

gives 24-bit precision when the phantom bit is included). Doubles occupy 64 bits overall, with 1 bit for the sign, 10 bits for the exponent, and 53 bits for the fractional mantissa (for 54-bit precision). This means that the exponents and mantissas for doubles are not simply double those of floats, as we see in Table 1.5.1. (In addition, the IEEE standard also permits *extended precision* that goes beyond doubles, but this is all complicated enough without going into that right now.)

To see this scheme in action, look at the 32-bit float representing (1.2):

	s	e			f
Bit position	31	30	23	22	0

The sign bit s is in bit position 31, the biased exponent e is in bits 30–23, and the fractional part of the mantissa f is in bits 22–0. Since 8 bits are used to store the exponent e and since $2^8 = 256$, e has the range

$$0 \leq e \leq 255.$$

The values $e = 0$ and 255 are special cases. With bias = 127_{10} , the full exponent

$$p = e_{10} - 127,$$

and, as indicated in Table 1.5.1, for singles has the range

$$-126 \leq p \leq 127.$$

The mantissa f for singles is stored as the 23 bits in positions 22–0. For *normal numbers*, that is, numbers with $0 < e < 255$, f is the fractional part of the mantissa, and therefore the actual number represented by the 32 bits is

$$\text{Normal floating-point number} = (-1)^s \times 1.f \times 2^{e-127}.$$

Subnormal numbers have $e = 0$, $f \neq 0$. For these, f is the entire mantissa, so the actual number represented by these 32 bit is

$$\text{Subnormal numbers} = (-1)^s \times 0.f \times 2^{e-126}. \tag{1.4}$$

The 23 bits $m_{22}m_0$, which are used to store the mantissa of normal singles, correspond to the representation

$$\text{Mantissa} = 1.f = 1 + m_{22} \times 2^{-1} + m_{21} \times 2^{-2} + \dots + m_0 \times 2^{-23}, \tag{1.5}$$

with $0.f$ used for subnormal numbers. The special $e = 0$ representations used to store ± 0 and $\pm \infty$ are given in Table 1.5.1.

clarified 254

To see how this works in practice (Figure 1.7), the largest positive normal floating-point number possible for a 32-bit machine has the maximum value $e = 254$ (255 being reserved) and the maximum value for f :

1111 should be
1110

$$\begin{aligned} X_{\max} &= 01111\ 1110\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 111 \\ &= (0)(1111\ 1110)(1111\ 1111\ 1111\ 1111\ 1111\ 1111), \end{aligned} \tag{1.6}$$

Your **problem** is to use this series to calculate $\sin x$ for $x < 2\pi$ and $x > 2\pi$, with an absolute error in each case of less than 1 part in 10^8 . While an infinite series is exact in a mathematical sense, it is not a good algorithm because we must stop summing at some point. An algorithm would be the finite sum

$$\sin x \simeq \sum_{n=1}^N \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!} \quad (\text{algorithm}). \quad (1.15)$$

But how do we decide when to stop summing? (Do not even think of saying, “When the answer agrees with a table or with the built-in library function.”)

1.6.1 Numerical Summation (Method)

Never mind that the algorithm (1.15) indicates that we should calculate $(-1)^{n-1} x^{2n-1}$ and then divide it by $(2n-1)!$. This is not a good way to compute. On the one hand, both $(2n-1)!$ and x^{2n-1} can get very large and cause overflows, even though their quotient may not. On the other hand, powers and factorials are very expensive (time-consuming) to evaluate on the computer. Consequently, a better approach is to use a single multiplication to relate the next term in the series to the previous one:

$$\begin{aligned} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!} &= \frac{-x^2}{(2n-1)(2n-2)} \frac{(-1)^{n-2} x^{2n-3}}{(2n-3)!} \\ \Rightarrow \quad nth \text{ term} &= \frac{-x^2}{(2n-1)(2n-2)} \times (n-1)\text{th term}. \end{aligned} \quad (1.16)$$

