavg  = xyavg/npoint;
xxavg  = xxavg/npoint;

printf("The averages ---------------------------- \n");
printf("x = %f\t y = %f\t xy = %f\t x^2 = %f\n", xavg, yavg, xyavg, xxavg);
fprintf(outfile, "#The averages ---------------------------- \n");
fprintf(outfile, "#x = %f\t y = %f\t xy = %f\t x^2 = %f\n", xavg, yavg, xyavg, xxavg);

slope    = (xyavg - xavg*yavg)/(xxavg - xavg*xavg);
constant = yavg - slope*xavg;

/* print the results ------------------------------------------ */
printf("\nThe best straight line has slope = %f\n", slope);
printf  ("                          constant = %f\n", constant);
fprintf(outfile, "\n#The best straight line has slope = %f\n", slope);
fprintf  (outfile, "#                          constant = %f\n", constant);

printf("\nx               y               fitted y ");
printf("\n----------------------------------------\n");
```

```
  fprintf(outfile, "\n#x               y               fitted y ");
  fprintf(outfile, "\n#-----------------------------------------\n");
  for(ip = 0; ip < npoint; ++ip)
    {
      /* ---------------- print the data points and the fitted values */
      printf("%f\t%f\t%f\n", x[ip], y[ip], slope*x[ip]+constant);
      fprintf(outfile, "%f\t%f\t%f\n", x[ip], y[ip], slope*x[ip]+constant);
    }

  fclose(outfile);
  /* ------------------------- optional code ------------------------ */
  /* ------------------ for plotting using gnuplot ------------------ */

  /* ------------- Prepare the file of gnuplot commands ------------- */
  fprintf(plotfile, "set xrange [%g:%g] \n", x[0]-1.0, x[npoint-1]+1.0);
  fprintf(plotfile, "plot \"LeastSquare.out\" u 1:2 w p, \"LeastSquare.out\" u 1:3 w l\n
  fprintf(plotfile, "pause -1\n");
  fflush(plotfile);
  fclose(plotfile);

  /* ----------------------- invoke gnuplot ------------------------- */
  system("xterm -e gnuplot LeastSquare.gnu &");

  return 0;

}
```

The input file used `LeastSquare.inp` and the two output file `LeastSquare.out`, `LeastSquare.gnu` are given below.

——————— `LeastSquare.inp` ———————

```
20
0.5 1.1
1.0 1.3
1.5 2.0
2.0 2.7
2.5 3.4
3.0 3.3
3.5 4.4
4.0 4.8
4.5 5.1
5.0 5.5
5.5 6.0
6.0 6.7
6.5 7.0
7.0 7.8
7.5 8.1
```

```
8.0 8.3
9.0 9.6
10.0 10.0
10.5 11.1
11.0 11.5
12.0 13.0
```

LeastSquare.out

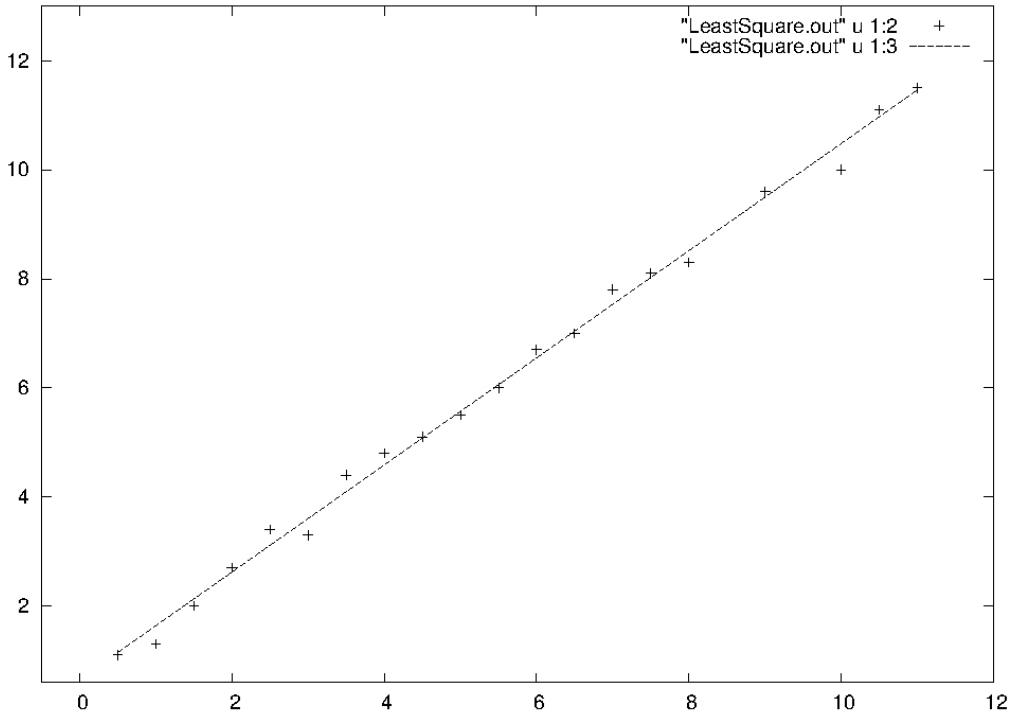
```
#The averages ----------------------------
#x = 5.425000  y = 5.985000  xy = 42.180000  x^2 = 39.312500

#The best straight line has slope = 0.982746
#                         constant = 0.653600

#x               y               fitted y
#-----------------------------------------
0.500000 1.100000 1.144973
1.000000 1.300000 1.636346
1.500000 2.000000 2.127720
2.000000 2.700000 2.619093
2.500000 3.400000 3.110466
3.000000 3.300000 3.601839
3.500000 4.400000 4.093212
4.000000 4.800000 4.584586
4.500000 5.100000 5.075959
5.000000 5.500000 5.567332
5.500000 6.000000 6.058705
6.000000 6.700000 6.550079
6.500000 7.000000 7.041452
7.000000 7.800000 7.532825
7.500000 8.100000 8.024198
8.000000 8.300000 8.515572
9.000000 9.600000 9.498318
10.000000 10.000000 10.481065
10.500000 11.100000 10.972438
11.000000 11.500000 11.463811
```

LeastSquare.gnu

