Chapter 1. Preliminaries

1.0 Introduction

This book, like its predecessor edition, is supposed to teach you methods of numerical computing that are practical, efficient, and (insofar as possible) elegant. We presume throughout this book that you, the reader, have particular tasks that you want to get done. We view our job as educating you on how to proceed. Occasionally we may try to reroute you briefly onto a particularly beautiful side road; but by and large, we will guide you along main highways that lead to practical destinations.

Throughout this book, you will find us fearlessly editorializing, telling you what you should and shouldn't do. This prescriptive tone results from a conscious decision on our part, and we hope that you will not find it irritating. We do not claim that our advice is infallible! Rather, we are reacting against a tendency, in the textbook literature of computation, to discuss every possible method that has ever been invented, without ever offering a practical judgment on relative merit. We do, therefore, offer you our practical judgments whenever we can. As you gain experience, you will form your own opinion of how reliable our advice is.

We presume that you are able to read computer programs in `FORTRAN`, that being the language of this version of *Numerical Recipes* (Second Edition). The book *Numerical Recipes in C* (Second Edition) is separately available, if you prefer to program in that language. Earlier editions of *Numerical Recipes in Pascal* and *Numerical Recipes Routines and Examples in BASIC* are also available; while not containing the additional material of the Second Edition versions in `C` and `FORTRAN`, these versions are perfectly serviceable if `Pascal` or `BASIC` is your language of choice.

When we include programs in the text, they look like this:

```
SUBROUTINE flmoon(n,nph,jd,frac)
INTEGER jd,n,nph
REAL frac,RAD
PARAMETER (RAD=3.14159265/180.)
```

Our programs begin with an introductory comment summarizing their purpose and explaining their calling sequence. This routine calculates the phases of the moon. Given an integer `n` and a code `nph` for the phase desired (`nph` $= 0$ for new moon, 1 for first quarter, 2 for full, 3 for last quarter), the routine returns the Julian Day Number `jd`, and the fractional part of a day `frac` to be added to it, of the `n`th such phase since January, 1900. Greenwich Mean Time is assumed.

```
INTEGER i
REAL am,as,c,t,t2,xtra
c=n+nph/4.                              This is how we comment an individual line.
t=c/1236.85
t2=t**2
```

```
as=359.2242+29.105356*c                  You aren't really intended to understand this al-
am=306.0253+385.816918*c+0.010730*t2       gorithm, but it does work!
jd=2415020+28*n+7*nph
xtra=0.75933+1.53058868*c+(1.178e-4-1.55e-7*t)*t2
if(nph.eq.0.or.nph.eq.2)then
    xtra=xtra+(0.1734-3.93e-4*t)*sin(RAD*as)-0.4068*sin(RAD*am)
else if(nph.eq.1.or.nph.eq.3)then
    xtra=xtra+(0.1721-4.e-4*t)*sin(RAD*as)-0.6280*sin(RAD*am)
else
    pause 'nph is unknown in flmoon'     This is how we will indicate error conditions.
endif
if(xtra.ge.0.)then
    i=int(xtra)
else
    i=int(xtra-1.)
endif
jd=jd+i
frac=xtra-i
return
END
```

A few remarks about our typographical conventions and programming style are in order at this point:

- It is good programming practice to declare all variables and identifiers in explicit "type" statements (`REAL`, `INTEGER`, etc.), even though the implicit declaration rules of `FORTRAN` do not require this. We will always do so. (As an aside to non-`FORTRAN` programmers, the implicit declaration rules are that variables which begin with the letters `i,j,k,l,m,n` are implicitly declared to be type `INTEGER`, while all other variables are implicitly declared to be type `REAL`. Explicit declarations override these conventions.)
- In sympathy with modular and object-oriented programming practice, we separate, typographically, a routine's "public" or "interface" section from its "private" or "implementation" section. We do this even though `FORTRAN` is by no means a modular or object-oriented language: the separation makes sense simply as good programming style.
- The *public* section contains the calling interface and declarations of its variables. We find it useful to consider `PARAMETER` statements, and their associated declarations, as also being in the public section, since a user may want to modify parameter values to suit a particular purpose. `COMMON` blocks are likewise usually part of the public section, since they involve communication between routines.
- As the last entry in the public section, we will, where applicable, put a standardized comment line with the word `USES` (not a `FORTRAN` keyword), followed by a list of all external subroutines and functions that the routine references, excluding built-in `FORTRAN` functions. (For examples, see the routines in §6.1.)
- An introductory comment, set in type as an indented paragraph, separates the public section from the private or implementation section.
- Within the introductory comments, as well as in the text, we will frequently use the notation `a(1:m)` to mean "the array elements `a(1)`, `a(2)`, ..., `a(m)`." Likewise, notations like `b(2:7)` or `c(1:m,1:n)` are to be

interpreted as ranges of array indices. (This use of colon to denote ranges comes from FORTRAN-77's syntax for array declarators and character substrings.)

- The *implementation section* contains the declarations of variables that are used only internally in the routine, any necessary SAVE statements for static variables (variables that must be preserved between calls to the routine), and of course the routine's actual executable code.
- Case is not significant in FORTRAN, so it can be used to promote readability. Our convention is to use upper case for two different, nonconflicting, purposes. First, nonexecutable compiler keywords are in upper case (e.g., SUBROUTINE, REAL, COMMON); second, parameter identifiers are in upper case. The reason for capitalizing parameters is that, because their values are liable to be modified, the user often needs to scan the implementation section of code to see exactly how the parameters are used.
- For simplicity, we adopt the convention of handling all errors and exceptional cases by the pause statement. In general, we do not intend that you continue program execution after a pause occurs, but FORTRAN allows you to do so — if you want to see what kind of wrong answer or catastrophic error results. In many applications, you will want to modify our programs to do more sophisticated error handling, for example to return with an error flag set, or call an error-handling routine.
- In the printed form of this book, we take some special typographical liberties regarding statement labels, and do ... continue constructions. These are described in §1.1. Note that no such liberties are taken in the machine-readable *Numerical Recipes* diskettes, where all routines are in standard ANSI FORTRAN-77.

Computational Environment and Program Validation

Our goal is that the programs in this book be as portable as possible, across different platforms (models of computer), across different operating systems, and across different FORTRAN compilers. As surrogates for the large number of possible combinations, we have tested all the programs in this book on the combinations of machines, operating systems, and compilers shown on the accompanying table. More generally, the programs should run without modification on any compiler that implements the ANSI FORTRAN-77 standard. At the time of writing, there are not enough installed implementations of the successor FORTRAN-90 standard to justify our using any of its more advanced features. Since FORTRAN-90 is backwards-compatible with FORTRAN-77, there should be no difficulty in using the programs in this book on FORTRAN-90 compilers, as they become available.

In validating the programs, we have taken the program source code directly from the machine-readable form of the book's manuscript, to decrease the chance of propagating typographical errors. "Driver" or demonstration programs that we used as part of our validations are available separately as the *Numerical Recipes Example Book (FORTRAN)*, as well as in machine-readable form. If you plan to use more than a few of the programs in this book, or if you plan to use programs in this book on more than one different computer, then you may find it useful to obtain a copy of these demonstration programs.

Tested Machines and Compilers		
Hardware	O/S Version	Compiler Version
IBM PC compatible 486/33	MS-DOS 5.0	Microsoft Fortran 5.1
IBM RS6000	AIX 3.0	IBM AIX XL FORTRAN Compiler/6000
IBM PC-RT	BSD UNIX 4.3	"UNIX Fortran 77"
DEC VAX 4000	VMS 5.4	VAX Fortran 5.4
DEC VAXstation 2000	BSD UNIX 4.3	Berkeley f77 2.0 (4.3 bsd, SCCS lev. 6)
DECstation 5000/200	ULTRIX 4.2	DEC Fortran for ULTRIX RISC 3.1
DECsystem 5400	ULTRIX 4.1	MIPS f77 2.10
Sun SPARCstation 2	SunOS 4.1	Sun Fortran 1.4 (SC 1.0)
Apple Macintosh	System 6.0.7 / MPW 3.2	Absoft Fortran 77 Compiler 3.1.2

Of course we would be foolish to claim that there are no bugs in our programs, and we do not make such a claim. We have been very careful, and have benefitted from the experience of the many readers who have written to us. If you find a new bug, please document it and tell us!

Compatibility with the First Edition

If you are accustomed to the *Numerical Recipes* routines of the First Edition, rest assured: almost all of them are still here, with the same names and functionalities, often with major improvements in the code itself. In addition, we hope that you will soon become equally familiar with the added capabilities of the more than 100 routines that are new to this edition.

We have retired a small number of First Edition routines, those that we believe to be clearly dominated by better methods implemented in this edition. A table, following, lists the retired routines and suggests replacements.

First Edition users should also be aware that some routines common to both editions have alterations in their calling interfaces, so are not directly "plug compatible." A fairly complete list is: `chsone`, `chstwo`, `covsrt`, `dfpmin`, `laguer`, `lfit`, `memcof`, `mrqcof`, `mrqmin`, `pzextr`, `ran4`, `realft`, `rzextr`, `shoot`, `shootf`. There may be others (depending in part on which printing of the First Edition is taken for the comparison). If you have written software of any appreciable complexity that is dependent on First Edition routines, we do *not* recommend blindly replacing them by the corresponding routines in this book. We do recommend that any new programming efforts use the new routines.

About References

You will find references, and suggestions for further reading, listed at the end of most sections of this book. References are cited in the text by bracketed numbers like this [1].

Because computer algorithms often circulate informally for quite some time before appearing in a published form, the task of uncovering "primary literature"

Previous Routines Omitted from This Edition		
Name(s)	Replacement(s)	Comment
ADI	mglin or mgfas	better method
COSFT	cosft1 or cosft2	choice of boundary conditions
CEL, EL2	rf, rd, rj, rc	better algorithms
DES, DESKS	ran4 now uses psdes	was too slow
MDIAN1, MDIAN2	select, selip	more general
QCKSRT	sort	name change (SORT is now hpsort)
RKQC	rkqs	better method
SMOOFT	use convlv with coefficients from savgol	
SPARSE	linbcg	more general

is sometimes quite difficult. We have not attempted this, and we do not pretend to any degree of bibliographical completeness in this book. For topics where a substantial secondary literature exists (discussion in textbooks, reviews, etc.) we have consciously limited our references to a few of the more useful secondary sources, especially those with good references to the primary literature. Where the existing secondary literature is insufficient, we give references to a few primary sources that are intended to serve as starting points for further reading, not as complete bibliographies for the field.

The order in which references are listed is not necessarily significant. It reflects a compromise between listing cited references in the order cited, and listing suggestions for further reading in a roughly prioritized order, with the most useful ones first.

The remaining two sections of this chapter review some basic concepts of programming (control structures, etc.) and of numerical analysis (roundoff error, etc.). Thereafter, we plunge into the substantive material of the book.

CITED REFERENCES AND FURTHER READING:

Meeus, J. 1982, *Astronomical Formulae for Calculators*, 2nd ed., revised and enlarged (Richmond, VA: Willmann-Bell). [1]

1.1 Program Organization and Control Structures

We sometimes like to point out the close analogies between computer programs, on the one hand, and written poetry or written musical scores, on the other. All three present themselves as visual media, symbols on a two-dimensional page or computer screen. Yet, in all three cases, the visual, two-dimensional, *frozen-in-time* representation communicates (or is supposed to communicate) something rather