18
Molecular Dynamics Simulations

You may have seen in introductory chemistry or physics classes that the ideal gas law can be derived from first principles when the interactions of the molecules with each other are ignored, and only reflections off the walls of the surrounding box are considered. We extend that model so that we can solve for the motion of every molecule in a box interacting with every other molecule in the box (but not with the walls). While our example is a simple one, molecular dynamics is of key importance in many fields, including material science and biology.

Your *problem* is to determine whether a collection of argon molecules placed in a box will form an ordered structure at low temperature.

18.1
Molecular Dynamics (Theory)

Molecular dynamics (MD) is a powerful simulation technique for studying the physical and chemical properties of solids, liquids, amorphous materials, and biological molecules. Although we know that quantum mechanics is the proper theory for molecular interactions, MD uses Newton's laws as the basis of the technique and focuses on bulk properties, which do not depend much on small-r behaviors. In 1985, Car and Parrinello showed how MD can be extended to include quantum mechanics by applying density functional theory to calculate the force (Car and Parrinello, 1985). This technique, known as *quantum MD*, is an active area of research but is beyond the realm of this chapter.[1)] For those with more interest in the subject, there are full texts (Allan and Tildesley, 1987; Rapaport, 1995; Hockney, 1988) on MD and good discussions (Gould *et al.*, 2006; Thijssen, 1999; Fosdick *et al.*, 1996), as well as primers (Ercolessi, 1997) and codes, (Nelson *et al.*, 1996; Refson, 2000; Anderson *et al.*, 2008) available online.

MD's solution of Newton's laws is conceptually simple, yet when applied to a very large number of particles becomes the "high school physics problem from hell." Some approximations must be made in order not to have to solve

1) We thank Satoru S. Kano for pointing this out to us.

Computational Physics, 3rd edition. Rubin H. Landau, Manuel J. Páez, Cristian C. Bordeianu.
©2015 WILEY-VCH Verlag GmbH & Co. KGaA. Published 2015 by WILEY-VCH Verlag GmbH & Co. KGaA.

the 10^{23}–10^{25} equations of motion describing a realistic sample, but instead to limit the problem to $\sim 10^6$ particles for protein simulations, and $\sim 10^8$ particles for materials simulations. If we have some success, then it is a good bet that the model will improve if we incorporate more particles or more quantum mechanics, something that becomes easier as computing power continues to increase.

In a number of ways, MD simulations are similar to the thermal Monte Carlo simulations we studied in Chapter 17. Both typically involve a large number N of interacting particles that start out in some set configuration and then equilibrate into some dynamic state on the computer. However, in MD we have what statistical mechanics calls a *microcanonical ensemble* in which the energy E and volume V of the N particles are fixed. We then use Newton's laws to generate the dynamics of the system. In contrast, Monte Carlo simulations do not start with first principles, but instead, incorporate an element of chance and have the system in contact with a heat bath at a fixed temperature rather than keeping the energy E fixed. This is called a *canonical ensemble*.

Because a system of molecules in an MD simulation is dynamic, the velocities and positions of the molecules change continuously. After the simulation has run long enough to stabilize, we will compute time averages of the dynamic quantities in order to deduce the thermodynamic properties. We apply Newton's laws with the assumption that the net force on each molecule is the sum of the two-body forces with all other $(N-1)$ molecules:

$$m\frac{\mathrm{d}^2\boldsymbol{r}_i}{\mathrm{d}t^2} = \boldsymbol{F}_i(\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{N-1}) \,, \tag{18.1}$$

$$m\frac{\mathrm{d}^2\boldsymbol{r}_i}{\mathrm{d}t^2} = \sum_{i<j=0}^{N-1} \boldsymbol{f}_{ij} \,, \quad i = 0, \ldots, (N-1) \,. \tag{18.2}$$

In writing these equations, we have ignored the fact that the force between argon atoms really arises from the particle–particle interactions of the 18 electrons and the nucleus that constitute each atom (Figure 18.1). Although it may be possible to ignore this internal structure when deducing the long-range properties of inert elements, it matters for systems such as polyatomic molecules that display rotational, vibrational, and electronic degrees of freedom as the temperature is raised.[2)]

The force on molecule i derives from the sum of molecule–molecule potentials:

$$\boldsymbol{F}_i(\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{N-1}) = -\boldsymbol{\nabla}_{\boldsymbol{r}_i} U(\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{N-1}) \,, \tag{18.3}$$

$$U(\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{N-1}) = \sum_{i<j} u(r_{ij}) = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} u(r_{ij}) \tag{18.4}$$

$$\Rightarrow \quad \boldsymbol{f}_{ij} = -\frac{\mathrm{d}u(r_{ij})}{\mathrm{d}r_{ij}} \left(\frac{x_i - x_j}{r_{ij}}\hat{\boldsymbol{e}}_x + \frac{y_i - y_j}{r_{ij}}\hat{\boldsymbol{e}}_y + \frac{z_i - z_j}{r_{ij}}\hat{\boldsymbol{e}}_z \right) . \tag{18.5}$$

2) We thank Saturo Kano for clarifying this point.

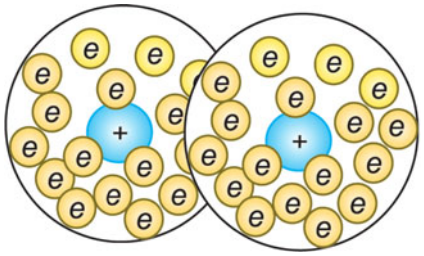

Figure 18.1 The molecule–molecule effective interaction arises from the many-body interaction of the electrons and nucleus in one molecule (circle) with the electrons and nucleus in another molecule (other circle). Note, the size of the nucleus at the center of each molecule is highly exaggerated, and real electrons have no size.

Here $r_{ij} = |\boldsymbol{r}_i - \boldsymbol{r}_j| = r_{ji}$ is the distance between the centers of molecules i and j, and the limits on the sums are such that no interaction is counted twice. Because we have assumed a *conservative* potential, the total energy of the system, that is, the potential plus kinetic energies summed over all particles, should be conserved over time. Nonetheless, in a practical computation we "cut the potential off" (assume $u(r_{ij}) = 0$) when the molecules are far apart. Because the derivative of the potential produces an infinite force at this cutoff point, energy will no longer be precisely conserved. Yet because the cutoff radius is large, the cutoff occurs only when the forces are minuscule, and so the violation of energy conservation should be small relative to approximation and round-off errors.

In a first-principles calculation, the potential between any two argon atoms arises from the sum over approximately 1000 electron–electron and electron–nucleus Coulomb interactions. A more practical calculation would derive an effective potential based on a form of many-body theory, such as Hartree–Fock or density functional theory. Our approach is simpler yet. We use the Lennard–Jones potential,

$$u(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] , \tag{18.6}$$

$$\boldsymbol{f}(r) = -\frac{\mathrm{d}u}{\mathrm{d}r}\frac{\boldsymbol{r}}{r} = \frac{48\epsilon}{r^2}\left[\left(\frac{\sigma}{r}\right)^{12} - \frac{1}{2}\left(\frac{\sigma}{r}\right)^{6} \right] \boldsymbol{r} . \tag{18.7}$$

Here the parameter ϵ governs the strength of the interaction, and σ determines the length scale. Both are deduced by fits to data, which is why this potential is called a "phenomenological" potential.

Some typical values for the parameters and scales for the variables are given in Table 18.1. In order to make the program simpler and to avoid under- and overflows, it is helpful to measure all variables in the natural units formed by these constants. The interparticle potential and force then take the forms

$$u(r) = 4\left[\frac{1}{r^{12}} - \frac{1}{r^6}\right] , \quad f(r) = \frac{48}{r}\left[\frac{1}{r^{12}} - \frac{1}{2r^6}\right] . \tag{18.8}$$

Table 18.1 Parameter and scales for the Lennard–Jones potential.

Quantity	Mass	Length	Energy	Time	Temperature
Unit	m	σ	ϵ	$\sqrt{m\sigma^2/\epsilon}$	ϵ/k_B
Value	6.7×10^{-26} kg	3.4×10^{-10} m	1.65×10^{-21} J	4.5×10^{-12} s	119 K

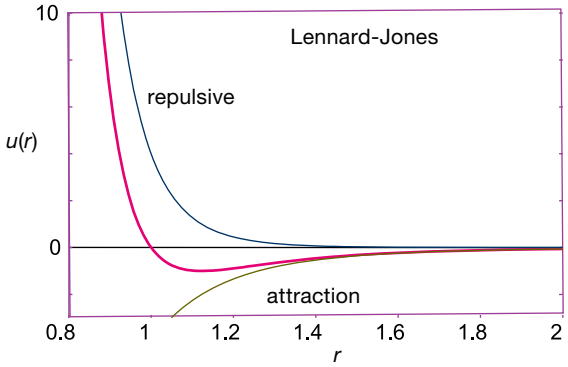

Figure 18.2 The Lennard–Jones effective potential used in many MD simulations. Note the sign change at $r = 1$ and the minimum at $r \simeq 1.1225$ (natural units). Also note that because the r-axis does not extend to $r = 0$, the infinitely high central repulsion is not shown.

The Lennard–Jones potential is seen in Figure 18.2 to be the sum of a long-range attractive interaction $\propto 1/r^6$ and a short-range repulsive one $\propto 1/r^{12}$. The change from repulsion to attraction occurs at $r = \sigma$. The minimum of the potential occurs at $r = 2^{1/6}\sigma = 1.1225\sigma$, which would be the atom–atom spacing in a solid bound by this potential. The repulsive $1/r^{12}$ term in the Lennard–Jones potential (18.6) arises when the electron clouds from two atoms overlap, in which case the Coulomb interaction and the Pauli exclusion principle force the electrons apart. The $1/r^{12}$ term dominates at short distances and makes atoms behave like hard spheres. The precise value of 12 is not of theoretical significance (although it being large is) and was probably chosen because it is 2×6.

The $1/r^6$ term that dominates at large distances models the weak *van der Waals* induced dipole–dipole attraction between two molecules. The attraction arises from fluctuations in which at some instant in time a molecule on the right tends to be more positive on the left side, like a dipole $\Leftarrow$. This in turn attracts the negative charge in a molecule on its left, thereby inducing a dipole $\Leftarrow$ in it. As long as the molecules stay close to each other, the polarities continue to fluctuate in synchronization $\Leftarrow\Leftarrow$ so that the attraction is maintained. The resultant dipole–dipole attraction behaves like $1/r^6$, and although much weaker than a Coulomb force, it is responsible for the binding of neutral, inert elements, such as argon for which the Coulomb force vanishes.

18.1.1 Connection to Thermodynamic Variables

We assume that the number of particles is large enough to use statistical mechanics to relate the results of our simulation to the thermodynamic quantities. The simulation is valid for any number of particles, but the use of statistics requires large numbers. The equipartition theorem tells us that, on the average, for molecules in thermal equilibrium at temperature T, each degree of freedom has an energy $k_{\mathrm{B}}T/2$ associated with it, where $k_{\mathrm{B}} = 1.38 \times 10^{-23}$ J/K is Boltzmann's constant. A simulation provides the kinetic energy of translation[3]:

$$\mathrm{KE} = \frac{1}{2}\left\langle \sum_{i=0}^{N-1} v_i^2 \right\rangle . \tag{18.9}$$

The time average of KE (three degrees of freedom) is related to temperature by

$$\langle \mathrm{KE} \rangle = N\frac{3}{2}k_{\mathrm{B}}T \quad \Rightarrow \quad T = \frac{2\langle \mathrm{KE} \rangle}{3k_{\mathrm{B}}N} . \tag{18.10}$$

The system's pressure P is determined by a version of the *Virial theorem*,

$$PV = Nk_{\mathrm{B}}T + \frac{w}{3} , \quad w = \left\langle \sum_{i<j}^{N-1} \boldsymbol{r}_{ij} \cdot \boldsymbol{f}_{ij} \right\rangle , \tag{18.11}$$

where the Virial w is a weighted average of the forces. Note that because ideal gases have no intermolecular forces, their Virial vanishes and we have the ideal gas law. The pressure is thus

$$P = \frac{\rho}{3N}(2\langle \mathrm{KE} \rangle + w) , \tag{18.12}$$

where $\rho = N/V$ is the density of the particles.

18.1.2 Setting Initial Velocities

Although we start the system off with a velocity distribution characteristic of a definite temperature, this is not a true temperature of the system because the system is not in equilibrium initially, and there will a redistribution of energy between KE and PE (Thijssen, 1999). Note that this initial randomization is the only place where chance enters into our MD simulation, and it is there to speed the simulation along. Indeed, in Figure 18.3 we show results of simulations in which the molecules are initially at rest and equally spaced. Once started, the time evolution is determined by Newton's laws, in contrast to Monte Carlo simulations which are inherently stochastic. We produce a Gaussian (Maxwellian) velocity distribution with the methods discussed in Chapter 4. In our sample code, we take the average $1/12\sum_{i=1}^{12} r_i$ of uniform random numbers $0 \le r_i \le 1$ to produce a Gaussian distribution with mean $\langle r \rangle = 0.5$. We then subtract this mean value to obtain a distribution about 0.

3) Unless the temperature is very high, argon atoms, being inert spheres, have no rotational energy.

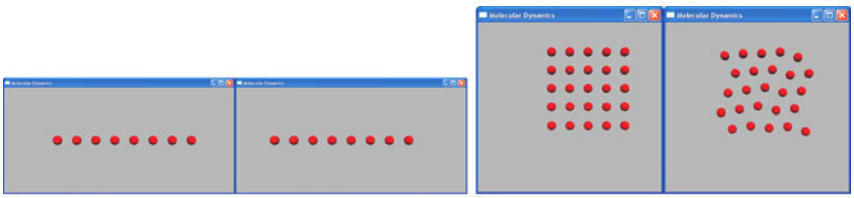

Figure 18.3 (a) Two frames from an animation showing the results of a 1D MD simulation that starts with uniformly spaced atoms. Note the unequal spacing resulting from an image atom moving in from the left after an atom left from the right. (b) Two frames from the animation of a 2D MD simulation showing the initial and an equilibrated state. Note how the atoms start off in a simple cubic arrangement but then equilibrate to a face-centered-cubic lattice. In both the cases, the atoms remain confined as a result of the interatomic forces.

18.1.3 Periodic Boundary Conditions and Potential Cutoff

It is easy to believe that a simulation of 10^{23} molecules should predict bulk properties well, but with typical MD simulations employing only 10^3–10^6 particles, one must be clever to make less seem like more. Furthermore, because computers are finite, the molecules in the simulation are constrained to lie within a finite box, which inevitably introduces artificial *surface effects* arising from the walls. Surface effects are particularly significant when the number of particles is small because then a large fraction of the molecules reside near the walls. For example, if 1000 particles are arranged in a $10 \times 10 \times 10 \times 10$ cube, there are $10^3 - 8^3 = 488$ particles one unit from the surface, that is, 49% of the molecules, while for 10^6 particles this fraction falls to 6%.

The imposition of *periodic boundary conditions* (PBCs) strives to minimize the shortcomings of both the small numbers of particles and of artificial boundaries. Although we limit our simulation to an $L_x \times L_y \times L_z$ box, we imagine this box being replicated to infinity in all directions (Figure 18.4). Accordingly, after each time-integration step we examine the position of each particle and check if it has left the simulation region. If it has, then we bring an *image* of the particle back through the opposite boundary (Figure 18.4):

$$x \quad \Rightarrow \quad \begin{cases} x + L_x\,, & \text{if} \quad x \le 0\,, \\ x - L_x\,, & \text{if} \quad x > L_x\,. \end{cases} \tag{18.13}$$

Consequently, each box looks the same and has continuous properties at the edges. As shown by the one-headed arrows in Figure 18.4, if a particle exits the simulation volume, its image enters from the other side, and so balance is maintained.

In principle, a molecule interacts with all others molecules and their images, so despite the fact that there is a finite number of atoms in the interaction volume, there is an effective infinite number of interactions (Ercolessi, 1997). Nonetheless, because the Lennard–Jones potential falls off so rapidly for large r, $V(r = 3\sigma) \simeq$

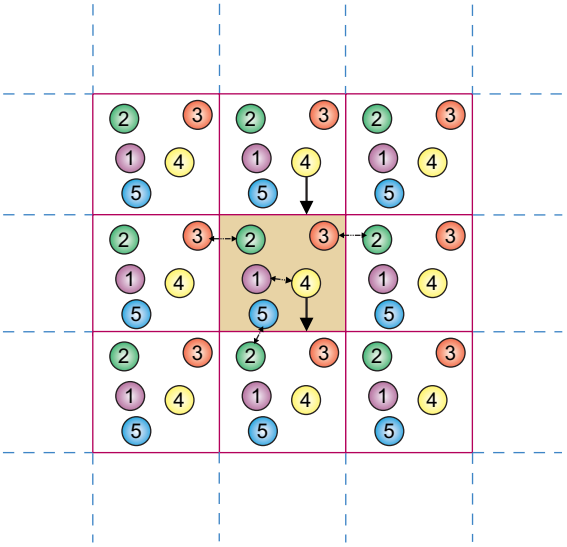

Figure 18.4 The infinite space generated by imposing periodic boundary conditions on the particles within the simulation volume (shaded box). The two-headed arrows indicate how a particle interacts with the nearest version of another particle, be that within the simulation volume or an image. The vertical arrows indicate how the image of particle 4 enters when the actual particle 4 exits.

$V(1.13\sigma)/200$, far-off molecules do not contribute significantly to the motion of a molecule, and we pick a value $r_{\text{cut}} \simeq 2.5\sigma$ beyond which we ignore the effect of the potential:

$$u(r) = \begin{cases} 4(r^{-12} - r^{-6}) , & \text{for} \quad r < r_{\text{cut}} , \\ 0 , & \text{for} \quad r > r_{\text{cut}} . \end{cases} \tag{18.14}$$

Accordingly, if the simulation region is large enough for $u(r > L_i/2) \simeq 0$, an atom interacts with only the *nearest image* of another atom.

As already indicated, a shortcoming with the cutoff potential (18.14) is that because the derivative $\mathrm{d}u/\,\mathrm{d}r$ is singular at $r = r_{\text{cut}}$, the potential is no longer conservative and thus energy conservation is no longer ensured. However, because the forces are already very small at r_{cut}, the violation will also be very small.

18.2 Verlet and Velocity–Verlet Algorithms

A realistic MD simulation may require integration of the 3D equations of motion for 10^{10} time steps for each of 10^3–10^6 particles. Although we could use our standard `rk4` ODE solver for this, time is saved by using a simple rule embedded in the program. The Verlet algorithm uses the central-difference approximation (Chapter 5) for the second derivative to advance the solutions by a single time step h for

all N particles simultaneously:

$$\boldsymbol{F}_i[\boldsymbol{r}(t), t] = \frac{\mathrm{d}^2 \boldsymbol{r}_i}{\mathrm{d}t^2} \simeq \frac{\boldsymbol{r}_i(t+h) + \boldsymbol{r}_i(t-h) - 2\boldsymbol{r}_i(t)}{h^2} \tag{18.15}$$

$$\Rightarrow \quad \boldsymbol{r}_i(t+h) \simeq 2\boldsymbol{r}_i(t) - \boldsymbol{r}_i(t-h) + h^2 \boldsymbol{F}_i(t) + O(h^4) \,, \tag{18.16}$$

where we have set $m = 1$. (Improved algorithms may vary the time step depending upon the speed of the particle.) Note that although the atom–atom force does not have an explicit time dependence, we include a t dependence in it as a way of indicating its dependence upon the atoms' positions at a particular time. Because this is really an implicit time dependence, energy remains conserved.

Part of the efficiency of the Verlet algorithm (18.16) is that it solves for the position of each particle without requiring a separate solution for the particle's velocity. However, once we have deduced the position for various times, we can use the central-difference approximation for the first derivative of $\boldsymbol{r}_i$ to obtain the velocity:

$$\boldsymbol{v}_i(t) = \frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t} \simeq \frac{\boldsymbol{r}_i(t+h) - \boldsymbol{r}_i(t-h)}{2h} + O(h^2) \,. \tag{18.17}$$

Finally, note that because the Verlet algorithm needs $\boldsymbol{r}$ from two previous steps, it is not self-starting and so we start it with the forward difference

$$\boldsymbol{r}(t = -h) \simeq \boldsymbol{r}(0) - h\boldsymbol{v}(0) + \frac{h^2}{2}\boldsymbol{F}(0) \,. \tag{18.18}$$

Velocity–Verlet Algorithm Another version of the Verlet algorithm, which we recommend because of its increased stability, uses a forward-difference approximation for the derivative to advance *both* the position and velocity simultaneously:

$$\boldsymbol{r}_i(t+h) \simeq \boldsymbol{r}_i(t) + h\boldsymbol{v}_i(t) + \frac{h^2}{2}\boldsymbol{F}_i(t) + O(h^3) \,, \tag{18.19}$$

$$\boldsymbol{v}_i(t+h) \simeq \boldsymbol{v}_i(t) + h\overline{\boldsymbol{a}(t)} + O(h^2) \tag{18.20}$$

$$\simeq \boldsymbol{v}_i(t) + h\left[\frac{\boldsymbol{F}_i(t+h) + \boldsymbol{F}_i(t)}{2}\right] + O(h^2) \,. \tag{18.21}$$

Although this algorithm appears to be of lower order than (18.16), the use of updated positions when calculating velocities, and the subsequent use of these velocities, make both algorithms of similar precision.

Of interest is that (18.21) approximates the average force during a time step as $[\boldsymbol{F}_i(t+h) + \boldsymbol{F}_i(t)]/2$. Updating the velocity is a little tricky because we need the force at time $t+h$, which depends on the particle positions at $t+h$. Consequently, we must update all the particle positions and forces to $t+h$ before we update any velocities, while saving the forces at the earlier time for use in (18.21). As soon as the positions are updated, we impose periodic boundary conditions to establish that we have not lost any particles, and then we calculate the forces.

18.3
1D Implementation and Exercise

In the supplementary materials to this book, you will find a number of 2D animations (movies) of solutions to the MD equations. Some frames from these animations are shown in Figure 18.3. We recommend that you look at the movies in order to better visualize what the particles do during an MD simulation. In particular, these simulations use a potential and temperature that should lead to a solid or liquid system, and so you should see the particles binding together.

Listing 18.1 MD.py performs a 2D MD simulation with a small number of rather large time steps for just a few particles. To be realistic the user should change the parameters and the number of random numbers added to form the Gaussian distribution.

```
# MD.py:            Molecular dynamics in 2D

from visual import *
from visual.graph import *
import random

scene = display(x=0,y=0,width=350,height=350, title='Molecular Dynamics',
                range=10)
sceneK = gdisplay(x=0,y=350,width=600,height=150,title='Average KE',
         ymin=0.0,ymax=5.0,xmin=0,xmax=500,xtitle='time',ytitle='KE avg')
Kavegraph=gcurve(color= color.red)
sceneT = gdisplay(x=0,y=500,width=600,height=150,title='Average PE',
          ymin=-60,ymax=0.,xmin=0,xmax=500,xtitle='time',ytitle='PE avg')
Tcurve = gcurve(color=color.cyan)
Natom = 25
Nmax =  25
Tinit = 2.0

dens = 1.0                                          # Density (1.20 for fcc)
t1 = 0
x  = zeros( (Nmax),    float)
y  = zeros( (Nmax),    float)
vx = zeros( (Nmax),    float)
vy = zeros( (Nmax),    float)
fx = zeros( (Nmax, 2), float)
fy = zeros( (Nmax, 2), float)
L = int(1.*Natom**0.5)                                  # Side of lattice
atoms=[]

def twelveran():                              # Average 12 rands for Gaussian
    s=0.0
    for i in range (1,13):
        s += random.random()
    return s/12.0-0.5

def initialposvel():                                        # Initialize
    i = -1
    for ix in range(0, L):                    # x->   0  1  2  3  4
        for iy in range(0, L):                # y=0   0  5  10 15 20
            i = i + 1                         # y=1   1  6  11 16 21
            x[i]  = ix                        # y=2   2  7  12 17 22
            y[i]  = iy                        # y=3   3  8  13 18 23
            vx[i] = twelveran()               # y=4   4  9  14 19 24
            vy[i] = twelveran()               # numbering of 25 atoms
            vx[i] = vx[i]*sqrt(Tinit)
            vy[i] = vy[i]*sqrt(Tinit)
    for j in range(0,Natom):
```

```
        xc = 2*x[j] - 4
        yc = 2*y[j] - 4
        atoms.append(sphere(pos=(xc,yc), radius=0.5,color=color.red))

def sign(a, b):
    if (b >=  0.0):
        return abs(a)
    else:
        return  - abs(a)

def Forces(t, w, PE, PEorW):                                    # Forces
    # invr2 = 0.
    r2cut = 9.                                  # Switch: PEorW = 1 for PE
    PE = 0.
    for i in range(0, Natom):
        fx[i][t] = fy[i][t]  = 0.0
    for i in range( 0, Natom-1 ):
          for j in range(i + 1, Natom):
              dx = x[i] - x[j]
              dy = y[i] - y[j]
              if (abs(dx) > 0.50*L):
                  dx = dx  -  sign(L, dx)    # Interact with closer image
              if (abs(dy) > 0.50*L):
                  dy = dy  -  sign(L, dy)
              r2 = dx*dx  +  dy*dy
              if (r2 < r2cut):
                  if (r2 ==  0.):               # To avoid 0 denominator
                      r2 = 0.0001
                  invr2 = 1./r2
                  wij =  48.*(invr2**3 - 0.5) *invr2**3
                  fijx = wij*invr2*dx
                  fijy = wij*invr2*dy
                  fx[i][t] = fx[i][t]  +  fijx
                  fy[i][t] = fy[i][t]  +  fijy
                  fx[j][t] = fx[j][t]  -  fijx
                  fy[j][t] = fy[j][t]  -  fijy
                  PE = PE  +  4.*(invr2**3)*((invr2**3)  -  1.)
                  w = w  +  wij
    if (PEorW ==  1):
        return PE
    else:
        return w

def timevolution():
    avT = 0.0
    avP = 0.0
    Pavg = 0.0
    avKE = 0.0
    avPE = 0.0
    t1 = 0
    PE = 0.0
    h = 0.031                                                   # step
    hover2 = h/2.0
    # initial KE & PE via Forces
    KE = 0.0
    w = 0.0
    initialposvel()
    for i in range(0, Natom):
        KE = KE+(vx[i]*vx[i]+vy[i]*vy[i])/2.0
  # System.out.println(""+t+" PE= "+PE+" KE = "+KE+" PE+KE = "+(PE+KE));
    PE = Forces(t1,w,PE,1)
    time =1
    while 1:
        rate(100)
        for i in range(0,  Natom):
            PE = Forces(t1,w,PE,1)
            x[i] = x[i] + h*(vx[i] + hover2*fx[i][t1])
```

```
            y[i] = y[i] + h*(vy[i] + hover2*fy[i][t1]);
            if x[i] <= 0.:
                x[i] = x[i] + L            # Periodic boundary conditions
            if x[i] >= L :
               x[i] = x[i] - L
            if y[i] <= 0.:
                y[i] = y[i] + L
            if y[i] >= L:
                y[i] = y[i] - L
            xc = 2*x[i] - 4
            yc = 2*y[i] - 4
            atoms[i].pos=(xc,yc)
        PE = 0.
        t2=1
        PE = Forces(t2, w, PE, 1)
        KE = 0.
        w = 0.
        for  i in range(0 , Natom):
            vx[i] = vx[i] + hover2*(fx[i][t1] + fx[i][t2])
            vy[i] = vy[i] + hover2*(fy[i][t1] + fy[i][t2])
            KE = KE + (vx[i]*vx[i] + vy[i]*vy[i])/2
        w = Forces(t2, w, PE, 2)
        P=dens*(KE+w)
        T=KE/(Natom)
        # increment averages
        avT = avT + T
        avP = avP + P
        avKE = avKE + KE
        avPE = avPE + PE
        time += 1
        t=time
        if (t==0):
            t=1
        Pavg  = avP /t
        eKavg = avKE /t
        ePavg = avPE /t
        Tavg  = avT /t
        pre = (int)(Pavg*1000)
        Pavg = pre/1000.0
        kener = (int)(eKavg*1000)
        eKavg = kener/1000.0
        Kavegraph.plot(pos=(t,eKavg))
        pener = (int)(ePavg*1000)
        ePavg = pener/1000.0
        tempe = (int)(Tavg*1000000)
        Tavg = tempe/1000000.0
        Tcurve.plot(pos=(t,ePavg),display=sceneT)

timevolution()
```

The program MD.py in Listings 18.1 implements an MD simulation in 1D using the velocity-Verlet algorithm. Use it as a model and do the following:

1. Establish that you can run and visualize the 1D simulation.
2. Place the particles initially at the sites of a simple cubic lattice. The equilibrium configuration for a Lennard–Jones system at low temperature is a face-centered-cubic, and if your simulation is running properly, then the particles should migrate from SC to FCC. An FCC lattice has four quarters of a particle per unit cell, so an L^3 box with a lattice constant L/N contains (parts of) $4N^3 = 32, 108, 256, \ldots$ particles.

3. To save computing time, assign initial particle velocities corresponding to a fixed-temperature Maxwellian distribution.
4. Print the code and indicate on it which integration algorithm is used, where the periodic boundary conditions are imposed, where the nearest image interaction is evaluated, and where the potential is cut off.
5. A typical time step is $\Delta t = 10^{-14}$ s, which in our natural units equals 0.004. You probably will need to make 10^4–10^5 such steps to equilibrate, which corresponds to a total time of only 10^{-9} s (a lot can happen to a speedy molecule in 10^{-9} s). Choose the *largest* time step that provides stability and gives results similar to Figure 18.5.
6. The PE and KE change with time as the system equilibrates. Even after that, there will be fluctuations because this is a dynamic system. Evaluate the time-averaged energies for an equilibrated system.
7. Compare the final temperature of your system to the initial temperature. Change the initial temperature and look for a simple relation between it and the final temperature (Figure 18.6).

18.4 Analysis

1. Modify your program so that it outputs the coordinates and velocities of a few particles throughout the simulation. Note that you do not need as many time steps to follow a trajectory as you do to compute it, and so you may want to use the *mod* operator `%100` for output.
   a) Start your assessment with a 1D simulation at zero temperature. The particles should remain in place without vibration. Increase the temperature and note how the particles begin to move about and interact.
   b) Try starting off all your particles at the minima in the Lennard–Jones potential. The particles should remain bound within the potential until you raise the temperature.
   c) Repeat the simulations for a 2D system. The trajectories should resemble billiard ball-like collisions.
   d) Create an animation of the time-dependent locations of several particles.
   e) Calculate and plot the root-mean-square displacement of molecules as a function of temperature:

$$R_{\text{rms}} = \sqrt{\langle |\boldsymbol{r}(t + \Delta t) - \boldsymbol{r}(t)|^2 \rangle}\,, \tag{18.22}$$

   where the average is over all the particles in the box. Determine the approximate time dependence of R_{rms}.
   f) Test your system for time-reversal invariance. Stop it at a fixed time, reverse all velocities, and see if the system retraces its trajectories back to the initial configuration after this same fixed time.

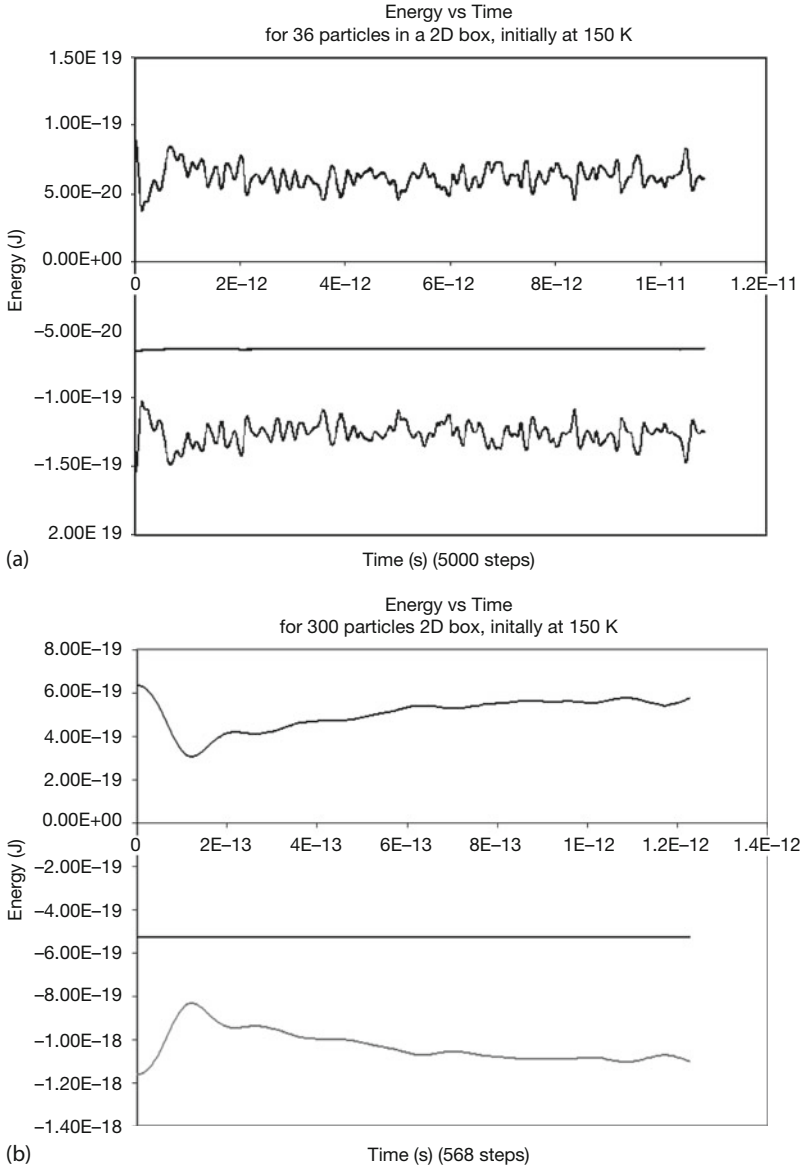

Figure 18.5 The kinetic, potential, and total energy as a function of time or number of steps for a 2D MD simulation with 36 particles (a), and 300 particles (b), both with an initial temperature of 150 K. The potential energy is negative, the kinetic energy is positive, and the total energy is seen to be conserved (flat).

2. *Hand Computation* We wish to make an MD simulation *by hand* of the positions of particles 1 and 2 that are in a 1D box of side 8. For an origin located at the center of the box, the particles are initially at rest and at loca-

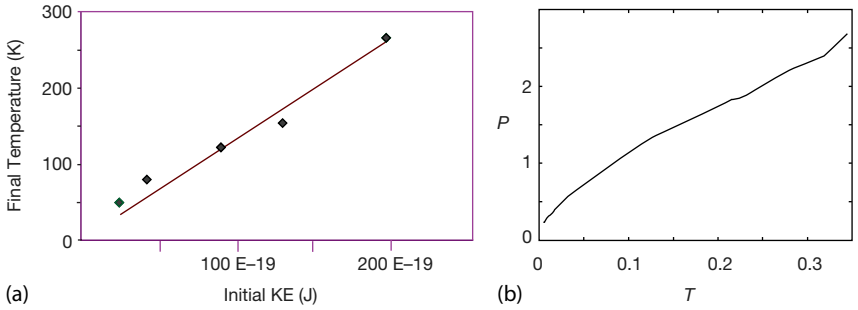

Figure 18.6 (a) The temperature after equilibration as a function of initial kinetic energy for a 2D MD simulation with 36 particles. The two are nearly proportional. (b) The pressure vs. temperature for a simulation with several hundred particles. An ideal gas (noninteracting particles) would yield a straight line (courtesy of J. Wetzel).

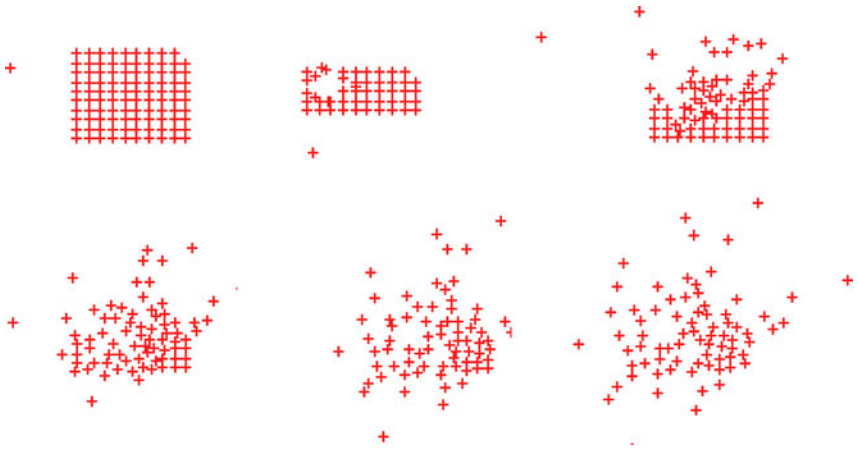

Figure 18.7 A simulation of a projectile shot into a group of particles. The energy introduced by the projectile is seen to lead to evaporation of the particles (courtesy of J. Wetzel).

tions $x_i(0) = -x_2(0) = 1$. The particles are subject to the force

$$F(x) = \begin{cases} 10\,, & \text{for} \quad |x_1 - x_2| \le 1\,, \\ -1\,, & \text{for} \quad 1 \le |x_1 - x_2| \le 3\,, \\ 0\,, & \text{otherwise}\,. \end{cases} \tag{18.23}$$

Use a simple algorithm to determine the positions of the particles up until the time they leave the box. Make sure to apply periodic boundary conditions. *Hint:* Because the configuration is symmetric, you know the location of particle 2 by symmetry and do not need to solve for it. We suggest the Verlet algorithm (no velocities) with a forward-difference algorithm to initialize it. To speed things along, use a time step of $h = 1/\sqrt{2}$.

3. *Diffusion* It is well known that light molecules diffuse more quickly than heavier ones. See if you can simulate diffusion with your MD simulation using a Lennard–Jones potential and periodic boundary conditions (Satoh, 2011).
   a) Generalize the velocity-Verlet algorithm so that it can be used for molecules of different masses.
   b) Modify the simulation code so that it can be used for five heavy molecules of mass $M = 10$ and five light molecules of mass $m = 1$.
   c) Start with the molecules placed randomly near the center of the square simulation region.
   d) Assign random initial velocities to the molecules.
   e) Run the simulation several times and verify visually that the lighter molecules tend to diffuse more quickly than the heavier ones.
   f) For each ensemble of molecules, calculate the RMS velocity at regular instances of time, and then plot the RMS velocities as functions of time. Do the lighter particles have a greater RMS velocity?
4. As shown in Figure 18.7, simulate the impact of a projectile with a block of material.

19
PDE Review and Electrostatics via Finite Differences and Electrostatics via Finite Differences

This chapter is the first of several dealing with partial differential equations (PDEs); several because PDEs are more complex than ODEs, and because each type of PDE requires its own algorithm. We start the chapter with a discussion of PDEs in general, and the requirements for a unique solution of each type to exist. Then we get down to business and examine the simple, but powerful, *finite-differences* method for solving Poisson's and Laplace's equations on a lattice in space. Chapter 23 covers the more complicated, but ultimately more efficient, *finite elements* method for solving the same equations.

19.1
PDE Generalities

Physical quantities such as temperature and pressure vary continuously in both space and time. Such being our world, the function or *field* $U(x, y, z, t)$ used to describe these quantities must contain independent space and time variations. As time evolves, the changes in $U(x, y, z, t)$ at any one position affect the field at neighboring points. This means that the dynamic equations describing the dependence of U on four independent space–time variables must be written in terms of partial derivatives, and therefore the equations must be *partial differential equations* (PDEs), in contrast to ordinary differential equations (ODEs).

The most general form for a two-independent variable PDE is

$$A\frac{\partial^2 U}{\partial x^2} + 2B\frac{\partial^2 U}{\partial x \partial y} + C\frac{\partial^2 U}{\partial y^2} + D\frac{\partial U}{\partial x} + E\frac{\partial U}{\partial y} = F\,, \tag{19.1}$$

where A, B, C, and F are arbitrary functions of the variables x and y. In Table 19.1, we define the classes of PDEs by the value of the discriminant $d = AC - B^2$ in row two (Arfken and Weber, 2001), and give examples in rows three and four.

We usually think of an elliptic equation as containing the second-order derivatives of all the variables, with all having the same sign when placed on the same side of the equal sign; a parabolic equation as containing a first-order derivative in one variable and a second-order derivative in the other; and a hyperbolic equa-

Computational Physics, 3rd edition. Rubin H. Landau, Manuel J. Páez, Cristian C. Bordeianu.
©2015 WILEY-VCH Verlag GmbH & Co. KGaA. Published 2015 by WILEY-VCH Verlag GmbH & Co. KGaA.

Table 19.1 Three categories of PDE based on the value of their discriminant d.

Elliptic	Parabolic	Hyperbolic
$d = AC - B^2 > 0$	$d = AC - B^2 = 0$	$d = AC - B^2 < 0$
$\nabla^2 U(x) = -4\pi\rho(x)$	$\nabla^2 U(\boldsymbol{x}, t) = a\partial U/\partial t$	$\nabla^2 U(\boldsymbol{x}, t) = c^{-2}\partial^2 U/\partial t^2$
Poisson's	Heat	Wave

tion as containing second-order derivatives of all the variables, with opposite signs when placed on the same side of the equal sign.

After solving enough problems, one often develops some physical intuition as to whether one has sufficient *boundary conditions* for there to exist a unique solution for a given physical situation (this, of course, is in addition to requisite *initial conditions*). Table 19.2 gives the requisite boundary conditions for a unique solution to exist for each type of PDE. For instance, a string tied at both ends and a heated bar placed in an infinite heat bath are physical situations for which the boundary conditions are adequate. If the boundary condition is the value of the solution on a surrounding closed surface, we have a *Dirichlet boundary condition*. If the boundary condition is the value of the normal derivative on the surrounding surface, we have a *Neumann boundary condition*. If the value of both the solution and its derivative are specified on a closed boundary, we have a *Cauchy boundary condition.* Although having an adequate boundary condition is necessary for a unique solution, having too many boundary conditions, for instance, both Neumann and Dirichlet, may be an overspecification for which no solution exists.[1)]

Solving PDEs numerically differs from solving ODEs in a number of ways. First, because we are able to write all ODEs in a standard form

$$\frac{\mathrm{d}\boldsymbol{y}(t)}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{y}, t)\ , \tag{19.2}$$

with t the single independent variable, we are able to use a standard algorithm such as `rk4` to solve all such equations. Yet because PDEs have several independent variables, for example, $\rho(x, y, z, t)$, we would have to apply (19.2) simultaneously and independently to each variable, which would be very complicated. Second, because there are more equations to solve with PDEs than with ODEs, we need more information than just the two *initial conditions* $[x(0), \dot{x}(0)]$. In addition, because each PDE often has its own particular set of boundary conditions, we have to develop a special algorithm for each particular problem.

1) Although conclusions drawn for exact PDEs may differ from those drawn for the finite-difference equations, we use for our algorithms, they are usually the same. In fact, Morse and Feshbach (Morse and Feshbach, 1953) use the finite-difference form to derive the relations between boundary conditions and uniqueness for each type of equation shown in Table 19.2 (Jackson, 1988).

19.2
Electrostatic Potentials

Your *problem* is to find the electric potential for all points *inside* the charge-free square shown in Figure 19.1. The bottom and sides of the region are made up of wires that are "grounded" (kept at 0 V). The top wire is connected to a voltage source that keeps it at a constant 100 V.

19.2.1
Laplace's Elliptic PDE (Theory)

We consider the entire square in Figure 19.1 as our boundary with the voltages prescribed upon it. If we imagine infinitesimal insulators placed at the top corners of the box, then we will have a closed boundary. Because the values of the potential are given on all sides, we have Neumann conditions on the boundary and, according to Table 19.2, a unique and stable solution.

Table 19.2 The relation between boundary conditions and uniqueness for PDEs.

Boundary Condition	Elliptic (Poisson equation)	Hyperbolic (Wave equation)	Parabolic (Heat equation)
Dirichlet open surface	Underspecified	Underspecified	*Unique and stable (1D)*
Dirichlet closed surface	*Unique and stable*	Overspecified	Overspecified
Neumann open surface	Underspecified	Underspecified	*Unique and stable (1D)*
Neumann closed surface	*Unique and stable*	Overspecified	Overspecified
Cauchy open surface	Nonphysical	*Unique and stable*	Overspecified
Cauchy closed surface	Overspecified	Overspecified	Overspecified

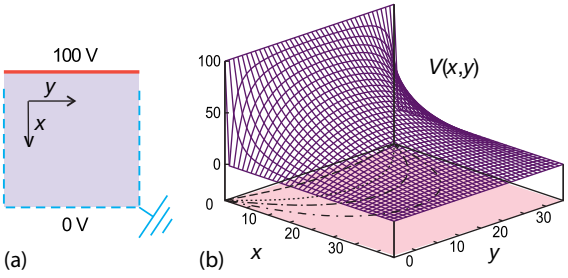

Figure 19.1 (a) The shaded region of space within a square in which we determine the electric potential by solving Laplace's equation. There is a wire at the top kept at a constant 100 V and a grounded wire (dashed) at the sides and bottom. (b) The computed electric potential as a function of x and y. The projections onto the shaded xy plane are equipotential (contour) lines.

It is known from classical electrodynamics that the electric potential $U(\boldsymbol{x})$ arising from static charges satisfies Poisson's PDE (Jackson, 1988):

$$\nabla^2 U(\boldsymbol{x}) = -4\pi\rho(\boldsymbol{x}) \ , \tag{19.3}$$

where $\rho(\boldsymbol{x})$ is the charge density. In charge-free regions of space, that is, regions where $\rho(\boldsymbol{x}) = 0$, the potential satisfies *Laplace's equation*:

$$\nabla^2 U(\boldsymbol{x}) = 0 \ . \tag{19.4}$$

Both these equations are elliptic PDEs of a form that occurs in various applications. We solve them in 2D rectangular coordinates:

$$\frac{\partial^2 U(x,\,y)}{\partial x^2} + \frac{\partial^2 U(x,\,y)}{\partial y^2} = 0 \ , \quad \text{Laplace's equation} \ , \tag{19.5}$$

$$\frac{\partial^2 U(x,\,y)}{\partial x^2} + \frac{\partial^2 U(x,\,y)}{\partial y^2} = -4\pi\rho(\boldsymbol{x}) \ , \quad \text{Poisson's equation} \ . \tag{19.6}$$

In both cases, we see that the potential depends simultaneously on x and y. For Laplace's equation, the charges, which are the source of the field, enter indirectly by specifying the potential values in some region of space; for Poisson's equation they enter directly.

19.3 Fourier Series Solution of a PDE

For the simple geometry of Figure 19.1, an analytic solution of Laplace's equation (19.5) exists in the form of an infinite series. If we assume that the solution is the product of independent functions of x and y and substitute the product into (19.5), we obtain

$$U(x,\,y) = X(x)Y(y) \quad \Rightarrow \quad \frac{\mathrm{d}^2 X(x)/\,\mathrm{d}x^2}{X(x)} + \frac{\mathrm{d}^2 Y(y)/\,\mathrm{d}y^2}{Y(y)} = 0 \ . \tag{19.7}$$

Because $X(x)$ is a function of only x, and $Y(y)$ is a function of only y, the derivatives in (19.7) are *ordinary* as opposed to *partial* derivatives. Because $X(x)$ and $Y(y)$ are assumed to be independent, the only way (19.7) can be valid for *all* values of x and y is for each term in (19.7) to be equal to a constant:

$$\frac{\mathrm{d}^2 Y(y)/\,\mathrm{d}y^2}{Y(y)} = -\frac{\mathrm{d}^2 X(x)/\,\mathrm{d}x^2}{X(x)} = k^2 \tag{19.8}$$

$$\Rightarrow \quad \frac{\mathrm{d}^2 X(x)}{\mathrm{d}x^2} + k^2 X(x) = 0 \ , \quad \frac{\mathrm{d}^2 Y(y)}{\mathrm{d}y^2} - k^2 Y(y) = 0 \ . \tag{19.9}$$

We shall see that this choice of sign for the constant matches the boundary conditions and gives us periodic behavior in x. The other choice of sign would give periodic behavior in y, and that would not work with these boundary conditions.

The solutions for $X(x)$ are periodic, and those for $Y(y)$ are exponential:

$$X(x) = A \sin kx + B \cos kx \ , \quad Y(y) = Ce^{ky} + De^{-ky} \ . \tag{19.10}$$

The $x = 0$ boundary condition $U(x = 0, y) = 0$ can be met only if $B = 0$. The $x = L$ boundary condition $U(x = L, y) = 0$ can be met only for

$$kL = n\pi \ , \quad n = 1, 2, \ldots \tag{19.11}$$

Such being the case, for each value of n there is the solution

$$X_n(x) = A_n \sin\left(\frac{n\pi}{L}x\right) \ . \tag{19.12}$$

For each value of k_n, $Y(y)$ must satisfy the y boundary condition $U(x, 0) = 0$, which requires $D = -C$:

$$Y_n(y) = C(e^{k_n y} - e^{-k_n y}) \equiv 2C \sinh\left(\frac{n\pi}{L}y\right) \ . \tag{19.13}$$

Because we are solving linear equations, the principle of linear superposition holds, which means that the most general solution is the sum of the products:

$$U(x, y) = \sum_{n=1}^{\infty} E_n \sin\left(\frac{n\pi}{L}x\right) \sinh\left(\frac{n\pi}{L}y\right) \ . \tag{19.14}$$

The E_n values are arbitrary constants and are fixed by requiring the solution to satisfy the remaining boundary condition at $y = L$, $U(x, y = L) = 100\,\mathrm{V}$:

$$\sum_{n=1}^{\infty} E_n \sin\frac{n\pi}{L}x \sinh n\pi = 100\,\mathrm{V} \ . \tag{19.15}$$

We determine the constants E_n by projection: multiply both sides of the equation by $\sin m\pi/Lx$, with m an integer, and integrate from 0 to L:

$$\sum_{n}^{\infty} E_n \sinh n\pi \int_0^L dx \sin\frac{n\pi}{L}x \sin\frac{m\pi}{L}x = \int_0^L dx 100 \sin\frac{m\pi}{L}x \ . \tag{19.16}$$

The integral on the LHS is nonzero only for $n = m$, which yields

$$E_n = \begin{cases} 0 \ , & \text{for} \quad n \text{ even} \ , \\ \frac{4(100)}{n\pi \sinh n\pi} \ , & \text{for} \quad n \text{ odd} \ . \end{cases} \tag{19.17}$$

Finally, we obtain an infinite series (analytic solution) for the potential at any point (x, y):

$$U(x, y) = \sum_{n=1,3,5,\ldots}^{\infty} \frac{400}{n\pi} \sin\left(\frac{n\pi x}{L}\right) \frac{\sinh(n\pi y/L)}{\sinh(n\pi)} \ . \tag{19.18}$$

19.3.1
Polynomial Expansion as an Algorithm

If we try to use (19.18) as an algorithm, we must terminate the sum at some point. Yet in practice the convergence of the series is so painfully slow that many terms are needed for good accuracy, and so the round-off error may become a problem. In addition, the sinh functions in (19.18) overflow for large n, which can be avoided somewhat by expressing the quotient of the two sinh functions in terms of exponentials and then taking a large n limit:

$$\frac{\sinh(n\pi y/L)}{\sinh(n\pi)} = \frac{e^{n\pi(y/L-1)} - e^{-n\pi(y/L+1)}}{1 - e^{-2n\pi}} \xrightarrow[n\to\infty]{} e^{n\pi(y/L-1)} . \tag{19.19}$$

A third problem with the "analytic" solution is that a Fourier series converges only in the *mean square* (Figure 19.2). This means that it converges to the *average* of the left- and right-hand limits in the regions where the solution is discontinuous (Kreyszig, 1998), such as in the corners of the box. Explicitly, what you see in Figure 19.2 is a phenomenon known as the *Gibbs overshoot* that occurs when a Fourier series with a finite number of terms is used to represent a discontinuous function. Rather than fall off abruptly, the series develops large oscillations that tend to overshoot the function at the corner. To obtain a smooth solution, we had to sum 40 000 terms, where, in contrast, the numerical solution required only several hundred steps.

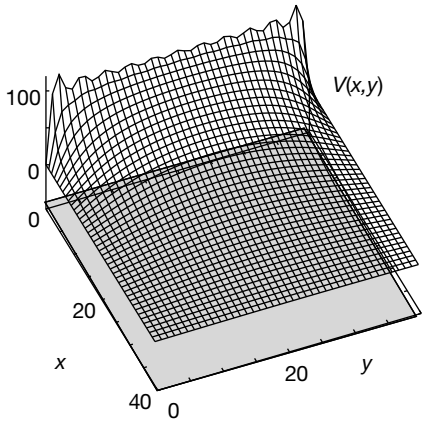

Figure 19.2 The analytic (Fourier series) solution of Laplace's equation summing 21 terms. Gibbs overshoot leads to the oscillations near $x = 0$, and persist even if a large number of terms is summed.