```
set xrange [-0.5:12]
set yrange [0.6:13]
plot "LeastSquare.out" u 1:2 w p, "LeastSquare.out" u 1:3 w l
pause -1
```



4.9.2 Simpson's One-third Rule

The following code may be used for numerical integration using Simpson's One-third rule. Note that the function $y = f(x)$ (here $f(x) = 3x^2 + 4x + 5$) has been defined in the beginning. This helps avoid writing the function explicitly at different places in the code.

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>

#define f(x) 3*x*x+4*x+5
/* #define f(x) sin(x) */
/* #define f(x) 1/x; */
main()
{
int i, m;
float a, b,  xodd, xeven, sumodd=0.0, sumeven=0.0, sum = 0.0;
float h, yodd, yeven, ya, yb;

printf("\n give me lower limit a, upper limit b\n");
scanf("%f%f", &a, &b);
printf("\n---------- Number of divisions  n = 2m ----------------");
printf("\nGive m        ");
scanf("%d", &m);
```

```
h=(b-a)/(2*m);
ya = f(a);
yb = f(b);

// --------------------- Sum the odd terms ----------------------
for(i=1;i<=2*m;i=i+2)
  {
    xodd = a+i*h;
    yodd=f(xodd);
    sumodd = sumodd + yodd;
  }

// --------------------- Sum the even terms ----------------------
for(i=2;i<2*m;i=i+2)
  {
    xeven = a+i*h;
    yeven = f(xeven);
    sumeven = sumeven + yeven;
  }

sum = (h/3)*(ya + 4*sumodd + 2*sumeven + yb);
printf("\nThe result is \t%f\n", sum);

return 0;
}
```

4.9.3 Solution of System of Linear Equations: Gauss-Seidal Method

The following code may be used for implementing Gauss-Seidal method for solving system of linear equations:

```
#include <iostream>
#include <fstream>
using namespace std;

int main(void)
{
float a[10][10], b[10], x[20][20], sum, accuracy;
int i, j, k, r, n;
ifstream in("Gauss1.inp");
ofstream out("Gauss.out");

cout << "\nThis programme solves equations of the form AX = B\n";
// read the dimension of the matrix A
cout << "\nReading the dimension of matrix A";
in >> n;
```

```
// read the elements of  matrix A from file

cout << "\nReading the elements of matrix A\n";
for(i=0;i<n;i=i+1)
{
for(j=0;j<n;j=j+1)
{
in>>a[i][j];
cout << a[i][j] << "\t";
}
cout << "\n";
}

cout << "\nReading the elements of matrix B\n";
for(i=0;i<n;i=i+1)
{
in>>b[i];
cout << b[i] << "\t";
}
cout << "\n";

cout<<"Reading the guess values of matrix X\n";
for(i=0;i<n;i=i+1)
{
in>>x[i][0];
cout << x[i][0] << "\t";
}
cout << "\n";

// run the iterative loop

for(r=1;r<20;r++)
{
cout << r << "  -------------------------\n";
// for term i
for(i=0; i<n; i++)
{
sum = b[i];
// cout << "sum1 " << sum << "\n";
for(k=0; k<i; k++)
{
sum = sum - a[i][k]*x[k][r];
}
// cout << "sum2 " << sum << "\n";
for(k=i+1; k<n; k++)
{
sum = sum - a[i][k]*x[k][r-1];
}
// cout << "sum3 " << sum << "\n";
```

```
x[i][r] = sum/a[i][i];
cout << x[i][r] << "\n";
}
}

return(0);
}
```

4.9.4 Newton's Formula for Forward interpolation

The following code may be used for implementing Newton's Formula for Forward interpolation

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>

int main(void)
{
  int ndata, i, j;
  float x[20], y[20][20], diff;
  float xnew, ynew, p, term, change;
  FILE *infile, *outfile;
  char input[80];

  infile  = fopen("interpolation.inp", "rt");
  outfile = fopen("interpolation.out", "wt");

  /*-------------------- Set out the basics --------------------*/
  printf("\n-------- This is a simple C program for interpolation -----------");
  printf("\n-------- The data points should be at equal intervals -----------");
  printf("\n------------- This can handle only 20 data points ---------------");

  /*-------------------- read in the input ---------------------*/
  fgets(input,80,infile);
  sscanf(input,"%d", &ndata);

  for(i=0; i<ndata; ++i)
    {
      fgets(input,80,infile);
      sscanf(input, "%f%f", &x[i], &y[0][i]);
    }
  /*------------ Now construct the difference table ------------*/

  for(i=1; i<ndata; ++i)
    {
```

```
      for(j=0; j<ndata-i; ++j) y[i][j] = y[i-1][j+1] - y[i-1][j];
    }

  /*------------ Print the data and difference table -----------*/

  fprintf(outfile, "-------------- Forward Difference table ----------------\n");
  printf("\n-------------- Forward Difference table ----------------\n");

  for(i=0; i<ndata; ++i)
    {
      fprintf(outfile, "%f\t", x[i]);
      printf("%f\t", x[i]);
    }

  for(i=0; i<ndata; ++i)
    {
      fprintf(outfile, "\n");
      printf("\n");

      for(j=0; j<ndata-i; ++j)
{
  fprintf(outfile, "%f\t", y[i][j]);
  printf("%f\t", y[i][j]);
}
    }
  fprintf(outfile, "\n");

  /*---------------- read in the new value of x --------------- */
  printf("\n\nGive new value of x    ");
  scanf("%f", &xnew);

