$$\text{curl}\,\mathbf{u} = \hat{\mathbf{e}}_r \frac{1}{r\sin\theta}\left(\frac{\partial}{\partial\theta}(\sin\theta\, u_\phi) - \frac{\partial u_\theta}{\partial\phi}\right)$$
$$+ \hat{\mathbf{e}}_\theta \frac{1}{r\sin\theta}\left(\frac{\partial u_r}{\partial\phi} - \sin\theta\frac{\partial}{\partial r}(r u_\phi)\right)$$
$$+ \hat{\mathbf{e}}_\phi \frac{1}{r}\left(\frac{\partial}{\partial r}(r u_\theta) - \frac{\partial u_r}{\partial\theta}\right) \quad (16)$$

$$\nabla^2\psi = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial\psi}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\psi}{\partial\theta}\right)$$
$$+ \frac{1}{r^2\sin^2\theta}\frac{\partial^2\psi}{\partial\phi^2}. \quad (17)$$

These expressions are used when we discuss spherical waves in Section 2.4 and the earth's normal modes in Section 2.9.

A final point worth noting is that the elements of volume and surface used in integrals are different in spherical coordinates from rectangular coordinates. In spherical coordinates (Fig. A.7-5) there are several scale factors, so an element of surface area is

$$dS = r^2 \sin\theta\, d\theta d\phi, \quad (18)$$

and an element of volume is

$$dV = r^2 \sin\theta dr\, d\theta d\phi. \quad (19)$$

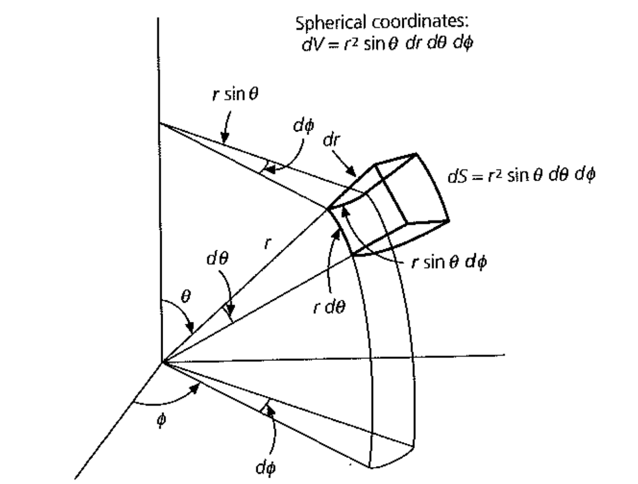

Fig. A.7-5 Definition of the element of volume in spherical coordinates. Unlike the case of Cartesian coordinates, the volume element in spherical coordinates in not a cube. (Marion, 1970. From *Classical Dynamics of Particles and Systems*, 2nd edn, copyright 1970 by Academic Press, reproduced by permission of the publisher.)

A.8 Scientific programming

Most seismological applications require computers, and these requirements, especially in exploration applications with very large data volumes, have spurred the development of computer software and hardware. Some remarks about the use of computers in seismology thus seem appropriate.

Computer usage in seismology includes several broad and overlapping categories:

- Computers are often used in data acquisition and recording systems.
- Data are initially displayed and manipulated using computers.
- Subsequent analysis is frequently done using computers. For example, seismograms can be filtered to enhance certain frequencies or combined to better resolve certain features.
- Theoretical, or *synthetic*, seismograms are often computed for a range of the parameters under study and compared to data to find the best fit.
- Computers are used to *invert* seismological data to determine the parameters of a model which best matches the data.
- Computer modeling is often used to draw geological inferences from seismological observations. For example, seismic velocity data are compared to the predictions of models for the velocity of rock as a function of composition, temperature, and pressure.

These applications often require *scientific programming*, a programming style used for essentially mathematical applications. Some problems in this book also require scientific programming. Although programming is a matter of personal style, this section discusses several points that may be helpful. The suggested reading provides some starting points for readers interested in pursuing these topics further.

A.8.1 Example: synthetic seismogram calculation

Consider a program to compute a synthetic seismogram for waves in a one-dimensional constant-velocity medium, a mathematically idealized string that illustrates features of wave behavior. The program is based on $u(x, t)$, the displacement as a function of position x and time t. The displacement is zero at the fixed ends of the string, $x = 0$ and $x = L$, between which waves travel at speed v. As in Section 2.2.5, the displacement can be written as the sum of the normal modes of the string, each of which is a standing wave with n half wavelengths along the string,

