

7

From Programs to Data Analysis

7.1 Canned Routines

Often there is no need to write subroutines ourselves. For common tasks there already exist collections of subroutines (see the sources listed below). Their reliability varies, but many good implementations are available.



Code Sources.

- The *Guide to Available Mathematical Software* <http://math.nist.gov> maintains a directory of subroutines from numerous public and proprietary repositories.
- *NETLIB* at www.netlib.org offers free sources by various authors and of varying quality.
- A more specialized, refereed set of routines is available to the public from the *Collected Algorithms of the ACM* at www.acm.org/calgo.
- *Numerical Recipes*, www.nr.com, explains and provides a broad and selective collection of reliable subroutines. Each program is available in C, C++, Fortran 77, and Fortran 90.
- Major commercial packages are *IMSL* (*International Mathematical and Statistical Library*) and *NAG* (*Numerical Algorithms Group*).
- The *Gnu Scientific Library* is a collection of open-source routines, www.gnu.org/software/gsl.

Certain tasks call for canned routines more than others. For special functions, such as the Error function, Bessel functions, the Gamma function, and so on, someone put a lot of thought into how to evaluate them efficiently. On the other hand, partial differential equation solvers demand great flexibility and appear in such variety that most of the time we can better write them on our own (“throw-away codes”).

In addition to subroutines provided by other programmers, there are also “libraries,” which are repositories of subroutines that are already compiled and whose source code we may be unable to see. Internal library functions are optimized for a particular hardware. Therefore, they are typically faster than portable programs, because they can take advantage of system specific features. A drawback is that the interface to an internal function can be machine dependent; name, required arguments, and output variables can vary with the individual library.

An example is the Fast Fourier Transform (FFT), an efficient algorithm for performing Fourier transforms. An internal FFT is typically multiple times faster than portable FFT routines. (An exceptional implementation is FFT-W, the “Fastest Fourier Transform in the West”. It is a portable source code that first detects what hardware it is running on and chooses different computational strategies depending on the specific hardware architecture. This way, it can simultaneously achieve portability and efficiency.)

7.2 Programming

“In computer programming, the technique of choice is not necessarily the most efficient, or elegant, or fastest executing one. Instead, it may be the one that is quick to implement, general, and easy to check.”

Numerical Recipes

To a large extent the same advice applies for scientific programming as for programming in general. Programs should be clear, robust, general. Only when performance is critical, and only in the parts where it is critical, should one compromise these principles. Most programmers find that it

is more efficient to write a part, test it, and then code the next part, rather than to write the whole program first and then start testing.

In other aspects, writing programs for research is different from software development. Research programs keep changing and are usually used only by the programmer herself or a small group of users. Further, simulations for research purposes often intend to chart unexplored territory of knowledge. Under these circumstances there is little reason to extract the last margin of efficiency, create nice user interfaces, write extensive documentation, or implement complex algorithms. All of that can actually be counterproductive.

It is easy to miss a mistake or an inconsistency within many lines of code. Already one wrong symbol in the program can invalidate the result. Program validation is a practical necessity. Some go so far as to add “blunders” to the list of common types of error in numerical calculations: roundoff, approximation errors, statistical errors, and blunders (typos). Absence of obvious contradictions may not be a sufficient standard of checking. Catching a mistake later or not at all may lead to a huge waste of effort, and this risk can be reduced by spending time on checking early on. One will compare with analytically known solutions, including trivial solutions, isolate pieces of code for testing, test independence from resolution or other parameters that should not matter, monitor variables, and use graphics to understand.

Programs undergo evolution. As time passes improvements are made on a program, bugs fixed, and the program matures. For time-intensive computations, it is thus never a good idea to make long runs right away. Moreover, the experience of analyzing the result of a short run might change which and in what form data are output. Simulations shorter than one minute allow for interactive improvements, and thus a rapid development cycle.

For lengthy runs one may wish to know how far the program has proceeded. While a program is executing its output is not immediately written to disk, because this may slow it down. This has the disadvantage that recent output is unavailable. Immediate output can be forced by closing the file and reopening it before further output occurs.

When data sets accumulate, it is practical to keep a log of the var-

ious simulations and their specifications. The situation is analogous to an experimentalist who maintains a laboratory notebook. Forgetting to write down one parameter might necessitate repetition of the experiment. And not labeling a sample can, under unfortunate circumstances, make it useless. Some people keep their notes not in an electronic file but in a real paper notebook.