  /* use Newton's forward difference interpolation formula */
  diff  = x[1] - x[0];
  p     = (xnew-x[0])/diff;
  fprintf(outfile, "\nNew x = %f\t p = %f\n", xnew, p);

  term = 1.0;
  ynew  = y[0][0];

  for (i=0; i<ndata-1; ++i)
    {
      term = term*(p-i)/(i+1);
      change = term*y[i+1][0];
      ynew  = ynew + change;
      printf("\nTerm  %d\t change = %f  ynew = \t%f", i, change, ynew);
    }

  fprintf(outfile, "\nxnew = %f\tynew = %f\n", xnew, ynew);
  printf ("\n\nxnew = %f\tynew = %f\n", xnew, ynew);
```

```
  return 1;
}
```

For the following input data

```
4
0.0 1.0
1.0 0.0
2.0 1.0
3.0 10.0
```

the output file is given below

```
-------------- Input data ----------------
0.000000 1.000000 2.000000 3.000000
1.000000 0.000000 1.000000 10.000000

-------------- Forward Difference table ----------------
-1.000000 1.000000 9.000000
2.000000 8.000000
6.000000

New x = 1.300000  p = 1.300000

xnew = 1.300000 ynew = -0.183000
```

4.9.5 Real Solutions of an Equation: Bisection Method

```
/* ------------------------------------------------------------------- */
/* ----------------- Finding the real roots of equations ------------- */
/* ------------------------------------------------------------------- */

#include<stdio.h>
#include<math.h>

float func(float x)
{
  return x*x-7.0*x+6.0;
}

main()
{
  float delta, xguess1, xguess2, x, x1, x2;
  int i;

  /* get a number for the accuracy of the solution ------------------- */
```

```
  printf("How close a solution do you want? delta = ");
  scanf("%f", &delta);

  /* get the guess values ------------------------------------------- */
  do
    {

      printf("Give the guess values xguess1, xguess2\n");
      scanf("%f %f", &xguess1, &xguess2);
      printf("func(%f) = %f, func(%f) = %f\n", xguess1, func(xguess1), xguess2,
func(xguess2));

      /* check if you have accidentally got the roots ----------------- */
      if(func(xguess1) == 0.0)
printf("One root of the equation is %f\n", xguess1);
      if(func(xguess2) == 0.0)
printf("One root of the equation is %f\n", xguess2);

      if(func(xguess1)*func(xguess2) == 0.0) return 0;

      if(func(xguess1) < 0.0 && func(xguess2) > 0.0)
{
  x1 = xguess1;
  x2 = xguess2;
}
      if(func(xguess1) > 0.0 && func(xguess2) < 0.0)
{
  x1 = xguess2;
  x2 = xguess1;
}
      else
{
  printf("Try a new set of guess values\n");
}

    }

  while (func(x1)*func(x2) > 0.0);

  printf("\nAccepted guess values %f %f\n", x1, x2);

  /* iterate to get one root ---------------------------------------- */
  do
    {
      x = (x1+x2)/2.0;
      if(func(x) > 0.0) x2 = x;
      if(func(x) < 0.0) x1 = x;
      /*      printf("%f %f %f %f \n", x1, x2, x, func(x)); */
    }
```

```
  while( fabs(func(x)) > delta );

  printf("\nOne root between %f & %f is %f\n", xguess1, xguess2, x);

  return 0;
}
```

4.9.6 Runge-Kutta Methods

```
% ------------------------------------------------------------------------
/* ---------------------------------------------------------------------- */
/* ---- solution of differential equations: Runge-Kutta Methods ---- */
/* ---------------------------------------------------------------------- */
#include<math.h>
#include<string.h>
#include<stdlib.h>

float funct( float x, float y )
{
  return y-x;
}

float RungeKutta ( float x, float y, float h)
{
  float x1, y1;
  float p1, p2, p3, p4;

  p1 = h*funct(x, y);
  p2 = h*funct(x+0.5*h, y+0.5*p1);
  p3 = h*funct(x+0.5*h, y+0.5*p2);
  p4 = h*funct(x+h, y+p3);

  y1 = y+ (p1+ 2.0*(p2+p3) + p4)/6.0;
  return y1;
}

main()
{
  float x0, y0, xnew, ynew, xold, yold, h;
  int istep, nstep;

  printf("\n4th order Runge-Kutta method : \n",
         "-------------------------------------------------\n");
  printf("\n Give x0, y0\t");
  scanf("%f%f", &x0, &y0);
  printf("\nInitial values \t%f  %f\n", x0, y0);
  printf("\n\n Give xnew, intermediate steps nstep\t");
  scanf("%f%d", &xnew, &nstep);
```

```
  h  = (xnew - x0)/nstep;
  printf("step size %f\n", h);

  xold = x0;
  yold = y0;

  printf("\n             Successive progression              ");
  printf("\n-------------------------------------------------\n");
  /*------------ ynew in each step is stored in yold ------------*/
  for( istep = 1; istep <= nstep; ++istep)
    {
      ynew = RungeKutta(xold, yold, h);
      yold = ynew;
      xold = x0 + istep*h;
      printf("%d\t%f\t%f\n", istep, xold, ynew);
    }

  return 0;
}
```

Chapter 5

Introduction to Gnuplot

5.1 What is ...

Gnuplot is a portable command-line driven interactive data and function plotting utility for UNIX, IBM OS/2, MS Windows, DOS, Macintosh, VMS, Atari and many other platforms. The software is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally intended to allow scientists and students to visualize mathematical functions and data. *Gnuplot* has been supported and under development since 1986.

Gnuplot supports many types of plots in either 2D and 3D. It can draw using lines, points, boxes, contours, vector fields, surfaces, and various associated text. It also supports various specialized plot types.

5.2 Getting Started

5.2.1 Plotting Functions

The following is the transcript of a session with *Gnuplot*. You start with typing `gnulpot` at the shell prompt and proceed and quit as shown below:

```
$ gnuplot
gnuplot> plot sin(x)
gnuplot> q
$
```

Now you have started and let's proceed further. Watch the effect of every command with *Gnuplot*.