$$u_n(x, t) = \sin(n\pi x/L)\cos(\omega_n t), \quad (1)$$

and vibrates at a characteristic frequency, or *eigenfrequency*,

$$\omega_n = n\pi v/L. \quad (2)$$

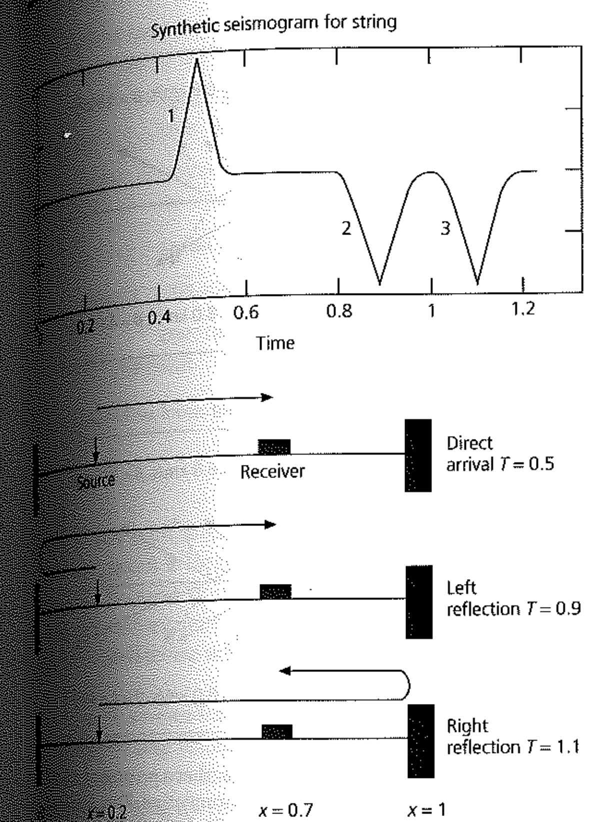

Fig. A.8-1 Top: Synthetic seismogram for a string showing the direct wave (1) and reflections (2, 3) from both ends. *Bottom*: Geometry of source and receiver positions, and the times of the direct and reflected arrivals.

A source at position x_s generates a pulse at time zero with ...; the propagating waves are described by a weighted sum of modes

$$u(x,t) = \sum_{n=1}^{\infty} \sin(n\pi x/L)\sin(n\pi x_s/L)\cos(\omega_n t)\exp[-(\omega_n\tau)^2/4]. \quad (3)$$

The displacement $u(x, t)$ for any position and time, ... ("stringogram") giving the displacement versus time at a receiver position x_r is $u(x_r, t)$. Alternatively, a ... of the displacement everywhere on the string at ... $u(x, t_0)$.

... a program to evaluate a synthetic seismogram ... sum. For simplicity, we use a string of length 1 m[1] ... speed 1 m/s, a source at $x_s = 0.2$ m and a receiver at ... To approximate the infinite sum, the program adds ... modes. The seismogram (Fig. A.8-1, *top*) is calculated ... steps, covering 1.25 s. This program is written in Fortran, a language that is especially suitable for scientific programming and is therefore commonly used in seismology (and thus in this book). The program could be also written in other languages, but the general points would still apply.

```
C SYNTHETIC SEISMOGRAM FOR HOMOGENEOUS STRING
C DISPLACEMENT U AS FUNCTION OF TIME T
C CALCULATED BY NORMAL MODE SUMMATION
      DIMENSION U(200)
      PI = 3.1415927
C
C PARAMETERS (NORMALLY WOULD COME FROM INPUT)
C STRING LENGTH (M)
      ALNGTH = 1.0
C VELOCITY (M/S)
      C = 1.0
C NUMBER OF MODES
      NMODE = 200
C SOURCE POSITION (M)
      XSRC = 0.2
C RECEIVER POSITION (M)
      XRCVR = 0.7
C SEISMOGRAM TIME DURATION (S)
      TDURAT = 1.25
C NUMBER TIME STEPS
      NTSTEP = 50
C TIME STEP (S)
      DT = TDURAT/NTSTEP
C SOURCE SHAPE TERM
      TAU = .02
C
C LIST PARAMETERS
      WRITE (6,3000)
3000  FORMAT('SYNTHETIC SEISMOGRAM FOR STRING')
      WRITE (6,3001) NMODE
3001  FORMAT('NUMBER OF MODES', I6)
      WRITE (6,3002) ALNGTH, C
3002  FORMAT ('LENGTH (M)' F7.3, 'VELOCITY,
     X (M/S)', F7.3)
      WRITE (6,3003) XSRC, XRCVR
3003  FORMAT ('POSITION (M): SOURCE', F7.3,
     X 'RECEIVER', F7.3)
      WRITE (6,3004) TDURAT, NTSTEP
3004  FORMAT ('SEISMOGRAM DURATION (S)', F7.3,
     X I6, 'TIME STEPS')
      WRITE (6,3005) TAU
3005  FORMAT ('SOURCE SHAPE TERM', F7.3)
C
C INITIALIZE DISPLACEMENT
        DO 5 I = 1, NTSTEP
      U(I) = 0.0
5     CONTINUE
C
C OUTER LOOP OVER MODES
      DO 10 N = 1, NMODE
        ANPIAL = N*PI/ALNGTH
```

