Computational Physics

Exercises for Chapter 9

Exercise 9.1: Solution of the Poisson equation using relaxation

Write a program, or modify the one from Example 9.1, to solve the Poisson equation for the system described in Example 9.2 using the relaxation method on a 100×100 grid. Work in units where $\epsilon_0 = 1$ and continue the iteration until your solution for the electric potential changes by less than 10^{-6} V per step at every grid point. Your results should look similar to Fig. 9.5 on page 407.

Exercise 9.2: Solution of the Poisson equation using matrix methods

Using the matrix form of the Laplacian from Eq. (9.7) on page 401, we can write the Poisson equation as

$$\frac{1}{a^2}\mathbf{L}\boldsymbol{\phi} = -\frac{\boldsymbol{\rho}}{\epsilon_0},$$

where $\boldsymbol{\rho}$ is the vector with elements ρ_i equal to the charge density at each grid point i. This equation has the form $\mathbf{Ax} = \mathbf{v}$ of a system of linear simultaneous equations of the kind we studied in Chapter 6, and we can solve it using the matrix methods described there. This approach is usually not as fast as using the relaxation method, but for smaller systems it is tractable, and it is reliable and easy to implement.

a) Solve the problem from Example 9.2 using a matrix method as follows. First, number the N grid points from 0 to $N-1$ with the numbers increasing along the first row of the array, then the second row, and so on. Using this numbering scheme create the $N \times N$ graph Laplacian matrix for the grid following the prescription in Eq. (9.8). Also create the vector $\mathbf{v} = -a^2\boldsymbol{\rho}/\epsilon_0$ in units where $\epsilon_0 = 1$.

b) Solve the equation $\mathbf{L}\boldsymbol{\phi} = \mathbf{v}$ for $\boldsymbol{\phi}$ using the `solve` function from `numpy` for a grid of 100×100 points.

c) This gives you the electric potential ϕ on each grid point, but not in a very useful form: the values are contained in a single one-dimensional array. Create a new, two-dimensional array in the shape of the original square lattice and copy the values of ϕ into it, each value going into its appropriate spot in the array. Then make a visualization of the resulting array using `imshow`. You should get results similar to those in Fig. 9.5 on page 407.

If you also did Exercise 9.1 above, you may find that this matrix method is actually faster than the relaxation method, but this is misleading, for two reasons. The first is that the `solve` function is a highly optimized piece of software, compiled into so-called machine language, and hence has a somewhat unfair speed advantage over standard Python. The second is that the relaxation method can actually be sped up a lot by a few simple modifications. These modifications are the subject of the next section.

Exercise 9.3: Solution of Laplace's equation using matrix methods

By analogy with Exercise 9.2, one can also solve Laplace's equation using matrix methods, although the approach is less obvious. Laplace's equation on a square grid implies Eq. (9.9), but for grid points adjacent to the boundaries the potential on one or more of the neighboring points is fixed by the boundary

conditions. For the square box in Example 9.1, for instance, with voltage V on the top boundary, the equation for grid points adjacent to this boundary would be

$$\phi(x+a,y) + \phi(x-a,y) + \phi(x,y-a) + V - 4\phi(x,y) = 0.$$

a) Show that Laplace's equation for this example can be written in the form $\mathbf{L}\boldsymbol{\phi} = \mathbf{v}$, where $\mathbf{L}$ is the graph Laplacian matrix of Eq. (9.8) for the interior points only (i.e., excluding the boundaries), and $\mathbf{v}$ is the vector with elements

$$v_i = \begin{cases} -V & \text{if point } i \text{ is adjacent to the top edge,} \\ 0 & \text{otherwise.} \end{cases}$$

b) Number the N grid points from 0 to $N-1$ with the numbers increasing along the first row of the array, then the second row, and so on. Using this numbering scheme create the $N \times N$ graph Laplacian matrix for the grid and the vector $\mathbf{v}$.

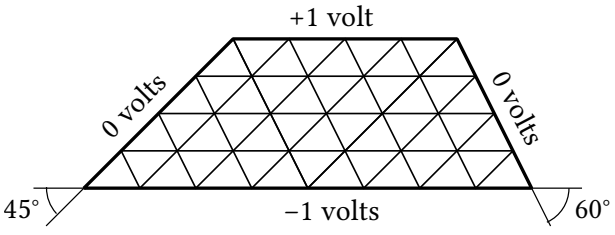
c) Solve the equation $\mathbf{L}\boldsymbol{\phi} = \mathbf{v}$ for $\boldsymbol{\phi}$ using the `solve` function from `numpy` for a grid of 100×100 points.

d) This gives you the electric potential ϕ on each grid point, but the values are contained in a single one-dimensional array, not arranged on a grid. Create a new, two-dimensional array in the shape of the original grid and copy the values of ϕ into it, each value going into its appropriate spot in the array. Then make a visualization of the resulting array using `imshow`. You should get results similar to those in Fig. 9.3 on page 405.

If you try running the relaxation method program from Example 9.1 you may find that the matrix method in this exercise is actually faster than the relaxation method, but this is mostly an illusion—see the discussion at the end of Exercise 9.2 above.

Exercise 9.4: Laplace's equation in an irregular space

Let us solve for the electric potential inside a trapezoidal space like this:



with voltages on the walls as indicated. We triangulate the space as shown, with M triangles along the top and side edges and $2M$ triangles along the bottom, where $M = 4$ in the figure.

a) Calculate the matrix elements L_{ij}, Eq. (9.18), for all pairs i, j of adjacent points as well as the diagonal elements L_{ii}.

b) Write a program to solve for the electric potential on all grid points for the case $M = 4$. You will have to choose a scheme for storing the field values ϕ_i. There are various possibilities, but perhaps the simplest is to just store the rows of grid points in a two-dimensional array. Not all rows have the same length, but if you don't mind wasting a little memory space you can just make the array large enough to store the longest row (which has $2M + 1$ grid points).

Now write code to perform the Jacobi iteration and compute the electric potential. You can use the program from Example 9.1 on page 403 as a starting point if you wish. Continue the iteration until your solution changes by less than 10^{-6} V per step at every grid point.

c) Add code to your program to visualize the solution. The simplest way to do this is just to make a density plot of the array holding the values ϕ_i. This will produce a picture with extra non-functional grid points at the end of each row, and the angles on the grid will not be correct. Alternatively you could compute the actual location of each grid point in two-dimensional space and then visualize the value of ϕ_i at each point by, for example, plotting a circle of an appropriate color, which can be done using the `scatter` function in `pyplot`. For a more sophisticated approach, take a look at the function `meshplot`, which can be found in Appendix C, Section C.4, and also in the file `meshplot.py` in the online resources, which makes a true density plot of the data on a triangulation.

d) Once you have your program working, increase the size of the system to $M = 100$ and rerun the calculation to produce a final, high-resolution picture of the solution.

Exercise 9.5: Laplace's equation in a complex space

The files `mesh.txt` and `triangles.txt` contain data for the triangulation shown in Fig. 9.7b, a square with a circular hole cut out of the middle. The file `mesh.txt` contains an $N \times 2$ array of the x and y coordinates of each point and `triangles.txt` contains an $N \times 3$ array with each row specifying the numbers of the grid points at the corners of one of the triangles.

a) Write a Python program to visualize the grid as a set of points and lines. For instance, you could call the `plot` function from `pyplot` separately for each triangle to draw the edges of the triangle, then draw the grid points as circles using the `scatter` function. Alternatively, take a look at the `triplot` function in `pyplot`, which is designed specifically for drawing triangulations like this.

b) The additional file `boundaries.txt` contains N integers specifying whether each point falls on a boundary of the space or not. The numbers in this file are 0 if a point is not on a boundary, 1 if it is on the boundary of the square, and 2 if it is on the inner circle. Add code to your program to read this file, and then modify your plot so that the circles representing the inner and outer boundaries are different colors from the circles representing the interior.

Given the triangles, it is straightforward to calculate the graph Laplacian matrix. For a triangle with corners at locations $\mathbf{r}_1$, $\mathbf{r}_2$, and $\mathbf{r}_3$, the matrix element L_{12} has a contribution from the edge between corners 1 and 2 equal to $\cot \alpha_3$ where α_3 is the angle at corner 3. Defining vectors along the edges of the triangle $\mathbf{r}_{13} = \mathbf{r}_1 - \mathbf{r}_3$ and $\mathbf{r}_{23} = \mathbf{r}_2 - \mathbf{r}_3$, we have $\mathbf{r}_{13} \cdot \mathbf{r}_{23} = |\mathbf{r}_{13}|\,|\mathbf{r}_{23}| \cos \alpha_3$, and hence

$$\alpha_3 = \arccos\left(\frac{\mathbf{r}_{13} \cdot \mathbf{r}_{23}}{|\mathbf{r}_{13}|\,|\mathbf{r}_{23}|}\right),$$

and there are similar formulas for the angles at the other two corners.

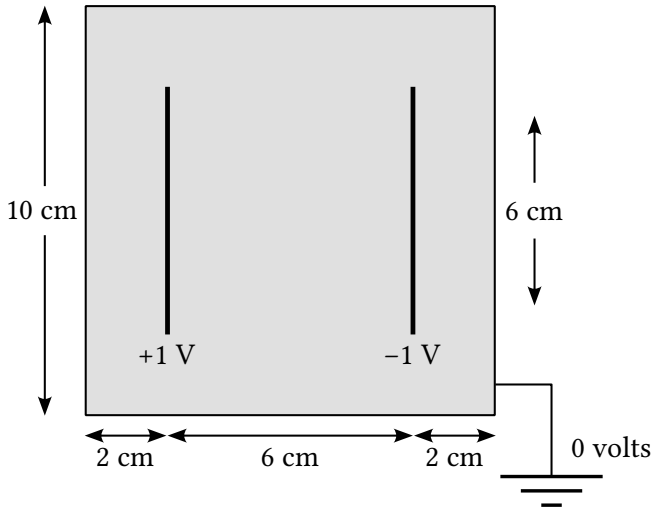
c) Using this approach, write a new program to calculate the cotangents of the angles and hence compute the complete graph Laplacian matrix for the grid. Hint: The arccos function in Python lives in the `math` package and is called `acos`.

d) Solve Laplace's equation on the grid using the Jacobi method for the case where the boundary of the inner circle is at a potential of 1 volt and the square outer boundary is at 0 volts everywhere.

e) Add code to your program to visualize the solution. A simple way to do this would be to use the `scatter` function again to draw the grid points, with circles whose colors represent the value of ϕ. For a more sophisticated approach, take a look at the function `meshplot` in Appendix C, Section C.4 (and also in the file `meshplot.py` in the online resources), which makes a true density plot of data on a triangulated grid.

Exercise 9.6: Use the combined overrelaxation/Gauss–Seidel method to solve Laplace's equation for the two-dimensional problem in Example 9.1—a square box 1 m on each side, at voltage $V = 1$ volt along the top wall and zero volts along the other three. Use a grid of spacing $a = 1$ cm, so that there are 100 grid points along each wall, or 101 if you count the points at both ends. Continue the iteration of the method until the value of the electric potential changes by no more than $\delta = 10^{-6}$ V at any grid point on any step, then make a density plot of the final solution, similar to that shown in Fig. 9.3. Experiment with different values of ω to find which one gives the fastest solution. In general larger values are faster, but if you use too large a value the speed will drop off and for values above 1 the calculation becomes unstable. (As mentioned above, you should find that a value around 0.9 does well.)

Exercise 9.7: Consider this system, which is a simple model of an electronic capacitor, consisting of two flat metal plates enclosed in a square metal box:



For simplicity let us model the system in two dimensions. Using any of the methods we have studied, write a program to calculate the electrostatic potential in the box on a grid of 100×100 points, where the walls of the box are at voltage zero and the two plates (which are of negligible thickness) are at

voltages ±1 V as shown. Have your program calculate the value of the potential at each grid point to a precision of 10^{-6} volts and then make a density plot of the result.

Hint: Notice that the capacitor plates are at fixed voltage, not fixed charge, so this problem differs from the problem with the two charges in Exercise 9.1. In effect, the capacitor plates are part of the boundary conditions, similar to the hot and cold plates in Example 9.3.

Exercise 9.8: Thermal diffusion in the Earth's crust

A classic example of a diffusion problem with a time-varying boundary condition is the diffusion of heat into the crust of the Earth, as surface temperature varies with the seasons. Suppose the mean daily temperature at a particular point on the surface varies as:

$$T_0(t) = A + B \sin \frac{2\pi t}{\tau},$$

where $\tau = 365$ days, $A = 10°\text{C}$ and $B = 12°\text{C}$. At a depth of 20 m below the surface almost all annual temperature variation is ironed out and the temperature is, to a good approximation, a constant 11°C (which is higher than the mean surface temperature of 10°C—temperature increases with depth, due to heating from the hot core of the planet). The thermal diffusivity of the Earth's crust varies somewhat from place to place, but for our purposes we will treat it as constant with value $D = 0.1\,\text{m}^2\,\text{day}^{-1}$.

Write a program, or modify one of the ones earlier in this chapter, to calculate the temperature profile of the crust as a function of depth up to 20 m and time up to 10 years. Start with temperature everywhere equal to 10°C, except at the surface and the deepest point, choose values for the number of grid points and the time-step h, and then run your program for the first nine simulated years, to allow it to settle down into whatever pattern it reaches. Then for the tenth and final year plot four temperature profiles taken at 3-month intervals on a single graph to illustrate how the temperature changes as a function of depth and time.

Exercise 9.9: FTCS solution of the wave equation

Consider a piano string of length L, initially at rest. At time $t = 0$ the string is struck by the piano hammer a distance d from the end of the string:

The string vibrates as a result of being struck, except at the ends, $x = 0$ and $x = L$, where it is held fixed.

a) Write a program that uses the FTCS method to solve the complete set of simultaneous first-order equations, Eq. (9.43), for the case $v = 100\,\text{m}\,\text{s}^{-1}$, with the initial condition that $\phi(x) = 0$ everywhere but the velocity $\psi(x)$ is nonzero, with profile

$$\psi(x) = C\frac{x(L-x)}{L^2} \exp\left[-\frac{(x-d)^2}{2\sigma^2}\right],$$

where $L = 1\,\text{m}$, $d = 10\,\text{cm}$, $C = 1\,\text{m}\,\text{s}^{-1}$, and $\sigma = 0.3\,\text{m}$. You will also need a value for the time-step h. A reasonable choice is $h = 10^{-6}\,\text{s}$.

b) Make an animation of the motion of the piano string using the `qdraw` package that we introduced in Section 3.4. There are various ways you could do this. A simple one would be just to place a small circle at the location of each grid point on the string. A more sophisticated approach would be to use the `line` object from the `qdraw` package, which can draw and animate a curve on the screen—see Appendix B for details. Since the vertical displacement of the string is much less than its horizontal length, you will probably need to multiply the vertical displacement by a fairly large factor to make it visible on the screen—a scale factor of 100 or more will probably be needed.

Allow your animation to run for some time, until numerical instabilities start to appear.

Exercise 9.10: The Crank–Nicolson method and the wave equation

In this exercise you will use the Crank–Nicolson method to solve for the motion of the piano string considered previously in Exercise 9.9. The string has length L and is initially at rest, then at time $t = 0$ it is struck by the piano hammer a distance d from the end, making it vibrate along its length, except at its ends at $x = 0$ and $x = L$, where it is held fixed.

The solution involves dividing the space into grid points with spacing a and applying the Crank–Nicolson equations, Eq. (9.64), repeatedly to solve for the position $\phi(x,t)$ and velocity $\psi(x,t)$ of the string, with initial conditions $\phi(x,0) = 0$ for all x and

$$\psi(x,0) = C\frac{x(L-x)}{L^2}\exp\left[-\frac{(x-d)^2}{2\sigma^2}\right],$$

where $L = 1\,\text{m}$, $d = 10\,\text{cm}$, $C = 1\,\text{m}\,\text{s}^{-1}$, and $\sigma = 0.3\,\text{m}$.

There are various ways to implement the method, but one approach is to define a vector

$$\mathbf{u}(t) = \begin{pmatrix} \phi(a,t) \\ \psi(a,t) \\ \phi(2a,t) \\ \psi(2a,t) \\ \phi(3a,t) \\ \vdots \end{pmatrix}.$$

Note how this vector contains the values of both ϕ and ψ in alternating positions.

a) Show that the Crank–Nicolson equations of Eq. (9.64) can be written in matrix form as $\mathbf{A}\mathbf{u}(t+h) = \mathbf{B}\mathbf{u}(t)$, where the matrices $\mathbf{A}$ and $\mathbf{B}$ are

$$\mathbf{A} = \begin{pmatrix} 1 & -c_1 & & & & & & \\ 2c_2 & 1 & -c_2 & & & & & \\ 0 & 0 & 1 & -c_1 & & & & \\ -c_2 & 0 & 2c_2 & 1 & -c_2 & & & \\ & 0 & 0 & 0 & 1 & -c_1 & & \\ & & -c_2 & 0 & 2c_2 & 1 & -c_2 & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix} 1 & c_1 & & & & & & \\ -2c_2 & 1 & c_2 & & & & & \\ 0 & 0 & 1 & c_1 & & & & \\ c_2 & 0 & -2c_2 & 1 & c_2 & & & \\ & 0 & 0 & 0 & 1 & c_1 & & \\ & & c_2 & 0 & -2c_2 & 1 & c_2 & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix},$$

with

$$c_1 = \tfrac{1}{2}h, \qquad c_2 = \frac{v^2}{2a^2}h,$$

and all matrix elements not shown being equal to zero. The equation $\mathbf{Au}(t+h) = \mathbf{Bu}(t)$ has precisely the form $\mathbf{Ax} = \mathbf{v}$ of the simultaneous equation problems we studied in Chapter 6 and can be solved using the same methods.

b) Write a program to perform a single step of the Crank–Nicolson method for this problem with initial conditions as above and using $N = 100$ spatial slices with $a = L/N$. You will have to perform the following steps. First, given the vector $\mathbf{u}(0)$ at $t = 0$, multiply by the matrix $\mathbf{B}$ to get a vector $\mathbf{v} = \mathbf{Bu}$. Because of the banded form of $\mathbf{B}$ this is fairly simple. For every even i (starting from zero in Python fashion) we have

$$v_i = u_i + c_1 u_{i+1}, \qquad v_{i+1} = u_{i+1} + c_2(u_{i-2} + u_{i+2} - 2u_i).$$

You will also need to choose a value for the time-step h. A reasonable choice is $h = 10^{-5}$ s.

Now solve the linear system $\mathbf{Au} = \mathbf{v}$ to get the new value of $\mathbf{u}$ at the next time-step. You could do this using a standard linear equation solver like the function `solve` in `numpy.linalg`, but since $\mathbf{A}$ is banded a better approach is to use the fast solver for banded matrices given in Appendix C, which can be imported from the file `banded.py` (which you can find in the online resources).

Once you have the code in place to perform a single step of the calculation, extend your program to perform repeated steps and thus solve for $\mathbf{u}$ and hence the displacement $\phi(x,t)$ at a sequence of times a separation h apart. Note that the matrix $\mathbf{A}$ is independent of time, so it does not change from one step to another. You can set it up once and then keep on reusing it for every step.

c) Extend your program to make an animation of the solution using the `qdraw` package. There are various ways you could do this. A simple one would be to just place a small circle at each grid point with vertical position representing the value of ϕ. A more sophisticated approach would be to use the `line` object from `qdraw`—see Appendix B for details. You will probably need to scale the values of ϕ by an additional constant to make the displacement of the string visible on the screen. (If you measure your x position in meters then a scale factor of 100 or more will probably be needed.)

Run your program for at least one second of simulated time. You should find that the vibration of the string is stable, neither decaying away nor growing uncontrollably as in Fig. 9.11.

Exercise 9.11: What would the equivalent be in three dimensions of Eq. (9.9) on page 402?

Exercise 9.12: The relaxation method for ordinary differential equations:

There is no reason why the relaxation method must be restricted to the solution of differential equations with two or more independent variables. It can also be applied to those with one independent variable, i.e., to ordinary differential equations. In this context, as with partial differential equations, it is a technique for solving boundary value problems, which are less common with ordinary differential equations but do occur—we discussed them in Section 8.6.

Consider the problem we looked at in Example 8.8 on page 385, in which a ball of mass $m = 1\,\text{kg}$ is thrown from height $x = 0$ into the air and lands back at $x = 0$ ten seconds later. The problem is to calculate the trajectory of the ball, but we cannot do it using initial value methods like the ordinary Runge–Kutta method because we are not told the initial velocity of the ball. One approach to finding a solution is the shooting method of Section 8.6.1. Another is the relaxation method.

Ignoring friction effects, the trajectory is the solution of the ordinary differential equation

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -g,$$

where g is the acceleration due to gravity.

a) Replacing the second derivative in this equation with its finite-difference approximation, Eq. (5.155), derive a relaxation-method equation for solving this problem on a time-like "grid" of points with separation h.

b) Taking the boundary conditions to be that $x = 0$ at $t = 0$ and $t = 10$, write a program to solve for the height of the ball as a function of time using the relaxation method with 100 points and make a plot of the result from $t = 0$ to $t = 10$. Run the calculation until the answers change by 10^{-6} or less at every point on each step.

Note that, unlike the shooting method, the relaxation method does not give us the initial value of the velocity needed to achieve the required solution. It gives us only the solution itself, although one could get an approximation to the initial velocity by calculating a numerical derivative of the solution at time $t = 0$. On balance, however, the relaxation method for ordinary differential equations is most useful when one wants to know the details of the solution itself, rather than the initial conditions needed to achieve it.

Exercise 9.13: The Schrödinger equation and the Crank–Nicolson method:

Perhaps the most important partial differential equation for physicists is the Schrödinger equation. This exercise uses the Crank–Nicolson method to solve the full time-dependent Schrödinger equation and hence develop a picture of how a wavefunction evolves over time. The following exercise, Exercise 9.14, solves the same problem again, but using the spectral method.

We will look at the Schrödinger equation in one dimension. The techniques for calculating solutions in two or three dimensions are basically the same as for one dimension, but the calculations take much longer, so in the interests of speed we will stick with one dimension. In one dimension the Schrödinger equation for a particle of mass M with no potential energy reads

$$-\frac{\hbar^2}{2M}\frac{\partial^2\psi}{\partial x^2} = \mathrm{i}\hbar\frac{\partial\psi}{\partial t}.$$

For simplicity, let us put our particle in a box with impenetrable walls, so that we only have to solve the equation in a finite space. The box forces the wavefunction ψ to be zero at the walls, which we will put at $x = 0$ and $x = D$.

Replacing the second derivative in the Schrödinger equation with a finite difference and applying Euler's method, we get the FTCS equation

$$\psi(x, t+h) = \psi(x,t) + h\frac{\mathrm{i}\hbar}{2ma^2}\bigl[\psi(x+a,t) + \psi(x-a,t) - 2\psi(x,t)\bigr],$$

where a is the spacing of the spatial grid points and h is the size of the time-step. (Be careful not to confuse the time-step h with Planck's constant $\hbar$.) Performing a similar step in reverse, we get the implicit equation

$$\psi(x,t+h) - h\frac{\mathrm{i}\hbar}{2ma^2}\bigl[\psi(x+a,t+h) + \psi(x-a,t+h) - 2\psi(x,t+h)\bigr] = \psi(x,t).$$

And taking the average of these two, we get the Crank–Nicolson equation for the Schrödinger equation:

$$\begin{aligned}\psi(x,t+h) - h\frac{\mathrm{i}\hbar}{4ma^2}&\bigl[\psi(x+a,t+h) + \psi(x-a,t+h) - 2\psi(x,t+h)\bigr]\\ &= \psi(x,t) + h\frac{\mathrm{i}\hbar}{4ma^2}\bigl[\psi(x+a,t) + \psi(x-a,t) - 2\psi(x,t)\bigr].\end{aligned}$$

This gives us a set of simultaneous equations, one for each grid point.

The boundary conditions on our problem tell us that $\psi = 0$ at $x = 0$ and $x = D$ for all t. In between these points we have grid points at a, $2a$, $3a$, and so forth. Let us arrange the values of ψ at these interior points into a vector

$$\boldsymbol{\psi}(t) = \begin{pmatrix}\psi(a,t)\\ \psi(2a,t)\\ \psi(3a,t)\\ \vdots\end{pmatrix}.$$

Then the Crank–Nicolson equations can be written in the form

$$\mathbf{A}\boldsymbol{\psi}(t+h) = \mathbf{B}\boldsymbol{\psi}(t),$$

where the matrices $\mathbf{A}$ and $\mathbf{B}$ are both symmetric and tridiagonal:

$$\mathbf{A} = \begin{pmatrix}a_1 & a_2 & & & \\ a_2 & a_1 & a_2 & & \\ & a_2 & a_1 & a_2 & \\ & & a_2 & a_1 & \\ & & & & \ddots\end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix}b_1 & b_2 & & & \\ b_2 & b_1 & b_2 & & \\ & b_2 & b_1 & b_2 & \\ & & b_2 & b_1 & \\ & & & & \ddots\end{pmatrix},$$

with

$$a_1 = 1 + h\frac{\mathrm{i}\hbar}{2ma^2}, \qquad a_2 = -h\frac{\mathrm{i}\hbar}{4ma^2}, \qquad b_1 = 1 - h\frac{\mathrm{i}\hbar}{2ma^2}, \qquad b_2 = h\frac{\mathrm{i}\hbar}{4ma^2}.$$

(Note the different signs and the factors of 2 and 4 in the denominators.)

The equation $\mathbf{A}\boldsymbol{\psi}(t+h) = \mathbf{B}\boldsymbol{\psi}(t)$ has precisely the form $\mathbf{A}\mathbf{x} = \mathbf{v}$ of the simultaneous equation problems we studied in Chapter 6 and can be solved using the same methods. Specifically, since the matrix $\mathbf{A}$ is tridiagonal in this case, we can use the fast tridiagonal version of Gaussian elimination that we looked at in Section 6.1.6.

Consider an electron (mass $M = 9.109 \times 10^{-31}$ kg) in a box of length $D = 10^{-8}$ m. Suppose that at time $t = 0$ the wavefunction of the electron has the form

$$\psi(x,0) = \exp\left[-\frac{(x-x_0)^2}{2\sigma^2}\right]\mathrm{e}^{\mathrm{i}\kappa x},$$

where

$$x_0 = \frac{D}{2}, \qquad \sigma = 1 \times 10^{-10}\,\text{m}, \qquad \kappa = 5 \times 10^{10}\,\text{m}^{-1},$$

and $\psi = 0$ on the walls at $x = 0$ and $x = D$. (This expression for $\psi(x,0)$ is not normalized. There should really be an overall multiplying coefficient to ensure that the probability density for the electron integrates to unity. It is safe to drop the constant, however, because the Schrödinger equation is linear, so the constant cancels out on both sides of the equation and plays no part in the solution.)

a) Write a program to perform a single step of the Crank–Nicolson method for this electron, calculating the vector $\boldsymbol{\psi}(t)$ of values of the wavefunction, given the initial wavefunction above and using $N = 1000$ spatial slices with $a = D/N$. Your program will have to perform the following steps. First, given the vector $\boldsymbol{\psi}(0)$ at $t = 0$, you will have to multiply by the matrix $\mathbf{B}$ to get a vector $\mathbf{v} = \mathbf{B}\boldsymbol{\psi}$. Because of the tridiagonal form of $\mathbf{B}$, this is fairly simple. The ith component of $\mathbf{v}$ is given by

$$v_i = b_1\psi_i + b_2(\psi_{i+1} + \psi_{i-1}).$$

You will also need a value for the time-step h. A reasonable choice in the present case is $h = 10^{-18}$ s.

Second, you will have to solve the linear system $\mathbf{A}\mathbf{x} = \mathbf{v}$ for $\mathbf{x}$, which gives you the new value of $\boldsymbol{\psi}$. You could do this using a standard linear equation solver like the function `solve` in `numpy.linalg`, but since the matrix $\mathbf{A}$ is tridiagonal a better approach is to use the fast solver for banded matrices given in Appendix C, which can be imported from the file `banded.py` (which you can find in the online resources). This solver works fine with complex-valued arrays, which you will need to use for the wavefunction $\boldsymbol{\psi}$ and the matrix $\mathbf{A}$.

b) Once you have the code in place to perform a single step of the calculation, extend your program to perform repeated steps and hence solve for ψ at a sequence of times a separation h apart. Note that the matrix $\mathbf{A}$ is independent of time, so it doesn't change from one step to another. You can set up the matrix just once and then keep on reusing it for every step.

c) Extend your program to make an animation of the solution by displaying the real part of the wavefunction at each time-step. There are various ways you could do the animation. A simple one would be to just place a small circle at each grid point with vertical position representing the value of the real part of the wavefunction. A more sophisticated approach would be to use the `line` object from the `qdraw` package—see Appendix B. You will need to scale the values of the wavefunction by an additional constant to make them a reasonable size on the screen. Multiplying by 10^{-9} makes them about the same size as the system itself (which is $D = 10^{-8}$) and hence makes the horizontal and vertical scales in the animation window comparable.

d) Run your animation for a while and describe what you see. Write a few sentences explaining in physics terms what is going on in the system.

Exercise 9.14: The Schrödinger equation and the spectral method:

This exercise uses the spectral method to solve the time-dependent Schödinger equation

$$-\frac{\hbar^2}{2M}\frac{\partial^2\psi}{\partial x^2} = \mathrm{i}\hbar\frac{\partial\psi}{\partial t}$$

for the same system as in Exercise 9.13, a single particle in one dimension in a box of length D with impenetrable walls. Following a similar argument to Section 9.3.4, we look for solutions of the form

$$\psi(x,t) = \rho(x)\,\mathrm{e}^{-\mathrm{i}Et/\hbar}.$$

Substituting into the Schrödinger equation, this gives us the time-independent equation

$$-\frac{\hbar^2}{2M}\frac{\mathrm{d}^2\rho}{\mathrm{d}x^2} = E\rho(x).$$

Now dividing the box into a one-dimensional "grid" of points with separation a, defining ρ_i to be the value of $\rho(x)$ at point i, and making use of the finite difference formula for the second derivative in its matrix form, we have

$$\frac{1}{a^2}\mathbf{L}\boldsymbol{\rho} = -\frac{2ME}{\hbar^2}\boldsymbol{\rho},$$

where $\boldsymbol{\rho}$ is the vector with elements ρ_i and $\mathbf{L}$ is the one-dimensional graph Laplacian matrix

$$\mathbf{L} = \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \end{pmatrix}.$$

In other words, $\boldsymbol{\rho}$ is an eigenvector of $\mathbf{L}$ and the energies E of the system are proportional to the eigenvalues.

As in Exercise 9.13, we consider an electron (mass $M = 9.109 \times 10^{-31}\,\mathrm{kg}$) in a box of length $D = 10^{-8}\,\mathrm{m}$. At time $t = 0$ the wavefunction of the electron has the form

$$\psi(x,0) = \exp\left[-\frac{(x-x_0)^2}{2\sigma^2}\right]\mathrm{e}^{\mathrm{i}\kappa x},$$

where

$$x_0 = \frac{D}{2}, \qquad \sigma = 1 \times 10^{-10}\,\mathrm{m}, \qquad \kappa = 5 \times 10^{10}\,\mathrm{m}^{-1},$$

and $\psi = 0$ on the walls at $x = 0$ and $x = D$.

a) Using $N = 1000$ spatial points, write a program to calculate the eigenvectors $\boldsymbol{\rho}_n$ for $n = 1 \ldots N$ and the corresponding energies E_n.

By analogy with Eq. (9.85) for the wave equation, a general solution of the Schrödinger equation on the grid points can now be written in vector form as a linear combination

$$\boldsymbol{\psi}(t) = \sum_{n=1}^{N} b_n \boldsymbol{\rho}_n \mathrm{e}^{-\mathrm{i}E_n t/\hbar},$$

for suitable values of the complex coefficients b_n. Since the Schrödinger equation is a first-order equation, we only need a single initial condition to fix the coefficients (unlike the wave equation, for which we need two). At time $t = 0$ we have

$$\boldsymbol{\psi}(0) = \sum_{n=1}^{N} b_n \boldsymbol{\rho}_n,$$

and, taking the dot product with the eigenvector $\boldsymbol{\rho}_m$ (and assuming the eigenvectors are properly normalized), we have

$$\boldsymbol{\rho}_m \cdot \boldsymbol{\psi}(0) = \sum_{n=1}^{N} b_n \, \boldsymbol{\rho}_m \cdot \boldsymbol{\rho}_n = b_m,$$

where we have used the fact that the eigenvectors are orthonormal, so $\boldsymbol{\rho}_m \cdot \boldsymbol{\rho}_n = \delta_{mn}$.

b) Add code to your program to calculate the values of the coefficients b_m.

c) Using these values in the general solution above, extend your program to calculate the real part of the wavefunction $\psi(x,t)$ on the grid points at an arbitrary time t. Test your program by making a graph of the wavefunction at time $t = 10^{-16}$ s.

d) Extend your program further to make an animation of the wavefunction over time, similar to that described in part (b) of Exercise 9.13. A suitable time interval for each frame of the animation is 10^{-18} s.

e) Run your animation for a while and describe what you see. In a few sentences, what is going on in the system?

Hint: Your program will run faster if you make use of Python's ability to do fast matrix arithmetic with arrays. Write the expression for the vector $\boldsymbol{\psi}(t)$ above in matrix form and then use the function `dot` from `numpy` to evaluate it quickly.