```
gnuplot> plot sin(x)
gnuplot> plot x**2
gnuplot> plot x**2 w l
gnuplot> plot sin(x) w p
gnuplot> plot x**2 w lp
gnuplot> q
```

This intutively gives you some idea how to plot functions with *Gnuplot*. You can use common, ready-to-use mathematical functions or, define your own function.

The general format for plotting a function is
`plot [xlow:xhigh] [ylow:yhigh] function1, function2, w options`
`xlow, xhigh` specify the range of `x` for which to plot the function
`ylow, yhigh` specify the range of `y`. `options` have values `l`, `p`, `lp`. Use the following commands and see their effects in changing the colour, point type and line width.

```
gnuplot> plot [0.0:6.5] sin(x), cos(x), 2*x
gnuplot> plot [-5.0:5.0] tanh(2*x), x
gnuplot> plot [0.0:6.5] sin(x)
gnuplot> plot [0.0:6.5] cos(x)
gnuplot> plot [0.0:6.5] cos(x)
gnuplot> plot [0.0:6.5] [-1:1] sin(x)/cos(x)
gnuplot> plot [0.0:6.5] [-5:5] sin(x)/cos(x) w lp
gnuplot> plot [0.0:10.0] sin(x)/x
gnuplot> plot [0.0:10.0] 2.0*x**2
gnuplot> q
```

Now watch for some applications of *colour* and *point type* with the following.

```
gnuplot> plot sin(x) w lp 3 4
gnuplot> plot sin(x) lw 2
gnuplot> plot sin(x) lw 3
gnuplot> plot sin(x) lw 3 5
gnuplot> plot sin(x) w p 3 4
```

Let's get deeper into *Gnuplot* and look for finer effects.
If you don't want the labels use `set nokey` as follows:

```
gnuplot> set nokey
gnuplot> plot [0.0:6.6] sin(x)
```

If you want a particular label then:

```
gnuplot> set key
gnuplot> plot [0.0:6.6] sin(x) title 'bhoot(x)'
```

You can set labels for *x-axis* and *y-axis*:

```
gnuplot> set xlabel '{theta}'
gnuplot> set ylabel 'Sin(theta)'
gnuplot> plot [0.0:6.6] sin(x) title 'sin(x)'
```

```
$ gnuplot
gnuplot> plot [0.0:6.6] sin(x)
```

```
gnuplot> plot [0.0:6.6] sin(x) title 'sin(x)'
gnuplot> plot [0.0:6.6] sin(x) title 'sin(x)', cos(x) title 'cos(x)'
gnuplot> plot [0.0:6.6] sin(x)
gnuplot> plot [0.0:6.6] sin(x) title 'sin(x)'
gnuplot> set xlabel 'theta'
gnuplot> set ylabel 'sin(theta)'
gnuplot> plot [0.0:6.6] sin(x) title 'sin(x)'
gnuplot> q
$
```

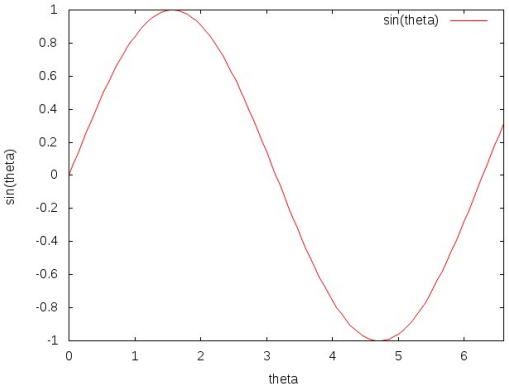
5.2.2 Creating a figure file

Now let us prepare a figure file (`sinx.ps`) to include in a LaTeX document.

```
gnuplot> set xlabel 'theta'
gnuplot> set ylabel 'sin(theta)'
gnuplot> set term jpeg landscape
gnuplot> set out 'sinx.jpg'
gnuplot> plot [0.0:6.6] sin(x) title 'sin(theta)'
```

This time you don't see anything happen as the output of the plot command has been stored in the *sinx.jpg* file. `set term post landscape` sets the terminal to *postscript* mode and the orientation is *landscape*. You could also use `portrait` and see the difference.

The following figure has been produced by the above set of commands.



Also use the following and see the difference:

```
gnuplot> set xlabel 'theta'
gnuplot> set ylabel 'sin(theta)'
gnuplot> set term post landscape color
gnuplot> set out 'sinx-2.ps'
gnuplot> plot [0.0:6.6] sin(x) title 'sin(theta)' w lp 3 2
```

5.2.3 Repeated Use of a Set of Commands: .gnu files

Now, instead of typing a set of commands many times in the same session or in every new session we may put a set of commands in a file and invoke it when needed:

```
$ gnuplot sinx.gnu
$ gv sinx.ps
```

The corresponding `sinx.gnu` file will look like:

```
set xlabel 'theta'
set ylabel 'Sin(theta)'
set term post
set out 'sinx.ps'
plot [0.0:6.6] sin(x) title 'sin(theta)'
```

5.2.4 Plotting from a Data File

Quite often you may obtain the data from a program and need to plot the data. Let's first see how such a data file (`plot.txt`) should look like and the commands to be used.