[1] ... arbitrary values on a computer; we could also use 1 km or 1 ... physical 1 km string is another matter . . .

```
C SPACE TERMS: SOURCE AND RECEIVER
        SXS = SIN(ANPIAL*XSRC)
        SXR = SIN(ANPIAL*XRCVR)
C MODE FREQUENCY
        WN = N*PI*C/ALNGTH
C TIME INDEPENDENT TERMS
        DMP = (TAU*WN)**2
        SCALE = EXP(-DMP/4.)
        SPACE = SXS*SXR*SCALE
C
C INNER LOOP OVER TIME STEPS
        DO 15 J = 1, NTSTEP
          T = DT*(J - 1)
          CWT = COS(WN*T)
C COMPUTE DISPLACEMENT
          U(J) = U(J) + CWT*SPACE
15      CONTINUE
10    CONTINUE
C
C OUTPUT SEISMOGRAM FOR LATER PLOTTING
        WRITE (6, 3101)(U(J), J = 1, NTSTEP)
3101  FORMAT (7F10.4)
      STOP
      END
```

This example brings out several points:

- *Is the answer correct?* Two different types of error occur in scientific programs. First, the *program* may be wrong. In this case, the mathematical formulation correctly describes the physical problem, but the program incorrectly implements this formulation. This is the usual situation, in which "bugs" are identified and corrected. Second, the *formulation* may be wrong, so the program correctly implements an incorrect mathematical model. This could occur because of a mathematical error, like an attempt to sum a divergent series, or a physical error, such as an equation that does not correctly describe waves on a string. An incorrect formulation is particularly disturbing because it cannot be detected by checking the program. For example, Fig. A.8-2 shows two computer simulations for waves bending as they pass from one medium into another with higher velocities. Figure A.8-2 (*top*) uses the correct formulation of Snell's law (Section 2.5), whereas Fig. A.8-2 (*bottom*) looks equally convincing but is wrong because the equation which the program illustrates is incorrect.

Programmers check for both types of errors by choosing cases for which the results can be predicted analytically and comparing the results to those of the program. Several tests are easily done for the string. The wave following the shortest (direct) path appears at the expected time, 0.5 s (Fig. A.8-1, *bottom*), because the source and the receiver are 0.5 m apart. The next two arrivals, reflections from the ends of the string, also occur at the expected times. Moreover, these arrivals have polarities opposite that of the initial pulse, as should occur (Section 2.2.3) upon reflection at the string's fixed ends. The program can also be checked for different string lengths, speeds, and source and receiver positions. Similarly, in addition to synthetic seismograms, displacements along the string at fixed times could be computed. Such tests are important, because if the mathematical model is not appropriate for the physical situation, then time spent debugging, documenting, and optimizing the program is wasted.

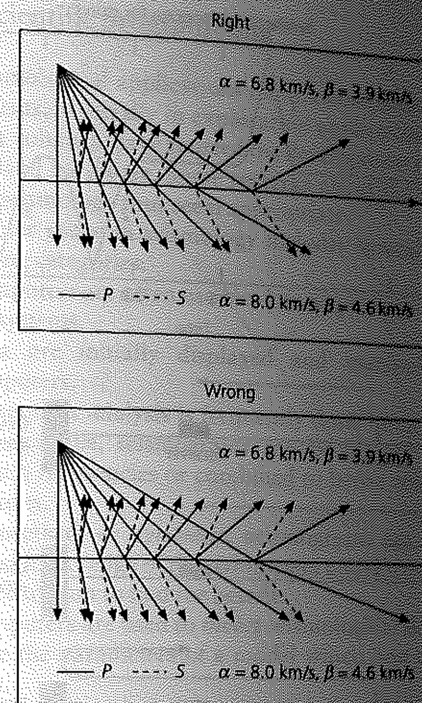

Fig. A.8-2 Demonstration of the danger that a program accurately computes an incorrect mathematical formulation. *Top*: A correct simulation of wave refraction using Snell's law, $\sin i_1/v_1 = \sin i_2/v_2$. *Bottom*: The same simulation using a wrong formula for Snell's law, $i_1/v_2 = i_2/v_2$.