While we want to ensure definite accuracy for $\sin x$, that is not so easy to do. What is easy to do is to assume that the error in the summation is approximately the last term summed (this assumes no round-off error, a subject we talk about in Chapter ??, “Errors & Uncertainties in Computations”). To obtain an absolute error of 1 part in 10^8 , we then stop the calculation when

$$\left| \frac{nth \text{ term}}{\text{sum}} \right| < 10^{-8}, \quad (1.17)$$

where “term” is the last term kept in the series (1.15) and “sum” is the accumulated sum of all the terms. In general, you are free to pick any tolerance level you desire, although if it is too close to, or smaller than, machine precision, your calculation may not be able to attain it. A pseudocode for performing the summation is

```
term = x, sum = x, eps = 10^(-8) // Initialize do
do term = -term*x*x/(2*n+1)/(2*n-2); // New wrt old
sum = sum + term // Add term
while abs(term/sum) > eps // Break iteration
end do
```

removed extra
parentheses

9.4 DYNAMIC FORM FOR ODES (THEORY)

A standard form for ODEs, which has found use both in numerical analysis [?, ?] and in classical dynamics [?, ?, ?], is to express ODEs of *any order* as N simultaneous first-order ODEs:

$$\begin{aligned} \frac{dy^{(0)}}{dt} &= f^{(0)}(t, y^{(i)}), \\ \frac{dy^{(1)}}{dt} &= f^{(1)}(t, y^{(i)}), \end{aligned} \tag{9.16}$$

$$\ddots \tag{9.17}$$

$$\frac{dy^{(N-1)}}{dt} = f^{(N-1)}(t, y^{(i)}), \tag{9.18}$$

where $y^{(i)}$ dependence in f is allowed but not any dependence on derivatives $dy^{(i)}/dt$. These equations can be expressed more compactly by use of the N -dimensional vectors (indicated here in **boldface**) \mathbf{y} and \mathbf{f} :

$$d\mathbf{y}(t)/dt = \mathbf{f}(t, \mathbf{y}), \tag{9.19}$$

$$\mathbf{y} = \begin{pmatrix} y^{(0)}(t) \\ y^{(1)}(t) \\ \vdots \\ y^{(N-1)}(t) \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} f^{(0)}(t, \mathbf{y}) \\ f^{(1)}(t, \mathbf{y}) \\ \vdots \\ f^{(N-1)}(t, \mathbf{y}) \end{pmatrix}.$$

The utility of such compact notation is that we can study the properties of the ODEs, as well as develop algorithms to solve them, by dealing with the single equation (9.19) without having to worry about the individual components. To see how this works, let us convert Newton's law

$$\frac{d^2x}{dt^2} = \frac{1}{m}F\left(t, x, \frac{dx}{dt}\right) \tag{9.20}$$

to standard dynamic form. The rule is that the RHS may not contain any explicit derivatives, although individual components of $y^{(i)}$ may represent derivatives. To pull this off, we define the position x as the dependent variable $y^{(0)}$ and the velocity dx/dt as the dependent variable $y^{(1)}$:

$$y^{(0)}(t) \stackrel{\text{def}}{=} x(t), \quad y^{(1)}(t) \stackrel{\text{def}}{=} \frac{dx}{dt} = \frac{dy^{(0)}(t)}{dt}. \tag{9.21}$$

The second-order ODE (9.20) now becomes two simultaneous first-order ODEs:

$$\frac{dy^{(0)}}{dt} = y^{(1)}(t), \quad \frac{dy^{(1)}}{dt} = \frac{1}{m}F(t, y^{(0)}, y^{(1)}). \tag{9.22}$$

x before dx/dt

y(t) was missing

was $y^{(2)}$

This expresses the acceleration [the second derivative in (9.20)] as the first derivative of the velocity [$y^{(1)}$]. These equations are now in the standard form (9.19), with the derivative or force function \mathbf{f} having the two components

$$f^{(0)} = y^{(1)}(t), \quad f^{(1)} = \frac{1}{m}F(t, y^{(0)}, y^{(1)}), \quad (9.23)$$

where F may be an explicit function of time as well as of position and velocity.