```
#x               y               fitted y
#----------------------------------------
0.500000        1.100000        1.144973
1.000000        1.300000        1.636346
1.500000        2.000000        2.127720
2.000000        2.700000        2.619093
2.500000        3.400000        3.110466
3.000000        3.300000        3.601839
3.500000        4.400000        4.093212
4.000000        4.800000        4.584586
4.500000        5.100000        5.075959
5.000000        5.500000        5.567332
5.500000        6.000000        6.058705
6.000000        6.700000        6.550079
6.500000        7.000000        7.041452
7.000000        7.800000        7.532825
7.500000        8.100000        8.024198
8.000000        8.300000        8.515572
9.000000        9.600000        9.498318
10.000000       10.000000       10.481065
10.500000       11.100000       10.972438
11.000000       11.500000       11.463811
```

```
gnuplot> plot "plot.txt" u 1:2 w p;
gnuplot> plot "plot.txt" u 1:3 w p;
gnuplot> plot "plot.txt" u 1:2 w p 3, "plot.txt" u 1:3 w l 4
```

Note the following in the data file and the *Gnuplot* commands.

- *Gnuplot* ignores lines with #.
- The data should be presented as columns.
- Unless specified, the data in the 1st column are taken as `x` values and the corresponding elements in the 2nd column as `y` values.
- how to plot the data of the 3rd column *.vs.* that in the 1st column.
- how to put two plots in the same graph.

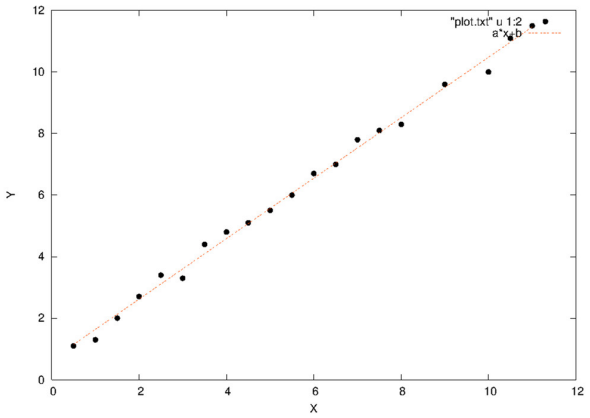
5.2.5 Fitting Data Points

Gnuplot may be used for fitting data points to a given function and obtain the best values for the parameters in the function. The following commands may be used for fitting the data points in the previous section to a straight line of the form $y = ax + b$ where a and b are the parameters to be determined. As mentioned earlier you may put the relevant *gnuplot* commands together in a file, say `fit.gnu` and use repeatedly.

```
set xlabel 'X'
set ylabel 'Y'
set term jpeg
set out 'fit.jpg'
fit [0.0:11.0] a*x+b "plot.txt" via a, b
plot "plot.txt" u 1:2 7, a*x+b  8
```

Note that the function to be fitted is given along with the *fit* command itself. Alternatively you may define a *function* (particularly if the function is rather complicated) and use it in the *fit* command:

```
set xlabel 'X'
set ylabel 'Y'
set term jpeg
set out 'fit.jpg'
f(x) = a*x+b
fit [0.0:11.0] f(x) "plot.txt" via a, b
plot "plot.txt" u 1:2 7, a*x+b  8
```



Note that *7* and *8* in the last line of `fit.gnu` control the point type and colour of the plots. The resultant figure is shown above.
Try fitting the data with $y = ax^2 + bx + c$ *or higher ploynomials and see the values of the fitted parameters.*

Chapter 6

Document preparation with LaTeX

6.1 Introduction to LaTeX

a scientist's tool for document preparation

LaTeXis the favourite tool for many scientists for preparing scientific documents. Let's start at the very beginning. You first prepare a text file, usually with **.tex** extension.

```
\documentclass[12pt,a4paper]{article}
\begin{document}
I am studying M.Sc.-II in Physics. We have a Computer Applications course.
This is my first \LaTeX document. Please note the use of
\back LaTeX
and it's effect. \\
.............................. \\
.............................. \\
\end{document}
```

You have to process the text file, say **abc.tex**, using the following commands, prepare the .pdf file and then view it. Portable

```
$ pdflatex abc
$ evince abc.pdf
```

What you see is the following:

I am studying M.Sc.-II in Physics. We have a Computer Applications course.

This is my first LaTeX document. Please note the use of \LaTeX and it's effect.

6.2 Basics of fonts

Let us first review the use of different fonts and font effects in LaTeX: You put the relevant part of the text within curly braces. The curly braces is the boundary for a particular `environment`.

```
{\em This is for emphasis.} \\
This is is regular text. \\
{\it This is for italics.}
```

This is for emphasis.
This is regular text.
This is for italics.

Note: if you want a line break, use \\ as shown above. For a new paragraph, leave a blank line.

```
{\bf This is for bold font.} \underline{This is for underline.}
```

This is for bold font. This is for underline.

```
\begin{verbatim }
This is for verbatim. See the dictionary to find out what it means.
\end{verbatim }

This is for verbatim. See the dictionary to find out what it means.
```

```
You may have different types of alignment of your text. Default is left
alignment.
\begin{flushleft}
You may align your text ‘left’. \\ And I don’t care, it is your document.
\end{flushleft}
\begin{flushright}
You may align your text ‘right’. \\ That is, spoil it any way you like.
\end{flushright}
\begin{center}
You may align your text ‘center’. \\ If you like it, that is.
I hope this is clear by now that \\ is used for breaking lines.
\end{center}
```

You may have different types of alignment of your txt. Default is left alignment.

You may align your text left.
And I don’t care, it is your document.

You may align your text right.
That is, spoil it in any way you like.

You may align your text centre.
I hope this is clear by now that \\ is used for line breaks.

6.3 Use of mathematical symbols

If you want to use mathematical symbols, references, etc., you have to include a few special packages. Then **abc.tex** will look as follows. Note that anything followed by % is a comment line and is ignored by LaTeX. Such comment lines may be placed anywhere in the **abc.tex** file. You may also define the width and height of your document as shown below.