- *The program is reasonably comprehensible.* Several features help clarify the program. The program's purpose and method are stated. Variable names somewhat resemble those in the equation: "SXS" is sin x_s, and so on. Comments identify the functions of portions of the program.

- *The program uses loops and arrays.* The seismogram is described by the array U(J), and its values at successive times are calculated by looping. Using an array, rather than discrete variables UT1, UT2, etc., makes the program clearer, closer to the mathematical formulation, and simplifies output. The loop structure also makes the program clearer and allows the number of time steps to be changed simply by changing the parameter NTSTEP. Similarly, the number of modes is easily changed.

- *The output is labeled.* The seismogram was placed in an output file for later plotting. The parameters used to compute the seismogram are included, so examination of the output


```
C OUTER LOOP OVER MODES
      DO 10 N = 1, NMODE

        terms for each mode
        that do not depend on time

C INNER LOOP OVER TIME STEPS
        DO 15 J = 1, NTSTEP

          terms that depend on time

C COMPUTE DISPLACEMENT
15        CONTINUE
10      CONTINUE
```

Fig. A.8-3 Structure of the loops for the string synthetic seismogram solution.

…how it was computed. This helps avoid the common …tion where, given a large collection of computer output, …are rerun because it is unclear what parameters were …ed. Moreover, subsequent "improved" versions of the pro…am can be checked to see whether they give the same results.

- *The program is somewhat efficient.* Some thought is generally put into *optimizing* scientific programs to make them run quickly. The program could find the displacement by looping over time and summing all the modes at each time step. However, consideration of the equation shows that three terms, …$(n\pi x_r/l)$, $\sin(n\pi x_s/l)$, and $\exp[-(\omega_n\tau)^2/4]$ are evaluated only once for each mode, whereas only $\cos(\omega_n t)$ is evaluated for each time step. It is thus more efficient to loop over the modes and evaluate each at all times (Fig. A.8-3). Because the outer (mode) loop is executed 200 times, whereas the inner (time) loop is executed $200 \times 50 = 10{,}000$ times, the inner loop should be as efficient as possible. The program would run more slowly if the loops were reversed. The difference, though not significant for this calculation, might be significant for much larger numbers of time steps and modes.

Further improvements could be made to fully optimize the program. Optimization is not an end in itself, because the programmer's time and the intelligibility of the program are also important. Programmers typically try to write reasonably optimized programs without making them impossible to understand and debug. Once fully tested, a program that will be used heavily may be worth further optimization if the computer time savings justify the effort required. There is no point in "getting the wrong answer as fast as possible."[2] Certain computers, such as those using parallel processors, may require specialized optimization.

A.8.2 Programming style

The style in which programs are written can make them easier to develop, debug and use. A few suggestions, though not absolute rules, may be useful.

- *Document the program.* Computer programs can be almost useless without adequate documentation. Stonehenge has been described as "the world's largest undocumented computer system."[3] Failure to document is often justified by the assumption that the program will not be used again. This rationalization is self-fulfilling, because even the author may find an undocumented program difficult to reuse once the details are forgotten.

Documentation should state the program's goals and method. The input and output variables, their units, and how they are defined should be listed. Implicit assumptions and restrictions are worth noting. Comments should identify major portions of the program and describe their functions.

Documentation is best written when writing a program because it can aid in debugging. Moreover, once a program is fully written, it is harder to remember how it works. Documentation included in the program is less prone to be lost than that written separately.

Finally, documentation helps scientists exchange programs and work in collaboration. This can be useful, except in the apocryphal cases of programmers writing gigantic undocumented programs to maximize their job security.

- *Use modular programming.* Large programs can generally be divided into smaller subroutines or functions, which can be used like the functions (e.g., sine, square root) supplied by many computer languages. Each subroutine can be tested separately and then used in various programs. Subroutines can handle applications that frequently recur, such as reading or plotting data or carrying out a mathematical operation. This approach saves the time needed to write and debug portions of a program similar to one already available. Moreover, the overall structure of a program containing a set of calls to subroutines is generally easier to understand, because many complexities are isolated into subroutines.

- *Make programs comprehensible.* It is helpful to be able to understand programs once written. Clear documentation and modular programming help. In addition, it should be easy to tell what portions will be executed under which circumstances. For this purpose, portions of a program should be executed sequentially, rather than jumping backwards and forwards within a program.

Similarly, the statements themselves can be written clearly. The use of mnemonic variable names and natural groupings of variables can help. For example, it is somewhat unclear that

X = 0.23873*A/(Y*Y*Y)

gives the average density X of a planet with mass A and radius Y, whereas

RHO = AMASS/((4.0/3.0)*PI*(RADIUS**3))

[2] Kernighan and Plauger (1978).

[3] Brooks (1975).