To be even more specific, applying these definitions to our spring problem (9.6), we obtain the coupled first-order equations

$$\frac{dy^{(0)}}{dt} = y^{(1)}(t), \quad \frac{dy^{(1)}}{dt} = \frac{1}{m} [F_{\text{ext}}(x, t) - ky^{(0)}(t)^{p-1}], \quad (9.24)$$

where $y^{(0)}(t)$ is the position of the mass at time t and $y^{(1)}(t)$ is its velocity. In the standard form, the components of the force function and the initial conditions are

$$\begin{aligned} f^{(0)}(t, \mathbf{y}) &= y^{(1)}(t), & f^{(1)}(t, \mathbf{y}) &= \frac{1}{m} [F_{\text{ext}}(x, t) - k(y^{(0)})^{p-1}], \\ y^{(0)}(0) &= x_0, & y^{(1)}(0) &= v_0. \end{aligned} \quad (9.25)$$

Breaking a second-order differential equation into two first-order ones is not just an arcane mathematical maneuver. In classical dynamics it occurs when transforming the single Newtonian equation of motion involving position and acceleration (9.1), into two *Hamiltonian* equations involving position and momentum:

$$\frac{dp_i}{dt} = F_i, \quad m \frac{dy_i}{dt} = p_i. \quad (9.26)$$

9.5 ODE ALGORITHMS

The classic way to solve a differential equation is to start with the known initial value of the dependent variable, $y_0 \equiv y(t = 0)$, and then use the derivative function $f(t, y)$ to find an approximate value for y at a small step $\Delta t = h$ forward in time; that is, $y(t = h) \equiv y_1$. Once you can do that, you can solve the ODE for all t values by just continuing stepping to larger times one small h at a time (Figure 9.3).² Error is always a concern when integrating differential equations because derivatives require small differences, and small differences are prone to subtractive cancellations and round-off error accumulation. In addition, because our stepping procedure for solving the differential equation is a continuous extrapolation of the initial conditions, with each step building on a previous extrapolation, this is somewhat like a castle built on sand; in contrast to interpolation, there are no tabulated values on which to anchor your solution.

²To avoid confusion, notice that $y^{(n)}$ is the n th component of the y vector, while y_n is the value of y after n time steps. (Yes, there is a price to pay for elegance in notation.)

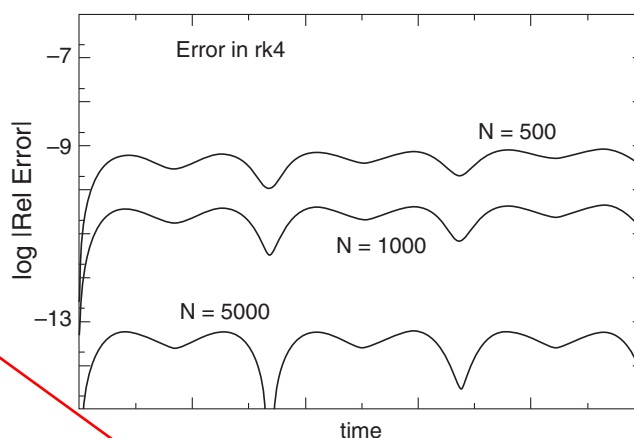


Figure 9.5 The log of the relative error (number of places of precision) obtained with `rk4` using a differing number N of steps over the same time interval.

5. Make a table of comparisons similar to Table 9.1. There we compare `rk4` and `rk45` for the two equations

$$2yy'' + y^2 - y'^2 = 0, \quad (9.41)$$

$$y'' + 6y^5 = 0, \quad (9.42)$$

with initial conditions $([y(0), y'(0)] = (1, 1))$. Although nonlinear, (9.41) does have the analytic solution $y(t) = 1 + \sin t$. But *be warned*, the `rk` procedures may be inaccurate for this equation if integrated through $y(t) = 0$; the two terms in the equation proportional to y vanish there, leaving $y'^2 = 0$, which is problematic. A different method, such as a predictor-corrector, might then be better.⁴ Equation (9.42) corresponds to our standard potential (9.4), with $p = 6$. Although we have not tuned `rk45`, the table shows that by setting its tolerance parameter to a small enough number, `rk45` will obtain better precision than `rk4` (Figure 9.5) but that it requires ~ 10 times more floating-point operations and takes ~ 5 times longer.

9.6 SOLUTION FOR NONLINEAR OSCILLATIONS (ASSESSMENT)

Use your `rk4` program to study anharmonic oscillations by trying powers in the range $p = 2$ – 12 or anharmonic strengths in the range $0 \leq \alpha x \leq 2$. Do *not* include any explicit time-dependent forces yet. Note that for large values of p you may need to decrease the step size h from the value used for the harmonic oscillator because the forces and accelerations get large near the turning points.

⁴We thank Guenter Schneider for pointing this out to us.

the Schrödinger equation (9.49) becomes

$$\frac{d^2\psi(x)}{dx^2} + \left(\frac{2m}{\hbar^2}V_0 - \kappa^2 \right) \psi(x) = 0, \quad \text{for } |x| \leq a, \quad (9.52)$$

$$\frac{d^2\psi(x)}{dx^2} - \kappa^2\psi(x) = 0, \quad \text{for } |x| > a. \quad (9.53)$$

To evaluate the ratio of constants here, we insert c^2 , the speed of light squared, into both the numerator and the denominator [?, Appendix A.1]:

$$\frac{2m}{\hbar^2} = \frac{2mc^2}{(\hbar c)^2} \simeq \frac{2 \times 940 \text{ MeV}}{(197.32 \text{ MeV fm})^2} = 0.0483 \text{ MeV}^{-1} \text{ fm}^{-2}. \quad (9.54)$$

9.11 COMBINED ALGORITHMS: EIGENVALUES VIA ODE SOLVER PLUS SEARCH

The solution of the eigenvalue problem combines the numerical solution of the ordinary differential equation (9.49) with a trial-and-error search for a wave function that satisfies the boundary conditions (9.50). This is done in several steps:⁸

1. Start on the very far *left* at $x = -X_{\max} \simeq -\infty$, where $X_{\max} \gg a$. Assume that the wave function there satisfies the left-hand boundary condition:

$$\psi_L(x = -X_{\max}) = e^{+\kappa x} = e^{-\kappa X_{\max}}.$$

2. Use your favorite ODE solver to step $\psi_L(x)$ in toward the origin (to the right) from $x = -X_{\max}$ until you reach the *matching radius* x_{match} . The exact value of this matching radius is not important, and our final solution should be independent of it. On the left in Figure 9.7, we show a sample solution with $x_{\text{match}} = -a$; that is, we match at the left edge of the potential well. In the middle and on the right in Figure 9.7 we see some guesses that do not match.

3. Start on the very far *right*, that is, at $x = +X_{\max} \simeq +\infty$, with a wave function that satisfies the right-hand boundary condition:

$$\psi_R(x = +X_{\max}) = e^{-\kappa x} = e^{-\kappa X_{\max}}.$$

4. Use your favorite ODE solver (e.g., `rk4`) to step $\psi_R(x)$ in toward the origin (to the left) from $x = +X_{\max}$ until you reach the *matching radius* x_{match} . This means that we have stepped through the potential well (Figure 9.7).
5. In order for probability and current to be continuous at $x = x_{\text{match}}$, $\psi(x)$ and $\psi'(x)$ must be continuous there. Requiring the ratio $\psi'(x)/\psi(x)$, called the

⁸The procedure outlined here is for a general potential that falls off gradually. For a square well with sharp cutoffs the asymptotic solution is valid right up unto the well walls, whereas for a Coulomb potential with a very slow falloff a modified asymptotic form is required.

Added footnote.

Because (18.2) is second-order in time, a second initial condition (beyond initial displacement) is needed to determine the solution. We interpret the “gentleness” of the pluck to mean the string is released from rest:

$$\frac{\partial y}{\partial t}(x, t = 0) = 0, \quad (\text{initial condition 2}). \quad (18.4)$$

The boundary conditions have both ends of the string tied down for all times:

$$y(0, t) \equiv 0, \quad y(L, t) \equiv 0, \quad (\text{boundary conditions}). \quad (18.5)$$

18.2.1 Solution via Normal-Mode Expansion

The analytic solution to (18.2) is obtained via the familiar separation-of-variables technique. We assume that the solution is the product of a function of space and a function of time:

$$y(x, t) = X(x)T(t). \quad (18.6)$$

We substitute (18.6) into (18.2), divide by $y(x, t)$, and are left with an equation that has a solution only if there are solutions to the two ODEs:

dt -> dx

$$\frac{d^2T(t)}{dt^2} + \omega^2 T(t) = 0, \quad \frac{d^2X(x)}{dx^2} + k^2 X(x) = 0, \quad k \stackrel{\text{def}}{=} \frac{\omega}{c}. \quad (18.7)$$

The angular frequency ω and the wave vector k are determined by demanding that the solutions satisfy the boundary conditions. Specifically, the string being attached at both ends demands

$$X(x = 0, t) = X(x = l, t) = 0 \quad (18.8)$$

$$\Rightarrow X_n(x) = A_n \sin k_n x, \quad k_n = \frac{\pi(n+1)}{L}, \quad n = 0, 1, \dots \quad (18.9)$$

The time solution is

$$T_n(t) = C_n \sin \omega_n t + D_n \cos \omega_n t, \quad \omega_n = nck_0 = n \frac{2\pi c}{L}, \quad (18.10)$$

where the frequency of this n th normal mode is also fixed. In fact, it is the single frequency of oscillation that defines a normal mode. The initial condition (18.3) of zero velocity, $\partial y / \partial t(t = 0) = 0$, requires the C_n values in (18.10) to be zero. Putting the pieces together, the normal-mode solutions are

$$y_n(x, t) = \sin k_n x \cos \omega_n t, \quad n = 0, 1, \dots \quad (18.11)$$

Since the wave equation (18.2) is linear in y , the principle of linear superposition holds and the most general solution for waves on a string with fixed ends can be written as the sum of normal modes:

$$y(x, t) = \sum_{n=0}^{\infty} B_n \sin k_n x \cos \omega_n t. \quad (18.12)$$

(Yet we will lose linear superposition once we include nonlinear terms in the wave equation.) The Fourier coefficient B_n is determined by the second initial condition (18.3), which describes how the wave is plucked:

$$y(x, t = 0) = \sum_n^{\infty} B_n \sin nk_0 x. \quad (18.13)$$

6.25 -> 12.5

Multiply both sides by $\sin mk_0 x$, substitute the value of $y(x, 0)$ from (18.3), and integrate from 0 to l to obtain

$$B_m = 12.5 \frac{\sin(0.8m\pi)}{m^2\pi^2}. \quad (18.14)$$

You will be asked to compare the Fourier series (18.12) to our numerical solution. While it is in the nature of the approximation that the precision of the numerical solution depends on the choice of step sizes, it is also revealing to realize that the precision of the analytic solution depends on summing an infinite number of terms, which can be done only approximately.

18.2.2 Algorithm: Time-Stepping

As with Laplace's equation and the heat equation, we look for a solution $y(x, t)$ only for discrete values of the independent variables x and t on a grid (Figure 18.2):

$$x = i\Delta x, \quad i = 1, \dots, N_x, \quad t = j\Delta t, \quad j = 1, \dots, N_t, \quad (18.15)$$

$$y(x, t) = y(i\Delta x, j\Delta t) \stackrel{\text{def}}{=} y_{i,j}. \quad (18.16)$$

In contrast to Laplace's equation where the grid was in two space dimensions, the grid in Figure 18.2 is in both space and time. That being the case, moving across a row corresponds to increasing x values along the string for a fixed time, while moving down a column corresponds to increasing time steps for a fixed position. Even though the grid in Figure 18.2 may be square, we cannot use a relaxation technique for the solution because we do not know the solution on all four sides. The boundary conditions determine the solution along the right and left sides, while the initial time condition determines the solution along the top.

As with the Laplace equation, we use the central-difference approximation to *discretize* the wave equation into a difference equation. First we express the second derivatives in terms of finite differences:

$$\frac{\partial^2 y}{\partial t^2} \simeq \frac{y_{i,j+1} + y_{i,j-1} - 2y_{i,j}}{(\Delta t)^2}, \quad \frac{\partial^2 y}{\partial x^2} \simeq \frac{y_{i+1,j} + y_{i-1,j} - 2y_{i,j}}{(\Delta x)^2}. \quad (18.17)$$

Substituting (18.17) in the wave equation (18.2) yields the difference equation

$$\frac{y_{i,j+1} + y_{i,j-1} - 2y_{i,j}}{c^2(\Delta t)^2} = \frac{y_{i+1,j} + y_{i-1,j} - 2y_{i,j}}{(\Delta x)^2}. \quad (18.18)$$

to obtain high precision, you may want to store the solution only for every fifth or tenth time.

Initializing the recurrence relation is a bit tricky because it requires displacements from two earlier times, whereas the initial conditions are for only one time. Nonetheless, the rest condition (18.3) when combined with the *central-difference* approximation lets us extrapolate to negative time:

Delta t not 0

$$\frac{\partial y}{\partial t}(x, 0) \simeq \frac{y(x, \Delta t) - y(x, -\Delta t)}{2\Delta t} = 0, \Rightarrow y_{i,0} = y_{i,2}. \quad (18.20)$$

Here we take the initial time as $j = 1$, and so $j = 0$ corresponds to $t = -\Delta t$. Substituting this relation into (18.19) yields for the initial step

2 in denominator

$$y_{i,2} = y_{i,1} + \frac{c^2}{2c'^2} [y_{i+1,1} + y_{i-1,1} - 2y_{i,1}] \quad (\text{for } j = 2 \text{ only}). \quad (18.21)$$

Equation (18.21) uses the solution throughout all space at the initial time $t = 0$ to propagate (leapfrog) it forward to a time Δt . Subsequent time steps use (18.19) and are continued for as long as you like.

As is also true with the heat equation, the success of the numerical method depends on the relative sizes of the time and space steps. If we apply a von Neumann stability analysis to this problem by substituting $y_{m,j} = \xi^j \exp(ikm \Delta x)$, as we did in §17.17.3, a complicated equation results. Nonetheless, [Press] shows that the difference-equation solution will be stable for the general class of transport equations if

$$c \leq c' = \Delta x / \Delta t \quad (\text{Courant condition}). \quad (18.22)$$

Equation (18.22) means that the solution gets better with smaller *time* steps but gets worse for smaller space steps (unless you simultaneously make the time step smaller). Having different sensitivities to the time and space steps may appear surprising because the wave equation (18.2) is symmetric in x and t , yet the symmetry is broken by the nonsymmetric initial and boundary conditions.

Exercise: Figure out a procedure for solving for the wave equation for all times in just one step. Estimate how much memory would be required. ■

Exercise: Try to figure out a procedure for solving for the wave motion with a relaxation technique. What would you take as your initial guess, and how would you know when the procedure has converged? ■

We simplify the algorithm and make its stability analysis simpler by normalizing the electric fields to have the same dimensions as the magnetic fields,

inverted fraction

$$\tilde{E} = \sqrt{\frac{\epsilon_0}{\mu_0}} E. \quad (18.68)$$

The algorithm (18.64) and (18.65) now becomes

$$\tilde{E}_x^{k,n+1/2} = \tilde{E}_x^{k,n-1/2} + \beta \left(H_y^{k-1/2,n} - H_y^{k+1/2,n} \right), \quad (18.69)$$

$$H_y^{k+1/2,n+1} = H_y^{k+1/2,n} + \beta \left(\tilde{E}_x^{k,n+1/2} - \tilde{E}_x^{k+1,n+1/2} \right), \quad (18.70)$$

$$\beta = \frac{c}{\Delta z / \Delta t}, \quad c = \frac{1}{\sqrt{\epsilon_0 \mu_0}}. \quad (18.71)$$

Here c is the speed of light in a vacuum and β is the ratio of the speed of light to grid velocity $\Delta z / \Delta t$.

The space step Δz and the time step Δt must be chosen so that the algorithm is stable. The scales of the space and time dimensions are set by the wavelength and frequency, respectively, of the propagating wave. As a minimum, we want at least 10 grid points to fall within a wavelength:

$$\Delta z \leq \frac{\lambda}{10}. \quad (18.72)$$

The time step is then determined by the Courant stability condition [?, ?] to be

$$\beta = \frac{c}{\Delta z / \Delta t} \leq \frac{1}{2}. \quad (18.73)$$

As we have seen before, (18.73) implies that making the time step smaller improves precision and maintains stability, but making the space step smaller must be accompanied by a simultaneous decrease in the time step in order to maintain stability (you should check this).

18.11.1 Implementation

In Listing 18.3 we provide a simple implementation of the FDTD algorithm for a z lattice of 200 sites. The initial condition corresponds to a Gaussian pulse in time for the E field, located at the midpoint of the z lattice (in Δt and Δz units):

$$E_x(z = 100, t) = \exp \left[\frac{1}{2} \left(\frac{40 - t}{12} \right)^2 \right]. \quad (18.74)$$

DX FIXES

The files given on the CD were ones we used to run OpenDX applications on our server. In order to get them to run on your system, you may need to do some things:

- Change the name of the X server to your local X server.
- The script `Custom_linear_plot2.net` calls a data file that needs to be supplied. You can create it or use the supplied `simple_linear_data.dat`.
- The script `Laplace.net` produces only one of the three images unless you supply a `topo-map.ps` file.

Follow link to updated DX data files.