```
\documentclass[12pt,a4paper]{article}
% ------------------------------------------------------------------------
\setlength{\textwidth}{17cm}
\setlength{\textheight}{23cm}
% ------------------------------------------------------------------------
\usepackage{amsmath}
\usepackage{amssymb}
\usepackage{pifont}
\usepackage{varioref}
\usepackage{graphics}
\usepackage{rotating}
% ------------------------------------------------------------------------
\begin{document}
Here put your text, equation, tables, figures here.
.................................
.................................
\end{document}
```

The following symbols (and many more) have to be used within *math environment* which is discussed in the next section.

6.4 Mathematical expressions and equations

We may have mathematical expressions within a paragraph, as a separate line or as an equation or as system of equations. Try the simple examples first before we get into more involved ones.

```
If we take the three sides of a triangle as vectors we may write the
following equation: $\vec{A} + \vec{B} + \vec{C} = 0$
```

If we take the three sides of a triangle as vectors we may write the following equation: $\vec{A}+\vec{B}+\vec{C} = 0$

```
If we take the three sides of a triangle as vectors we may write the
following equation: $$\vec{A} + \vec{B} + \vec{C} = 0$$
```

If we take the three sides of a triangle as vectors we may write the following equation:

$$\vec{A} + \vec{B} + \vec{C} = 0$$

```
If we take the three sides of a triangle as vectors we may write the
following equation:
\begin{equation}
\vec{A} + \vec{B} + \vec{C} = 0
\end{equation}
```

If we take the three sides of a triangle as vectors we may write the following equation:

$$\vec{A} + \vec{B} + \vec{C} = 0 \tag{6.1}$$

Notice the `equation` environment and the corresponding effects. Now for writing a set of consecutive equations we can use the following:

```
The electric and magnetic fields $\vec{E}$ and $\vec{B}$ satisfy the following
equations:
\begin{eqnarray}
\vec{\nabla} . \vec{E} & = & \frac{\rho}{\epsilon_{0}} \\
\vec{\nabla} . \vec{B} & = & 0 \\
\vec{\nabla}\times\vec{E} & = & -\frac{\partial \vec{B}}{\partial t} \\
\vec{\nabla}\times\vec{B} & = & \mu_{0}\vec{J} +
                          \mu_{0}\epsilon_{0}\frac{\partial \vec{E}}{\partial t}
\end{eqnarray}
```

The electric and magnetic fields $\vec{E}$ and $\vec{B}$ satisfy the following equations:

$$\vec{\nabla}.\vec{E} = \frac{\rho}{\epsilon_0} \tag{6.2}$$

$$\vec{\nabla}.\vec{B} = 0 \tag{6.3}$$

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{6.4}$$

$$\vec{\nabla} \times \vec{B} = \mu_0 \vec{J} + \mu_0 \epsilon_0 \frac{\partial \vec{E}}{\partial t} \tag{6.5}$$

6.4.1 References to Equations

```
It is also well known that
\begin{equation}
sec^{2}\theta - tan^{2}\theta = 1 \label{eqn:sec}
\end{equation}
You may like to refer an equation in the document and this is how to do it.
\begin{equation}
\frac{sinA}{a} = \frac{sinB}{b} = \frac{sinC}{c} \label{eqn:ratio}
\end{equation}
Equation \ref{eqn:ratio} is well known in trigonometry.
The following relation  (equation \ref{eqn:unity}) can be easily proved.
\begin{equation}
sin^{2}\theta + cos^{2}\theta = 1 \label{eqn:unity}
\end{equation}
```

It is also well known that

$$sec^2\theta - tan^2\theta = 1 \tag{6.6}$$

You may like to refer an equation in the document and this is how to do it.

$$\frac{sinA}{a} = \frac{sinB}{b} = \frac{sinC}{c} \tag{6.7}$$

Equation 6.7 is well known in trigonometry. The following relation (equation 6.8) can be easily proved.

$$sin^2\theta + cos^2\theta = 1 \tag{6.8}$$

You may have noticed the use of superscript. Now see the use of subscript.

```
$$\vec{P}.\vec{P} = P_{x}^{2} + P_{y}^{2} + P_{z}^{2}$$
```

$$\vec{P}.\vec{P} = P_x^2 + P_y^2 + P_z^2$$

6.5 Tables

Tables provide a convenient way of presenting data. Proper tabulation of data/information (text, numbers, symbols) is very important in preparing scientific documents. You have already seen some tabulation in this document itself.

6.5.1 Tabulation

If you are interested in making tables, carefully watch the next few paragraphs, and watch the differences. Also note carefully how a proper table is written with all the requirements as in table 6.2.

```
\begin{tabular}{rlcll}
   & Name        & Date of birth & Place of Birth & Address \\
1. & Putu Pal    & 1985          & Chandanpur     & Janai   \\
2. & Patal Ghosh & 1978          & Bhedia         & Bolpur  \\
3. & Bunchi Saha & 1991          & Labhpur        & Kolkata
\end{tabular}
```

	Name	Date of birth	Place of Birth	Address
1.	Putu Pal	1985	Chandanpur	Janai
2.	Patal Ghosh	1978	Bhedia	Bolpur
3.	Bunchi Saha	1991	Labhpur	Kolkata

```
\begin{tabular}{rllll}
\hline
   & Name        & Date of birth & Place of Birth & Address \\
\hline
```

```
1. & Putu Pal    & 1985          & Chandanpur     & Janai   \\
2. & Patal Ghosh & 1978          & Bhedia         & Bolpur   \\
3. & Bunchi Saha & 1991          & Labhpur        & Kolkata \\
\hline
\end{tabular}
```

	Name	Date of birth	Place of Birth	Address
1.	Putu Pal	1985	Chandanpur	Janai
2.	Patal Ghosh	1978	Bhedia	Bolpur
3.	Bunchi Saha	1991	Labhpur	Kolkata

