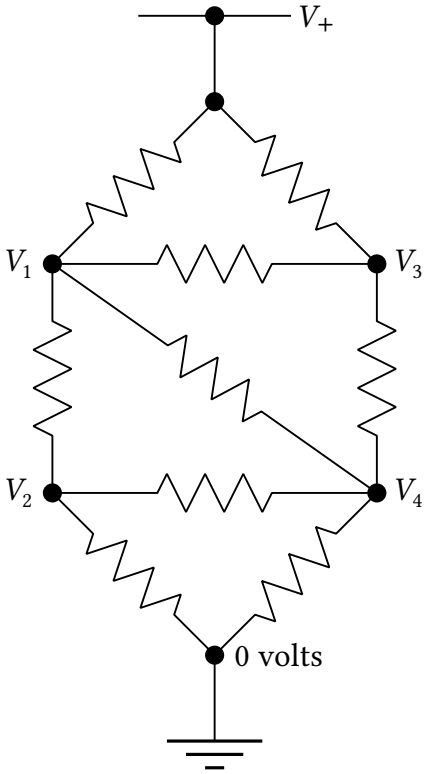
Computational Physics, 2nd edition

Exercises for Chapter 6

Exercise 6.1: A circuit of resistors

Consider the circuit of resistors shown here:



All the resistors have the same resistance R. The power rail at the top is at voltage $V_+ = 5\,\text{V}$. What are the other four voltages, V_1 to V_4?

To answer this question we use Ohm's law and the Kirchhoff current law, which says that the total net current flow out of (or into) any junction in a circuit must be zero. Thus for the junction at voltage V_1, for instance, we have

$$\frac{V_1 - V_2}{R} + \frac{V_1 - V_3}{R} + \frac{V_1 - V_4}{R} + \frac{V_1 - V_+}{R} = 0,$$

or equivalently

$$4V_1 - V_2 - V_3 - V_4 = V_+.$$

a) Write similar equations for the other three junctions with unknown voltages.

b) Write a program to solve the four resulting equations using Gaussian elimination and hence find the four voltages. (If you want you can modify a program you already have, such as the program `gausselim.py` in Example 6.1, which can be found in the online resources).

Exercise 6.2:

a) Modify the program `gausselim.py` in Example 6.1 to incorporate partial pivoting (or you can write your own program from scratch if you prefer). Run your program and demonstrate that it gives the same answers as the original program when applied to Eq. (6.1)

b) Use your program to solve the equations in (6.17) and show that it can find the solution to these as well, even though Gaussian elimination without pivoting fails.

Exercise 6.3: LU decomposition

This exercise invites you to write your own program to solve simultaneous equations using LU decomposition.

a) Starting, if you wish, with the program for Gaussian elimination in Example 6.1 on page 219, write a Python function that calculates the LU decomposition of a matrix. The calculation is same as that for Gaussian elimination, except that at each step you need to extract the appropriate elements of the matrix and assemble them to form the lower diagonal matrix $\mathbf{L}$ of Eq. (6.32). Test your function by calculating the LU decomposition of the matrix from Eq. (6.2), then multiplying $\mathbf{L}$ and $\mathbf{U}$ and verifying that you recover the original matrix once more. (Hint: Recall that matrix multiplication can be done with the function `dot` from `numpy`—see Section 2.4.4.)

b) Build on your LU decomposition function to create a complete program to solve Eq. (6.2) by performing a double backsubstitution as described above.

c) Solve the same equations using the function `solve` from `numpy.linalg` and verify that you get the same answer.

d) If you are feeling ambitious, try your hand at LU decomposition with partial pivoting. Partial pivoting works in the same way for LU decomposition as it does for Gaussian elimination, swapping rows to get the largest diagonal element as explained in Section 6.1.3, but the extension to LU decomposition requires two additional steps. First, every time you swap two rows you also have to swap the same rows in the matrix $\mathbf{L}$. Second, when you use your LU decomposition to solve a set of equations $\mathbf{A}\mathbf{x} = \mathbf{v}$ you will also need to perform the same sequence of swaps on the vector $\mathbf{v}$ on the right-hand side. This means you need to record the swaps as you are doing the decomposition so that you can recreate them later. The simplest way to do this is to set up a list or array in which the value of the ith element records the row you swapped with on the ith step of the process. For instance, if you swapped the first row with the second then the second with the fourth, the first two elements of the list would be 2 and 4. Solving a set of equations for given $\mathbf{v}$ involves first performing the required sequence of swaps on the elements of $\mathbf{v}$ then performing a double backsubstitution as usual. (In ordinary Gaussian elimination with pivoting, one swaps the elements of $\mathbf{v}$ as the algorithm proceeds, rather than all at once, but the result is the same either way.)

