

1.2×10^{-38} to 3.4×10^{38} can be represented. In double precision, eight bytes are usually used for each floating number. Among the 64 bits here, 1 bit is for the sign, 11 bits for the exponent, and 52 bits for the mantissa. Floating numbers with absolute values approximately in the range from 2.2×10^{-308} to 1.8×10^{308} may be represented by such an arrangement.

If we get a floating number whose absolute value is too small to be represented in a computer, an *underflow* condition is created. This happens, for example, when we get a single precision floating number with absolute value less than 10^{-38} . The usual course of action on such occasions is to replace the number by zero. If a number is produced with an absolute value much larger than 10^{38} in single precision or 10^{308} in double precision, an *overflow* condition is created. In this case, the usual response of the computer is to suspend the calculation, as an error will be introduced if we replace the number by anything else. Generally, one should check for possible underflows and, in particular, overflows in a program where such conditions are likely to occur and take the appropriate response.

The maximum number of significant figures that can be achieved in a floating number calculation is limited, in the first place, by the number of bits assigned to the mantissa. In single precision, the maximum is roughly seven significant figures and in double precision it is possible to reach 15 significant figures. Since the maximum number of significant figures is limited, *truncation* errors become a part of any floating number calculations. One important consideration in numerical work is to choose an algorithm that minimizes the truncation errors. If this is not done, the numerical errors may accumulate and become so large that no significant figures are left in the final results. At the same time, there is no use in trying to design an algorithm with a precision beyond the limitations imposed by truncation errors. We shall see different aspects of these considerations in many of the examples following.

1-3 Programming language and program library

In addition to hardware, such as central processor, memory and disk, software is also an integral part of any computer. As users, we do not wish to be overly involved in all the aspects that make the machine work for us; however, some background knowledge is useful. In getting a computer to solve our problem, the pieces of the software of most direct concern to us are the programming language and the subroutines to carry out some of the tasks to solve our problem.

High-level programming language For a computer to carry out a task, it is necessary to give it a set of specific instructions. This is generally referred as programming the computer. From a user point of view, it is desirable to give these instructions in terms of a language that is close to how we think of the problem, for example, in terms of equations. On the other hand, since the computer hardware is designed to carry out a very limited number of basic operations, we are still a long way from the possibility of such a direct communication. Furthermore, equations, by themselves,

are sometimes incomplete in describing a problem and, as a result, inadequate for a machine to carry out the calculations implied. For this reason, computer languages are developed as the interface between the user and the machine.

For our interest here, we shall be concerned with what are normally called high-level languages. That is, languages that are close to how users think of the problem and yet can be translated, or *compiled* into computer instructions. In scientific computations, C, Fortran, and Lisp are the three examples of high-level languages that come to mind most readily. Fortran is fairly easy to learn and has the support of a large number of subroutine libraries for scientific calculations developed over the years. Furthermore, because of its simple structure, it is relatively easy to develop highly optimized compilers. However, Fortran lacks the flexibility of the more modern C language. With the development of libraries accessible to both Fortran and C (as well as several other high-level languages) C is quickly becoming the language of choice in scientific computations. Lisp is more widely used in symbolic manipulation and is also essential for those interested in some of the finer details of computer algebra. Most of the discussions in this book are independent of computer languages. However, since Fortran remains to be the more popular medium of communication in numerical applications, the examples are based on Fortran.

Subroutine library Even with a high-level computer language, it will be quite tedious in most cases if we have to program everything needed in the project. Fortunately, among the software normally accompanying a computer these days, there is a large number of subroutines and utilities to carry out quite a few of the “routine” tasks in computation. For our purpose, we can divide these supporting programs into four categories, intrinsic library, system subroutines, utilities, and application library.

Many parts of a calculation are often common to a variety of problems. One obvious example is the trigonometry functions. For this reason, each high-level language compiler is usually associated with a standardized *intrinsic library* containing many of the functions generally needed in calculations. As an example, the functions in the intrinsic library of Fortran is given in Table 1-2.

Besides the intrinsic library, computers are equipped with a number of system routines that are special to the operating system, either in the way they are used or because they are specially adapted for the particular computer. One example is the routine to read the system clock so that one can, among others, print out the time when a calculation is done. Since the time is related to the master clock that, in a sense, sets the pace for the computer, the system clock routine depends very much on how the computer is built. Another example is the routine for us to ask whether an overflow or underflow condition has occurred so that we can take appropriate actions.

In using a computer, we need also to carry out a number of tasks not directly related to the problem to be solved. For example, to write a program, a text editor is required to enter each line of the code and to make corrections. We need also utilities to maintain files on the disk and to check on the status of our job. It will be impossible to make use of modern computers without such tools.