```
\begin{tabular}{|r|l|l|l|l|}
\hline
   & Name        & Date of birth & Place of Birth & Address \\
\hline
1. & Putu Pal    & 1985          & Chandanpur     & Janai   \\
2. & Patal Ghosh & 1978          & Bhedia         & Bolpur   \\
3. & Bunchi Saha & 1991          & Labhpur        & Kolkata \\
\hline
\end{tabular}
```

	Name	Date of birth	Place of Birth	Address
1.	Putu Pal	1985	Chandanpur	Janai
2.	Patal Ghosh	1978	Bhedia	Bolpur
3.	Bunchi Saha	1991	Labhpur	Kolkata

6.5.2 A Proper Table

```
Now we are making a proper table which may be refered to as table
\ref{tab:list} from anywhere in the text using the label as shown.
\begin{table}[htbp]
\begin{tabular}{|r|l|l|l|l|}
\hline
   & Name        & Date of birth & Place of Birth & Address \\
\hline
1. & Putu Pal    & 1985          & Chandanpur     & Janai   \\
2. & Patal Ghosh & 1978          & Bhedia         & Bolpur   \\
3. & Bunchi Saha & 1991          & Labhpur        & Kolkata \\
\hline
\end{tabular}
\caption{This a table with caption, table number and may be referenced
         anywhere in the text.}
\label{tab:list}
\end{table}
```

Now we are making a proper table which may be refered to as table 6.2 from anywhere in the text using the label as shown.

$\geq$	\ge	$\leq$	\le	$\langle$	\langle
$\sum$	\sum	$\approx$	\approx	$\rangle$	\rangle
$\ll$	\ll	$\gg$	\gg	$\leq$	\le
$\neq$	\ne	$\hat{x}$	\hat{x}	$\underbrace{A = B + C}$	\underbrace {A = B+C}
∂	\partial	$\vec{A}$	\vec{A}	$\leftarrow$	\leftarrow
$\square$	\square	$\bar{A}$	\bar{A}	$\rightarrow$	\rightarrow
$\sim$	\sim	$\simeq$	\simeq	$\Leftarrow$	\Leftarrow
$\gtrsim$	\gtrsim	$\mp$	\mp	$\hbar$	\hbar
$\emptyset$	\emptyset	$\pm$	\pm	$\Rightarrow$	\Rightarrow
∇	\nabla	$\propto$	\propto	$\not\Rightarrow$	\not\Rightarrow
∞	\infty	$\prod$	\prod	$\not\Leftarrow$	\not\Leftarrow
$\longrightarrow$	\longrightarrow	$\longleftarrow$	\longleftarrow	$\triangledown$	\triangledown
$\in$	\in	$\notin$	\not\in	$\cap$	\cap
$\cup$	\cup	$\bigcup$	\bigcup	$\bigcap$	\bigcap
$\hbar$	\hbar	$\Re$	\re	$\Im$	\im
$\int$	\int	$\iint$	\iint	$\iiint$	\iiint
$\oint$	\oint	$\int \cdots \int$	\idotsint	$\angle$	\angle
$\oplus$	\oplus	$\perp$	\perp	$\mid$	\mid
$\equiv$	\equiv				

Table 6.1: Some commonly used symbols in Physics and mathematics.

	Name	Date of birth	Place of Birth	Address
1.	Putu Pal	1985	Chandanpur	Janai
2.	Patal Ghosh	1978	Bhedia	Bolpur
3.	Bunchi Saha	1991	Labhpur	Kolkata

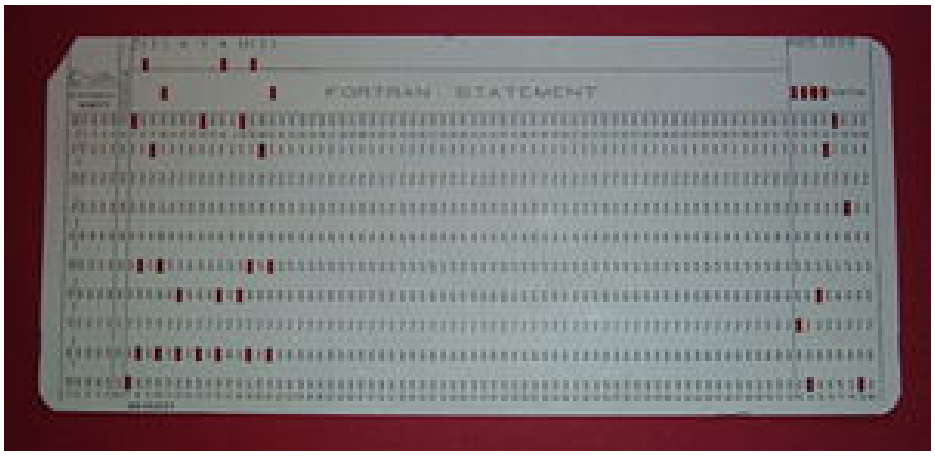
Table 6.2: This a table with caption, table number and may be referenced in the text.

`[htbp]` refers to the positioning of the table in the text. Often, to adjust space it may not be possible to place a table immediately after the preceding text. The letter `h` tells the processor to place it immediately after the preceding text. `t`, `b` and `p` refers to *top of the page*, *bottom of the page next page* respectively. You may restrict the options, but it is better not to do so.

6.6 Figures

Often we include graphs, figures, diagrams, even photographs in documents. We need to present them in a proper way with caption figure number and reference it in the document properly. The following line will include a figure (in .eps or .ps format) in the text.

```
\includegraphics[width=0.7\textwidth]{PunchedFortranCard.jpg}
```



However a better way to do is to use the `figure` environment for doing this with caption and label.

As you might have noticed you can adjust the size of the figure using the `[width=\textwidth]` part. In this case the width of the figure will be equal to the width of the printed part of the paper. See the figure 6.2 and note the difference.