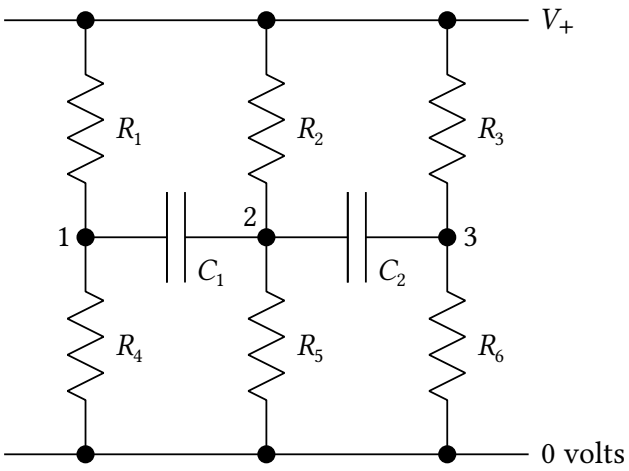
   Modify the function you wrote for part (a) to perform LU decomposition with partial pivoting. The function should return the matrices $\mathbf{L}$ and $\mathbf{U}$ for the LU decomposition of the swapped matrix, plus a list of the swaps made. Then modify the rest of your program to solve equations of the

form $\mathbf{Ax} = \mathbf{v}$ using LU decomposition with pivoting. Test your program on the example from Eq. (6.17), which cannot be solved without pivoting because of the zero in the first element of the matrix. Check your results against a solution of the same equations using the `solve` function from `numpy.linalg`.

Under the hood, the `solve` function also uses LU decomposition with partial pivoting. There is nothing wrong with using this function—it is well written, fast, and convenient. But it does nothing you haven't already done yourself if you have solved this exercise.

Exercise 6.4: Write a program to solve the resistor network problem of Exercise 6.1 on page 220 using the function `solve` from `numpy.linalg`. If you also did Exercise 6.1, check that you get the same answer both times.

Exercise 6.5: Here is a more complicated circuit problem:



The voltage V_+ is time-varying and sinusoidal of the form $V_+ = x_+ e^{i\omega t}$ with x_+ a constant. The resistors in the circuit can be treated using Ohm's law as usual. For the capacitors the charge Q and voltage V across them are related by the capacitor law $Q = CV$, where C is the capacitance. Differentiating both sides of this expression gives the current I flowing in on one side of the capacitor and out on the other:

$$I = \frac{dQ}{dt} = C\frac{dV}{dt}.$$

a) Assuming the voltages at the points labeled 1, 2, and 3 are of the form $V_1 = x_1 e^{i\omega t}$, $V_2 = x_2 e^{i\omega t}$, and $V_3 = x_3 e^{i\omega t}$, apply Kirchhoff's law at each of the three points, along with Ohm's law and the capacitor law, to show that the constants x_1, x_2, and x_3 satisfy the equations

$$\begin{aligned}
\left(\frac{1}{R_1} + \frac{1}{R_4} + i\omega C_1\right)x_1 - i\omega C_1 x_2 &= \frac{x_+}{R_1}, \\
-i\omega C_1 x_1 + \left(\frac{1}{R_2} + \frac{1}{R_5} + i\omega C_1 + i\omega C_2\right)x_2 - i\omega C_2 x_3 &= \frac{x_+}{R_2}, \\
-i\omega C_2 x_2 + \left(\frac{1}{R_3} + \frac{1}{R_6} + i\omega C_2\right)x_3 &= \frac{x_+}{R_3}.
\end{aligned}$$

b) Write a program to solve for x_1, x_2, and x_3 when

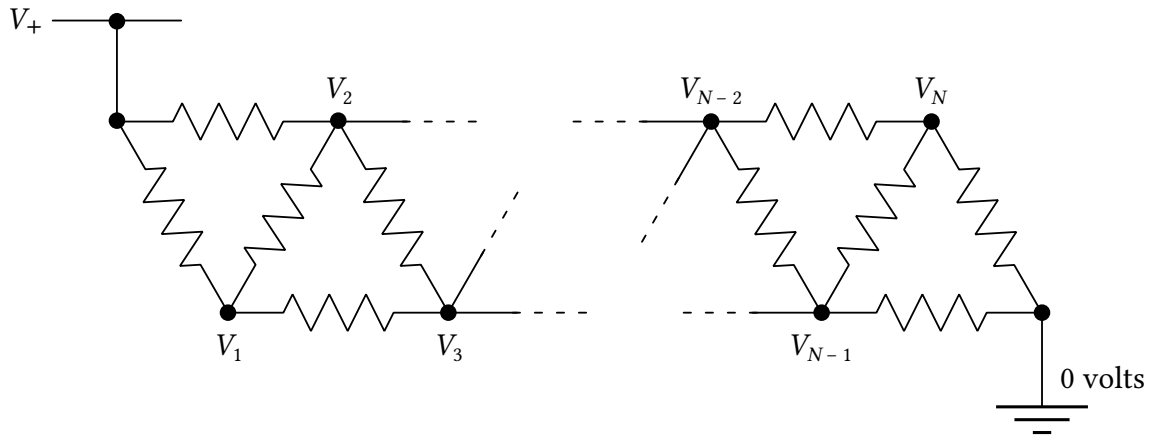
$$\begin{aligned} R_1 &= R_3 = R_5 = 1\,\text{k}\Omega, \\ R_2 &= R_4 = R_6 = 2\,\text{k}\Omega, \\ C_1 &= 1\,\mu\text{F}, \qquad C_2 = 0.5\,\mu\text{F}, \\ x_+ &= 3\,\text{V}, \qquad \omega = 1000\,\text{s}^{-1}. \end{aligned}$$