7.3 Common Beginner's Mistakes

1. *Executing a program in the foreground for hours or days that produces lots of output on the screen.* Starting a program in the background and redirecting its output to a file is a simple technicality not known to all. Moreover, excessive output to the screen can slow down a program considerably.
2. *Lack of knowledge about numerical analysis.* Numerical analysis is typically not among the courses taken by science students, leading to unnecessary barriers and weaknesses in the process of numerical problem-solving.
3. *Not to enable compiler optimization.* I have witnessed again and again scientists who carry out day-long calculations, unaware that simply enabling optimization during compilation would shorten computation time considerably.
4. *Rewriting another person's program in another language or change its style.* Some feel this is a way to familiarize themselves with the program, but realistically this tends to be a waste of time.
5. *To implement a numerically intensive application with an unsuitable choice of software or language.*
6. *No testing before runs.* Error is practically certain.
7. *Carry out lengthy and complicated model calculations right away, instead of short and simple ones first.* Initially, there tends to be an iterative process of changes and improvements. Besides the obvious difference in cumulative computation time (that is, a series of

minute-long calculations is much shorter than a series of hour-long calculations), the mind has to sluggishly refocus.

8. *Not to resist the keyboard.* Fascinated by computer work or numerical analysis, some become absorbed with computer and numerical issues that ultimately contribute nothing to the research results. This is a major habitual problem in computational physics.
9. *Thinking the problem lies in the numerics when it does not.* We tend to mistrust what we know least, but often a program correctly executes with a wrong input or a faulty algorithm.
10. *Lousy evaluation of hard-earned simulation data.* After elaborate model implementation and long computations, some people spend unproportionally little effort on careful evaluation of model results.

7.4 Data Handling

Data can be either evaluated as they are computed (run-time evaluation) or stored and evaluated afterwards (post-evaluation). Post-evaluation allows changes and flexibility in the evaluation process, without having to repeat the run. Note that numbers can be calculated much faster than they can be written to any kind of output; it is easy to calculate far more than can be stored. For this reason, output is selective.

For checking purposes it is advantageous to create human readable output, that is, plain text. Plain text is also a highly portable file format; except, the end of line encoding can vary between operating system, which can be a nuisance, until one discovers how to convert them.

Data, whether real or from computer simulations, come in different formats, are created on different operating systems, have different symbols for comment lines, and come in a different shape from what you may need. Handy tools for file manipulation are operating system utilities and scripting languages. Sophisticated automated manipulations can be done with text processing languages such as `awk`, `perl`, and `sed`.

7.5 Modern Curve Fitting

Fitting straight lines by the least-square method is straightforward. For linear regression we minimize the quadratic deviations $E = \sum_i (y_i - a - bx_i)^2$, where x_i and y_i are the data and a and b are, respectively, the intercept and slope of a straight line. The extremum conditions $\partial E / \partial a = 0$ and $\partial E / \partial b = 0$ lead to the linear equations $\sum_i (y_i - a - bx_i) = 0$ and $\sum_i x_i (y_i - a - bx_i) = 0$, which can be explicitly solved for a and b . The popularity of linear regression is partially due to do the computational convenience the fit parameters can be obtained.

Some other functions can be reduced to linear regression, e.g., $y^2 = \exp(x)$. If errors are distributed Gaussian then linear regression finds the most likely fit, but a transformation of variables spoils this property. If the fitting function cannot be reduced to linear regression it is necessary to minimize the error as a function of the fit parameter(s) nonlinearly.

Fits with quadratically weighted deviations are not particularly robust, since an outlying data point can affect it significantly. Weighting proportional with distance, for instance, improves robustness. In this case, we seek to minimize $\sum_i |y_i - a - bx_i|$, where, again, x_i and y_i are the data and a and b are, respectively, the intercept and slope of a straight line. Differentiation with respect to a and b yields the following two conditions: $\sum_i \text{sgn}(y_i - a - bx_i) = 0$ and $\sum_i x_i \text{sgn}(y_i - a - bx_i) = 0$. Unlike for the case where the square of the deviations is minimized, these equations are not linear in a and b , because of the sign function sgn . They are however still easier to solve than by nonlinear root finding in two variables. The first condition says that the number of positive elements must be equal to the number of negative elements, and hence a is the median among the set of numbers $y_i - bx_i$. Since a is determined as a function of b in this relatively simple way, the remaining problem is to solve the second condition, which requires nonlinear root-finding in only one variable. This type of fitting, minimizing absolute deviations, is still sort of underutilized—as are a number of other fitting methods that are computationally more expensive than linear regression.



Recommended Reading: Valuable articles on computer ergonomics can be found in numerous sources, e.g., “Healthy Computing” at IBM’s <http://www.pcco.ibm.com/ww/healthycomputing/> or on the ergonomics site of the Department of Labor at <http://www.osha.gov/SLTC/etools/computerworkstations>.



Brainteaser: Stay away from the computer for a while. A major habitual problem is to not resist the urge to type on the computer.