```
\begin{figure}[htbp]
\bcent
\includegraphics[width=\textwidth]{Calvin-Unfair.jpg}
\ecent
\caption{A comic is simple in technique and direct in approach.}
\label{fig:net}
\end{figure}
```

You may include more than one diagram/plot/figure in one `figure` environment, see figure 6.3.

```
\begin{figure}[tbp]
\bcent
\includegraphics[width=0.8\textwidth]{Calvin-Invention.jpg}
\includegraphics[width=0.8\textwidth]{PunchedFortranCard.jpg}
```

Figure 6.1: A comic is simple in technique and direct in approach.

Figure 6.2: Punched cards were used in computers till the 80's.

```
\includegraphics[width=0.8\textwidth]{Calvin-History.png}
\ecent
\caption{You can always try a new approach to the most common problem
         (top) the punched card used in computers till the 80's (middle) and a
 new approach to history.}
\label{fig:punchnet}
\end{figure}
```

Some of the options you may try are given below.

```
\includegraphics[height=\textheight]{PunchedFortranCard.jpg}
\includegraphics[width=\textwidth, height=0.8\textheight]{PunchedFortranCard.jpg}
\includegraphics[width=0.8\textheight, angle=90]{PunchedFortranCard.jpg}
\includegraphics[width=\textwidth]{PunchedFortranCard.jpg}
\includegraphics[width=\textwidth]{PunchedFortranCard.jpg}
\includegraphics[width=\textwidth]{PunchedFortranCard.jpg}

\begin{figure}[tbp]
\bcent
\includegraphics[width=0.8\textwidth]{CalvinInvention.jpg}
\includegraphics[width=0.8\textwidth]{PunchedFortranCard.jpg}
\ecent
\caption{You can always try a new approach to the most common problem
```

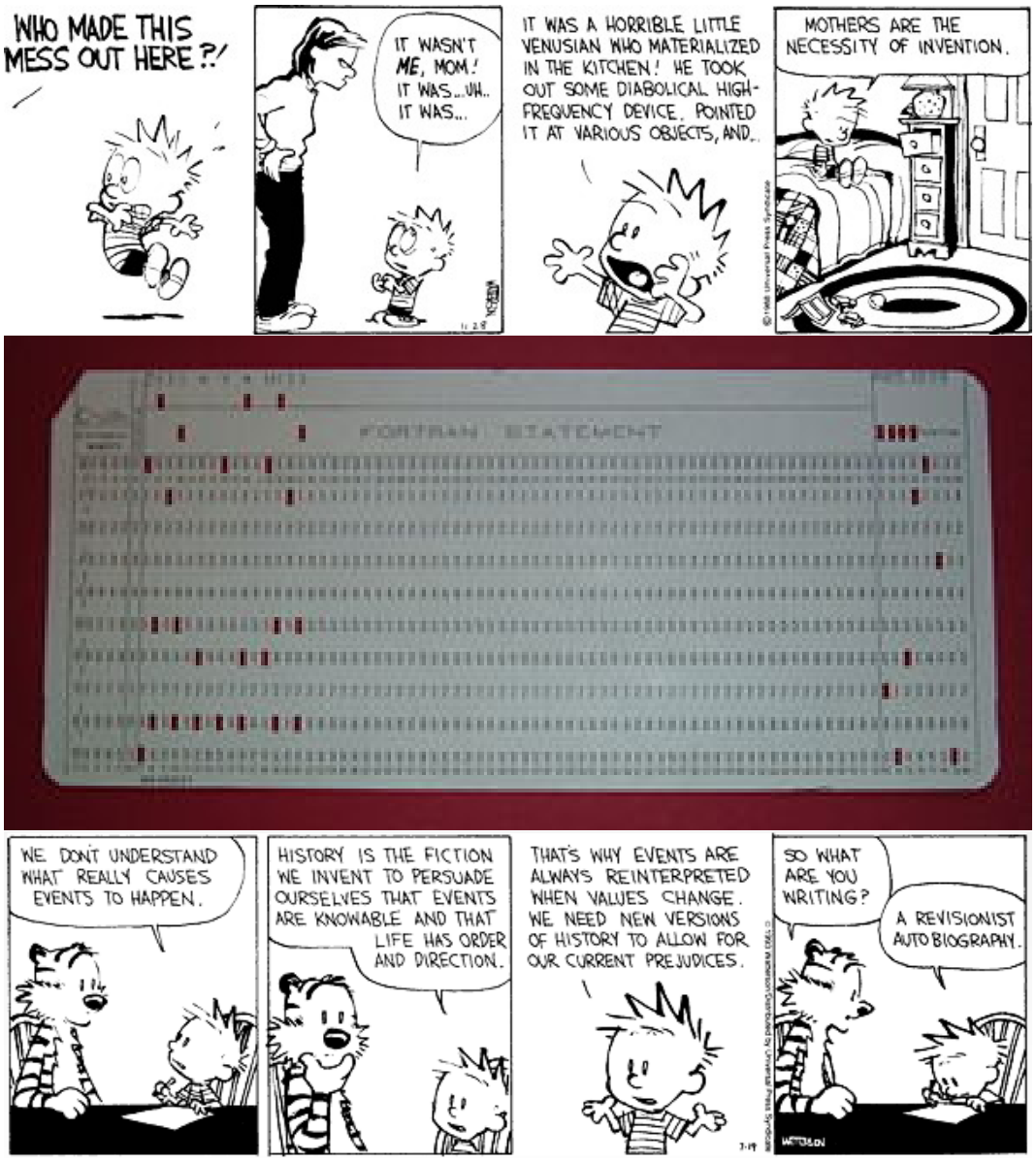


Figure 6.3: You can always try a new approach to the most common problem (top) the punched card used in computers till the 80's (middle) and a new approach to history.

```
         (top) and the punched card used in computers till the 80's (bottom).}
\label{fig:punchnet}
\end{figure}
```