Notice that the matrix for this problem has complex elements. You will need to define a complex array to hold it, but you can still use the `solve` function just as before to solve the equations—it works with either real or complex arguments. Using your solution have your program calculate and print the amplitudes of the three voltages V_1, V_2, and V_3 and their phases in degrees. (Hint: You may find the functions `polar` or `phase` in the `cmath` package useful. If `z` is a complex number then "`r,theta = polar(z)`" will return the modulus and phase (in radians) of `z` and "`theta = phase(z)`" will return the phase alone.)

Exercise 6.6: Starting with either the program `springs.py` on page 235 or `springsb.py` on page 237, remove the code that makes a graph of the results and replace it with code that creates an animation of the masses as they vibrate back and forth, their displacements relative to their resting positions being given by the real part of Eq. (6.55). For clarity, assume that the resting positions are two units apart in a horizontal line. At a minimum your animation should show each of the individual masses, perhaps as small circles.

Exercise 6.7: A chain of resistors

Consider a long chain of resistors wired up like this:



All the resistors have the same resistance R. The power rail at the top is at voltage $V_+ = 5$V. The problem is to find the voltages $V_1 \ldots V_N$ at the internal points in the circuit.

a) Using Ohm's law and the Kirchhoff current law, which says that the total net current flow out of (or into) any junction in a circuit must be zero, show that the voltages $V_1 \ldots V_N$ satisfy the

equations

$$\begin{aligned} 3V_1 - V_2 - V_3 &= V_+, \\ -V_1 + 4V_2 - V_3 - V_4 &= V_+, \\ &\vdots \\ -V_{i-2} - V_{i-1} + 4V_i - V_{i+1} - V_{i+2} &= 0, \\ &\vdots \\ -V_{N-3} - V_{N-2} + 4V_{N-1} - V_N &= 0, \\ -V_{N-2} - V_{N-1} + 3V_N &= 0. \end{aligned}$$

Express these equations in vector form $\mathbf{Av} = \mathbf{w}$ and find the values of the matrix $\mathbf{A}$ and the vector $\mathbf{w}$.

b) Write a program to solve for the values of the V_i when there are $N = 6$ internal junctions with unknown voltages. Hint: All the values of V_i should lie between zero and 5V. If they don't, something is wrong.

c) Now repeat your calculation for the case where there are $N = 10\,000$ internal junctions. This part is not possible using standard tools like the `solve` function. You need to make use of the fact that the matrix $\mathbf{A}$ is banded and use the Thomas algorith, or use the `banded` function from the file `banded.py`, introduced in Example 6.2.

Exercise 6.8: Consider the symmetric matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 2 & 2 & 1 \\ 1 & 0 & 0 & 2 & 3 \\ 2 & 0 & 3 & 4 & 0 \\ 2 & 2 & 4 & 3 & 2 \\ 1 & 3 & 0 & 2 & 0 \end{pmatrix}$$

a) Write a program to find the most positive eigenvalue of this matrix using the power method. Run the calculation until the eigenvalue stops changing in the first six decimal places.

b) Extend your program to also find the most negative eigenvalue.

c) Write another program to find the eigenvalue of smallest magnitude using inverse iteration.

d) Extend this program to find the eigenvalue closest in value to 4 using shifted inverse iteration.

Hints: You may find the `numpy` functions `dot` and `identity` useful. The first performs matrix multiplication and the second creates an identity matrix. For solving the linear system in Eq. (6.76) one should really perform an LU decomposition of $\mathbf{A}$ and then use backsubstitution, but for a matrix this small it is fine to just use the `solve` function repeatedly. (If you previously did Exercise 6.3 and wrote your own LU decomposition code, then you could also use that, although note that pivoting will be required, since the first element of the matrix is zero.)

Exercise 6.9: The QR algorithm

This exercise gives you the opportunity to write a program to calculate the eigenvalues and eigenvectors of a real symmetric matrix using the QR algorithm. The first challenge is to write a function that finds the QR decomposition of a matrix. Then we will use that decomposition to find the eigenvalues.

The QR decomposition expresses a real square matrix $\mathbf{A}$ in the form $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper-triangular matrix. Given an $N \times N$ matrix $\mathbf{A}$ we can compute the QR decomposition as follows.

Let us think of the matrix as a set of N column vectors $\mathbf{a}_0 \ldots \mathbf{a}_{N-1}$ thus:

$$\mathbf{A} = \begin{pmatrix} | & | & | & \cdots \\ \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \cdots \\ | & | & | & \cdots \end{pmatrix},$$

where we have numbered the vectors in Python fashion, starting from zero, which will be convenient when writing the program. We now define two new sets of vectors $\mathbf{u}_0 \ldots \mathbf{u}_{N-1}$ and $\mathbf{q}_0 \ldots \mathbf{q}_{N-1}$ as follows:

$$\begin{aligned} \mathbf{u}_0 &= \mathbf{a}_0, & \mathbf{q}_0 &= \frac{\mathbf{u}_0}{|\mathbf{u}_0|}, \\ \mathbf{u}_1 &= \mathbf{a}_1 - (\mathbf{q}_0 \cdot \mathbf{a}_1)\mathbf{q}_0, & \mathbf{q}_1 &= \frac{\mathbf{u}_1}{|\mathbf{u}_1|}, \\ \mathbf{u}_2 &= \mathbf{a}_2 - (\mathbf{q}_0 \cdot \mathbf{a}_2)\mathbf{q}_0 - (\mathbf{q}_1 \cdot \mathbf{a}_2)\mathbf{q}_1, & \mathbf{q}_2 &= \frac{\mathbf{u}_2}{|\mathbf{u}_2|}, \end{aligned}$$

and so forth, where $|\mathbf{u}|$ denotes the magnitude of vector $\mathbf{u}$. The general formulas for calculating $\mathbf{u}_i$ and $\mathbf{q}_i$ are

$$\mathbf{u}_i = \mathbf{a}_i - \sum_{j=0}^{i-1} (\mathbf{q}_j \cdot \mathbf{a}_i)\mathbf{q}_j, \qquad \mathbf{q}_i = \frac{\mathbf{u}_i}{|\mathbf{u}_i|}.$$

a) Show, by induction or otherwise, that the vectors $\mathbf{q}_i$ are orthonormal, i.e., that they satisfy

$$\mathbf{q}_i \cdot \mathbf{q}_j = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

Now, rearranging the definitions of the vectors, we have

$$\begin{aligned} \mathbf{a}_0 &= |\mathbf{u}_0|\, \mathbf{q}_0, \\ \mathbf{a}_1 &= |\mathbf{u}_1|\, \mathbf{q}_1 + (\mathbf{q}_0 \cdot \mathbf{a}_1)\mathbf{q}_0, \\ \mathbf{a}_2 &= |\mathbf{u}_2|\, \mathbf{q}_2 + (\mathbf{q}_0 \cdot \mathbf{a}_2)\mathbf{q}_0 + (\mathbf{q}_1 \cdot \mathbf{a}_2)\mathbf{q}_1, \end{aligned}$$

and so on. Or we can group the vectors $\mathbf{q}_i$ together as the columns of a matrix and write all of these equations as a single matrix equation

$$\mathbf{A} = \begin{pmatrix} | & | & | & \cdots \\ \mathbf{a}_0 & \mathbf{a}_1 & \mathbf{a}_2 & \cdots \\ | & | & | & \cdots \end{pmatrix} = \begin{pmatrix} | & | & | & \cdots \\ \mathbf{q}_0 & \mathbf{q}_1 & \mathbf{q}_2 & \cdots \\ | & | & | & \cdots \end{pmatrix} \begin{pmatrix} |\mathbf{u}_0| & \mathbf{q}_0 \cdot \mathbf{a}_1 & \mathbf{q}_0 \cdot \mathbf{a}_2 & \cdots \\ 0 & |\mathbf{u}_1| & \mathbf{q}_1 \cdot \mathbf{a}_2 & \cdots \\ 0 & 0 & |\mathbf{u}_2| & \cdots \end{pmatrix}.$$

(If this looks complicated it may be worth multiplying out the matrices on the right to verify for yourself that you get the correct expressions for the $\mathbf{a}_i$.)

Observe now that the first matrix on the right-hand side of this equation, the matrix with columns $\mathbf{q}_i$, is orthogonal, because the vectors $\mathbf{q}_i$ are orthonormal, and the second matrix is upper triangular. In other words, we have found the QR decomposition $\mathbf{A} = \mathbf{QR}$. The matrices $\mathbf{Q}$ and $\mathbf{R}$ are

$$\mathbf{Q} = \begin{pmatrix} | & | & | & \cdots \\ \mathbf{q}_0 & \mathbf{q}_1 & \mathbf{q}_2 & \cdots \\ | & | & | & \cdots \end{pmatrix}, \qquad \mathbf{R} = \begin{pmatrix} |\mathbf{u}_0| & \mathbf{q}_0\cdot\mathbf{a}_1 & \mathbf{q}_0\cdot\mathbf{a}_2 & \cdots \\ 0 & |\mathbf{u}_1| & \mathbf{q}_1\cdot\mathbf{a}_2 & \cdots \\ 0 & 0 & |\mathbf{u}_2| & \cdots \end{pmatrix}.$$

b) Write a Python function that takes as its argument a real square matrix $\mathbf{A}$ and returns the two matrices $\mathbf{Q}$ and $\mathbf{R}$ that form its QR decomposition. As a test case, try out your function on the matrix

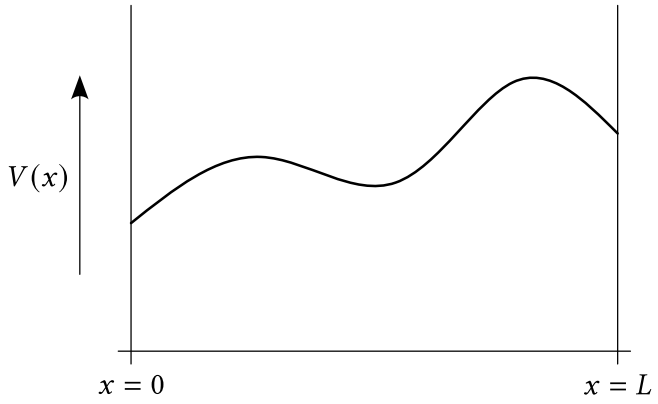
$$\mathbf{A} = \begin{pmatrix} 1 & 4 & 8 & 4 \\ 4 & 2 & 3 & 7 \\ 8 & 3 & 6 & 9 \\ 4 & 7 & 9 & 2 \end{pmatrix}.$$

Check the results by multiplying $\mathbf{Q}$ and $\mathbf{R}$ together to recover the original matrix $\mathbf{A}$ again.

c) Using your function, write a complete program to calculate the eigenvalues and eigenvectors of a real symmetric matrix using the QR algorithm. Continue the calculation until the eigenvalues change by less than 10^{-6} on each iteration. Test your program on the example matrix above. You should find that the eigenvalues are 21, 1, −3, and −8.

Exercise 6.10: Asymmetric quantum well

Quantum mechanics can be formulated as a matrix problem and solved on a computer using linear algebra methods. Suppose, for example, we have a particle of mass M in a one-dimensional quantum well of width L, but not a square well like the ones you commonly find discussed in textbooks. Instead suppose that the potential $V(x)$ varies somehow inside the well:



We cannot solve such problems analytically in general, but we can solve them on the computer.

In a pure state of energy E, the spatial part $\psi(x)$ of the wavefunction obeys the time-independent Schrödinger equation $\hat{H}\psi(x) = E\psi(x)$, where the Hamiltonian operator $\hat{H}$ is given by

$$\hat{H} = -\frac{\hbar^2}{2M}\frac{\mathrm{d}^2}{\mathrm{d}x^2} + V(x).$$

For simplicity, assume that the walls of the well are infinitely high, so that the wavefunction is zero outside the well, which means it must *go to* zero at $x = 0$ and $x = L$. In that case, the wavefunction can be expressed as a Fourier sine series thus:

$$\psi(x) = \sum_{n=1}^{\infty} \psi_n \sin \frac{\pi n x}{L},$$

where $\psi_1, \psi_2, \ldots$ are the Fourier coefficients.

a) Noting that, for positive integers m, n

$$\int_0^L \sin \frac{\pi m x}{L} \sin \frac{\pi n x}{L} \, \mathrm{d}x = \begin{cases} L/2 & \text{if } m = n, \\ 0 & \text{otherwise,} \end{cases}$$

show that the Schrödinger equation $\hat{H}\psi = E\psi$ implies that

$$\sum_{n=1}^{\infty} \psi_n \int_0^L \sin \frac{\pi m x}{L} \hat{H} \sin \frac{\pi n x}{L} \, \mathrm{d}x = \frac{1}{2} L E \psi_m.$$

Hence, defining a matrix $\mathbf{H}$ with elements

$$\begin{aligned} H_{mn} &= \frac{2}{L} \int_0^L \sin \frac{\pi m x}{L} \hat{H} \sin \frac{\pi n x}{L} \, \mathrm{d}x \\ &= \frac{2}{L} \int_0^L \sin \frac{\pi m x}{L} \left[-\frac{\hbar^2}{2M} \frac{\mathrm{d}^2}{\mathrm{d}x^2} + V(x) \right] \sin \frac{\pi n x}{L} \, \mathrm{d}x, \end{aligned}$$

show that Schrödinger's equation can be written in matrix form as $\mathbf{H}\boldsymbol{\psi} = E\boldsymbol{\psi}$, where $\boldsymbol{\psi}$ is the vector $(\psi_1, \psi_2, \ldots)$. Thus $\boldsymbol{\psi}$ is an eigenvector of the *Hamiltonian matrix* $\mathbf{H}$ with eigenvalue E. If we can calculate the eigenvalues of this matrix, then we know the allowed energies of the particle in the well.

b) For the case $V(x) = ax/L$, evaluate the integral in H_{mn} analytically and so find a general expression for the matrix element H_{mn}. Show that the matrix is real and symmetric. You will probably find it useful to know that

$$\int_0^L x \sin \frac{\pi m x}{L} \sin \frac{\pi n x}{L} \, \mathrm{d}x = \begin{cases} 0 & \text{if } m \neq n \text{ and } m, n \text{ are both even or odd,} \\ -\left(\dfrac{2L}{\pi}\right)^2 \dfrac{mn}{(m^2 - n^2)^2} & \text{if } m \neq n \text{ and one is even, one is odd,} \\ L^2/4 & \text{if } m = n. \end{cases}$$

Write a Python program to evaluate your expression for H_{mn} for arbitrary m and n when the particle in the well is an electron, the well has a width of 5 angstroms, and $a = 10$ eV. (The mass and charge of an electron are 9.1094×10^{-31} kg and 1.6022×10^{-19} C respectively.)

c) The matrix $\mathbf{H}$ is in theory infinitely large, so we cannot calculate all its eigenvalues. But we can get a pretty accurate solution for the first few of them by cutting off the matrix after the first few elements. Modify the program you wrote for part (b) above to create a 10×10 array of the elements of $\mathbf{H}$ up to $m, n = 10$. Calculate the eigenvalues of this matrix using the appropriate function from `numpy.linalg` and hence print out, in units of electron volts, the first ten energy

levels of the quantum well, within this approximation. You should find, for example, that the ground-state energy of the system is around 5.84 eV. Hint: Bear in mind that matrix indices in Python start at zero, while the indices in standard algebraic expressions, like those above, start at one. You will need to make allowances for this in your program.

d) Modify your program to use a 100×100 array instead and again calculate the first ten energy eigenvalues. Comparing with the values you calculated in part (c), what do you conclude about the accuracy of the calculation?

e) Modify your program once more to calculate the wavefunction $\psi(x)$ for the ground state and the first two excited states of the well. Use your results to make a graph with three curves showing the probability density $|\psi(x)|^2$ as a function of x in each of these three states. Pay special attention to the normalization of the wavefunction—it should satisfy the condition $\int_0^L |\psi(x)|^2 \, \mathrm{d}x = 1$.

Exercise 6.11: Consider the equation $x = 1 - \mathrm{e}^{-cx}$, where c is a known parameter and x is unknown. This equation arises in a variety of situations, including the physics of contact processes, mathematical models of epidemics, and the theory of random graphs.

a) Write a program to solve this equation for x using the relaxation method for the case $c = 2$. Calculate your solution to an accuracy of at least 10^{-6}.

b) Modify your program to calculate the solution for values of c from 0 to 3 in steps of 0.01 and make a plot of x as a function of c. You should see a clear transition from a regime in which $x = 0$ to a regime of nonzero x. This is another example of a phase transition. In physics this transition is known as the *percolation transition*; in epidemiology it is the *epidemic threshold*.

Exercise 6.12: Overrelaxation

If you did not already do Exercise 6.11, you should do it before this one.

The ordinary relaxation method involves iterating the equation $x' = f(x)$, starting from an initial guess, until it converges. As we have seen, this is often a fast and easy way to find solutions to non-linear equations. However, it is possible in some cases to make the method work even faster using the technique of *overrelaxation*.

Suppose our initial guess at the solution of a particular equation is, say, $x = 1$, and the final, true solution is $x = 5$. After the first step of the iterative process, we might then see a value of, say, $x = 3$. In the overrelaxation method, we observe this value and note that x is increasing, then we deliberately overshoot the calculated value, in the hope that this will get us closer to the final solution. In this case we might pass over $x = 3$ and go straight to a value of $x = 4$ perhaps, which is closer to the final solution of $x = 5$ and hence should get us to that solution quicker. The overrelaxation method provides a formula for performing this kind of overshooting in a controlled fashion and often, though not always, it does get us to our solution faster. In detail, it works as follows.

We can rewrite the equation $x' = f(x)$ in the form $x' = x + \Delta x$, where

$$\Delta x = x' - x = f(x) - x.$$

The overrelaxation method involves iterating the modified equation

$$x' = x + (1+\omega)\,\Delta x,$$

for some specified value of the constant ω (while keeping the definition of Δx the same). If ω is zero, then this is the same as the ordinary relaxation method, but for $\omega > 0$ the method takes the amount Δx by which x would have changed and changes it by a little more.

Using $\Delta x = f(x) - x$, we can also write x' as

$$x' = x + (1+\omega)[f(x) - x] = (1+\omega)f(x) - \omega x.$$

This is the form in which the overrelaxation method is usually applied. For the method to work, the value of ω must be chosen correctly, although there is some wiggle room. There is an optimal value, but other values close to it will usually give good results too. Unfortunately, there is no general theory that tells us what the optimal value is. Usually it is found by trial and error.

a) Derive an equivalent of Eq. (6.102) on page 257 for the overrelaxation method and hence show that the error on x', the equivalent of Eq. (6.103), is given by

$$\delta' \simeq \frac{x' - x}{1/[(1+\omega)f'(x) - \omega] - 1}.$$

b) Consider again the equation $x = 1 - \mathrm{e}^{-cx}$ that we solved in Exercise 6.11. Take the program you wrote for part (a) of that exercise, which solved the equation for the case $c = 2$, and modify it to print out the number of iterations it takes to converge to a solution accurate to 10^{-6}.

c) Now write a new program (or modify the previous one) to solve the same equation $x = 1 - \mathrm{e}^{-cx}$ for $c = 2$, again to an accuracy of 10^{-6}, but this time using overrelaxation. Have your program print out the answers it finds along with the number of iterations it took to find them. Experiment with different values of ω to see how fast you can get the method to converge. A value of $\omega = 0.5$ is a reasonable starting point. With some trial and error you should be able to get the calculation to converge at least twice as fast as the simple relaxation method, i.e., in about half as many iterations.

d) Are there any circumstances under which using a value $\omega < 0$ would help us find a solution faster than we can with the ordinary relaxation method? (Hint: The answer is yes, but why?)

Exercise 6.13: The biochemical process of *glycolysis*, the breakdown of glucose in the body to release energy, can be modeled by the equations

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -x + ay + x^2 y, \qquad \frac{\mathrm{d}y}{\mathrm{d}t} = b - ay - x^2 y.$$

Here x and y represent concentrations of two chemicals, ADP and F6P, and a and b are positive constants. One of the important features of nonlinear linear equations like these is their *stationary points*, meaning values of x and y at which the derivatives of both variables become zero simultaneously, so that the variables stop changing and become constant in time. Setting the derivatives to zero above, the stationary points of our glycolysis equations are solutions of

$$-x + ay + x^2 y = 0, \qquad b - ay - x^2 y = 0.$$

a) Demonstrate analytically that the solution of these equations is

$$x = b, \qquad y = \frac{b}{a + b^2}.$$

b) Show that the equations can be rearranged to read

$$x = y(a + x^2), \qquad y = \frac{b}{a + x^2}$$

and write a program to solve these for the stationary point using the relaxation method with $a = 1$ and $b = 2$. You should find that the method fails to converge to a solution in this case.

c) Find a different way to rearrange the equations such that when you apply the relaxation method again it now converges to a fixed point and gives a solution. Verify that the solution you get agrees with part (a).

Exercise 6.14: Wien's displacement constant

Planck's radiation law tells us that the intensity of radiation per unit area and per unit wavelength λ from a black body at temperature T is

$$I(\lambda) = \frac{2\pi hc^2\lambda^{-5}}{e^{hc/\lambda k_B T} - 1},$$

where h is Planck's constant, c is the speed of light, and k_B is Boltzmann's constant.

a) Show by differentiating that the wavelength λ at which the emitted radiation is strongest is the solution of the equation

$$5e^{-hc/\lambda k_B T} + \frac{hc}{\lambda k_B T} - 5 = 0.$$

Make the substitution $x = hc/\lambda k_B T$ and hence show that the wavelength of maximum radiation obeys the *Wien displacement law*:

$$\lambda = \frac{b}{T},$$

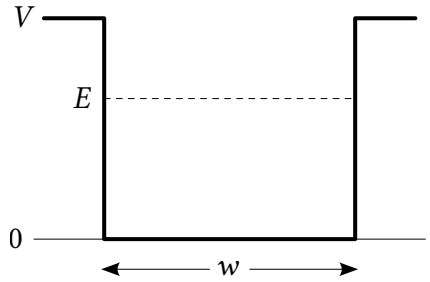
where the so-called *Wien displacement constant* is $b = hc/k_B x$, and x is the solution to the non-linear equation

$$5e^{-x} + x - 5 = 0.$$

b) Write a program to solve this equation to an accuracy of $\epsilon = 10^{-6}$ using the binary search method, and hence find a value for the displacement constant.

c) The displacement law is the basis for the method of *optical pyrometry*, a method for measuring the temperatures of objects by observing the color of the thermal radiation they emit. The method is commonly used to estimate the surface temperatures of astronomical bodies, such as the Sun. The wavelength peak in the Sun's emitted radiation falls at $\lambda = 502$ nm. From the equations above and your value of the displacement constant, estimate the surface temperature of the Sun.

Exercise 6.15: Consider a square potential well of width w, with walls of height V:



Using Schrödinger's equation, it can be shown that the allowed energies E of a single quantum particle of mass m trapped in the well are solutions of

$$\tan\sqrt{w^2mE/2\hbar^2} = \begin{cases} \sqrt{(V-E)/E} & \text{for the even numbered states,} \\ -\sqrt{E/(V-E)} & \text{for the odd numbered states,} \end{cases}$$

where the states are numbered starting from 0, with the ground state being state 0, the first excited state being state 1, and so forth.

a) For an electron (mass 9.1094×10^{-31} kg) in a well with $V = 20$ eV and $w = 1$ nm, write a Python program to plot the three quantities

$$y_1 = \tan\sqrt{w^2mE/2\hbar^2}, \qquad y_2 = \sqrt{\frac{V-E}{E}}, \qquad y_3 = -\sqrt{\frac{E}{V-E}},$$

on the same graph, as a function of E from $E = 0$ to $E = 20$ eV. From your plot make approximate estimates of the energies of the first six energy levels of the particle.

b) Write a second program to calculate the values of the first six energy levels in electron volts to an accuracy of 0.001 eV using binary search.

Exercise 6.16: The roots of a polynomial

Consider the sixth-order polynomial

$$P(x) = 924x^6 - 2772x^5 + 3150x^4 - 1680x^3 + 420x^2 - 42x + 1.$$

There is no general formula for the roots of a sixth-order polynomial, but one can find them easily enough using a computer.

a) Make a plot of $P(x)$ from $x = 0$ to $x = 1$ and by inspecting it find rough values for the six roots of the polynomial—the points at which the function is zero.

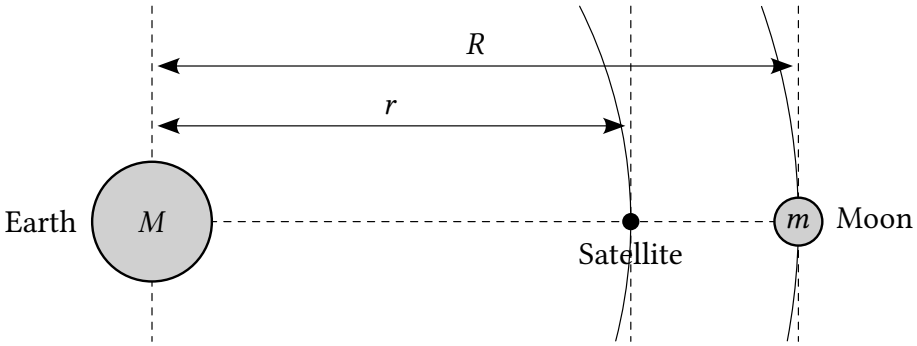
b) Write a Python program to solve for the positions of all six roots to at least ten decimal places of accuracy, using Newton's method.

Note that the polynomial in this example is just the sixth Legendre polynomial (mapped onto the interval from zero to one), so the calculation performed here is the same as finding the integration

points for 6-point Gaussian quadrature (see Section 5.6.2), and indeed Newton's method is the method of choice for calculating Gaussian quadrature points—see Appendix C, Section C1 on page 572.

Exercise 6.17: The Lagrange point

There is a magical point between the Earth and the Moon, called the L_1 Lagrange point, at which a satellite will orbit the Earth in perfect synchrony with the Moon, staying always in between the two. This works because the inward pull of the Earth and the outward pull of the Moon combine to create exactly the needed centripetal force that keeps the satellite in its orbit. The setup looks like this:



a) Assuming circular orbits, and assuming that the Earth is much more massive than either the Moon or the satellite, show that the distance r from the center of the Earth to the L_1 point satisfies

$$\frac{GM}{r^2} - \frac{Gm}{(R-r)^2} = \omega^2 r,$$

where R is the distance from the Earth to the Moon, M and m are the Earth and Moon masses, G is Newton's gravitational constant, and ω is the angular velocity of both the Moon and the satellite.

b) The equation above is a degree-five polynomial equation in r (also called a quintic equation). Such equations cannot be solved exactly in closed form, but it is straightforward to solve them numerically. Write a program that uses either Newton's method or the secant method to solve for the distance r from the Earth to the L_1 point. