

# 4

## Programming Tools

### 4.1 Choosing a Programming Language

Any programming language one is familiar with can be used for computational work, despite the fact that some people believe fanatically that their own favorite programming language is superior to all other languages.

C and Fortran, for example, are well suited for intensive scientific computing. Both languages are fast in execution, quick to program, and widely known. There exist large program repositories and fast compilers for them.

C is the common tongue of programmers and computer scientists. It has dynamic memory allocation, that is, the size of an array can be changed while the program is executing. The new C99 standard includes intrinsic complex arithmetic, which was absent from earlier C standards. C++ tends to be a little slower than C, because its more complex code is harder to understand by the compiler and the speed optimization is often not as good as for C. C++ becomes advantageous when code gets large and needs to be maintained.

Fortran is intended as programming language tailored to the needs of scientists and engineers and as such it will always remain particularly suited for this purpose. Fortran here means Fortran 90, or a later version, which greatly extends the capabilities of earlier Fortran standards. Fortran has fast integer powers ( $3^7$  is faster than  $3^{6.9}$ ), which C lacks. Fortran allows unformatted output (which is faster than formatted output and exactly preserves the accuracy of numbers). A major advantage of Fortran are its parallel computing abilities.

Fortran 77, compared to Fortran 90, is missing pointers and the pow-

erful parallel computing features. It lacks dynamic memory allocation, but already includes intrinsic complex arithmetic. Because of its simplicity and age, compiler optimization for Fortran 77 is the best available for any language.

Java tends to be slow and in its current implementation (2004) it has terrible roundoff properties, but it excels in portability.

Python is versatile and enables programmers to write code in less time, which makes it highly suited for scientific programming, but Python code usually does not compile to machine code, making it imperfect in execution speed.

High-performance dialects of several languages also exist, but are shorter lived and only marginally better.

Program code can be made executable by interpreters, compilers, or a combination of both. Interpreters read and immediately execute the source program line by line. Compilers process the entire program before it is executed, which permits better checking and speed optimization. Languages that can be compiled hence execute much faster than interpreted ones.

In this book Fortran and C are occasionally used. If you know one of these languages, you are probably also able to quickly understand the other by analogy. As a quick familiarization exercise, Table I shows a program in C and in Fortran that demonstrates similarities between the two languages.

The following features are not analogous: C is case-sensitive, Fortran is not. Array indices begin by default with 0 for C and with 1 for Fortran. Fortran output lines automatically end with a line break; not so in C. Initializing a variable with a value is done in C each time the subroutine is called, in Fortran only the first time the subroutine is called.



**Recommended Reading:** The best reference book on C is Kernighan & Ritchie, *The C Programming Language*, although as an introductory book it is challenging. A concise but complete coverage of Fortran is given by Metcalf & Reid, *Fortran 90/95 Explained*. There is

<pre> /* C program example */ #include &lt;math.h&gt; #include &lt;stdio.h&gt; void main() {   int i;   const int N=64;   float b,a[N];   b=-2.;   for(i=0;i&lt;N;i++) {     a[i]=sin(i/2.);     if (a[i]&gt;b) b=a[i];   }   b=pow(b,5.); b=b/N;   printf("%10.5f\n",b); } </pre>	<pre> ! Fortran program example program demo   implicit none   integer i   integer,parameter :: N=64   real b,a(N)   b=-2.   do i=1,N     a(i)=sin((i-1)/2.)     if (a(i)&gt;b) b=a(i)   enddo   b=b**5; b=b/N   print "(f10.5)",b end </pre>
--	---

Table 4-I: *Program examples that demonstrate similarities between two languages.*

good online Fortran documentation, for example at <http://www.liv.ac.uk/HPC/F90page.html>.

## 4.2 General-Purpose Mathematical Software Packages

There are ready-made software packages for numerical calculations. Many tasks that would otherwise require lengthy programs can be done with a few keystrokes. For instance, it only takes one command to find a root, say `FindRoot[sin(3 x)==x,{x,1}]`. Inverting a matrix may simply reduce to `Inverse[A]` or `1/A`. Such software tools have become so convenient and powerful that they are the preferred choice for many computational problems.

General-purpose mathematical software packages can be highly portable among computing platforms. For example, the exact same Matlab or

Mathematica programs can be run on different operating systems without modification.

Programs can be written for such software packages in their own application-specific language. Often these do not achieve the speed possible with languages like Fortran or C. One reason for that is the trade-off between universality and efficiency—a general method is not going to be the fastest. Further, we typically do not have access to the source codes to adjust them. Another reason is that, although individual commands may be compiled, a succession of commands is interpreted and hence slow. Such software can be of great help when a calculation takes little time, but they may be badly suited for time-intensive calculations.

In one popular application, the example program above would be:

```
% Matlab program example
N=64;
i=[0:N-1];
a=sin(i/2);
b=max(a);
b^5/N
```

Whether it is better to use a ready-made mathematical software package or write a program in a lower level language like C or Fortran depends on the task to be solved. Each has its domain of applicability. Practically it is advantageous to know two or three different languages or tools from different parts of the spectrum, then choose a tool appropriate for the given task.



Major general-purpose mathematical software packages that are currently popular: *Maple*, *Matlab*, and *Mathematica* do symbolic and numerical computations. *Matlab* is particularly strong and efficient for linear algebra tasks. *Octave* is open-source software that mimics *Matlab*, with numerical and graphical capabilities. There are also software packages that focus on data visualization and data analysis, such as *AVS* (*Advanced Visual Systems*), *IDL* (*Interactive Data Language*), and others.

### 4.3 Data Visualization

Graphics is an indispensable tool for data evaluation, program testing, and scientific analysis. We only want to avoid spending too much time on learning and coping with graphics software.

Often, data analysis is *exploratory*. It is thus desirable to be able to produce a graph quickly and with ease. We will want to take advantage of existing visualization software rather than write our own graphics routines, because writing graphics programs would be time consuming and such programs are often not portable. In all, simple graphics software can go a long way.

The interpretation of data formats varies among graphics programs. The type of line breaks, comment lines, the form of the exponent, or anomalously many digits can lead to misinterpretations.



*Gnuplot* is a simple and free graphics plotting program. It is quick to use and learn. Most graphs in this book are made with Gnuplot. The official Gnuplot site is [www.gnuplot.info](http://www.gnuplot.info). Another, similar tool is *Grace* (formerly *xmgr*), also freely available. A number of commercial software packages have powerful graphics capabilities, including general-purpose math packages listed above.

It can be advantageous to stick to mainstream packages, because they are widely available, can be expected to survive into the future, and tend to be quicker to learn. Migrating to a new software product may be costly in learning time.