Table 1-2: Intrinsic Fortran functions.

Name	Function	Name	Function
Type conversion		Trigonometric functions	
INT	To integer	COS	Cosine
REAL	To real	SIN	Sine
DBLE	To double precision	TAN	Tangent
CMPLX	To complex	ACOS	Arc cosine
AINT	Truncation to nearest integer	ASIN	Arc sine
ANINT	Nearest whole number	ATAN	Arc tangent (1 argument)
NINT	Nearest integer	ATAN2	Arc tangent (2 arguments)
Numeric functions		Hyperbolic functions	
ABS	Absolute value	COSH	Hyperbolic cosine
MOD	Modulo	SINH	Hyperbolic sine
SIGN	Transfer of sign	TANH	Hyperbolic tangent
DIM	Positive difference	Character functions	
MAX	Maximum	LGE	Lexical \geq
MIN	Minimum	LGT	Lexical $>$
DPROD	Double precision product	LLE	Lexical \leq
AIMAG	Imaginary part	LLT	Lexical $<$
CONJG	Conjugate	CHAR	Integer to character
Transcendental functions		ICHAR	Character to integer
SQRT	Square root	INDEX	Location of string
EXP	Exponential	LEN	Length of string
LOG	Natural logarithm		
LOG10	Logarithm base 10		

Application libraries, as the name implies, are more specific to the particular type of work we wish to carry out. For example, for calculations involving matrices, one of the important library is lapack[2] which contains a large number of routines to perform matrix operations. Each user may also have a collection of routines accumulated over time that are useful on a personal basis. There are also packages to make plots, to carry out algebraic manipulations, and to do the word processing for publication. All these also constitute an indispensable part of making the computer work for us.

A particular group of routines that merits special mentioning is the basic linear algebra subprograms, or BLAS, both for the functions they perform and as an example of application library. In linear algebra, as well as many other types of calculations, we often need to work with vectors and matrices. As an example, we can think of taking the scalar or dot product of two vectors (dot)

$$\alpha = \mathbf{x} \cdot \mathbf{y}$$

or adding a constant times one vector to another vector (axpy)

$$\mathbf{y} = \alpha \mathbf{x} + \mathbf{y}$$

where α is a scalar quantity and \mathbf{x} and \mathbf{y} are vector quantities. Since the operations are fairly basic it is not difficult to write a few lines of code to carry out the calculations.

One of the reasons for using routines in BLAS to carry out such basic steps lies in the potential for optimization. The work that can be taken over by BLAS are often some of the more time consuming parts in many calculations. As a result, it is useful to make them as efficient as possible. This means writing the codes in a language closer to the operations of the computer itself so that we can take advantage of the particular architecture of the machine we have in hand. For this reason, computers designed for intensive numerical work are often supplied with BLAS optimized for the particular system. As a result, it is possible to achieve performance far superior than what is possible with high-level languages alone. The usefulness of BLAS actually goes one step beyond this. Since these optimized routines are, by necessity, system dependent, it may be difficult to take programs depending on them to a computer with a different architecture. This is the question of portability of computer programs, an increasingly important issue as we are getting more and more into the situation that our codes must be used on a variety of "platforms." By standardizing BLAS and equipping each machine with its own optimized version, our code can be both efficiency and portable.

More detailed information on BLAS can be found in lapack[2] and references cited there. For computers without specially optimized BLAS library, a set of the routines in Fortran are available from netlib.[55] They are also useful for certain debugging purposes even on computers equipped with the optimized version.

Special libraries In addition to subprogram libraries that are useful for a large variety of calculations, there are also a number of application libraries written for specific tasks, such as matrix calculations, symbolic manipulations, and plotting. In general, they may be divided into two categories, public domain and commercial. Public domain software is usually available without charge and is "donated" by the authors for the benefit of users in general. With the spread of internet, it has become quite easy to obtain these codes once they are located. However, there are no depositories similar to central libraries for books for all the software available. In addition to papers published by the authors in such journals as *Computers in Physics* by the American Institute of Physics and *Computer Physics Communications* by North Holland, a good place to start is netlib.[55] Commercial software, on the other hand, is usually well advertised on the relevant journals to attract business.

In generally, the use of existing software, public domain, commercial, or one's own collection, tends to save the development time for a project. However, to make intelligent use of the these tools, the user must be knowledgeable in the general methods behind the packages so as to be able to judge if they are suitable for the job in hand. Furthermore, the onus is still on the user to test the packages thoroughly to ensure that they are used correctly and that they are right for the job.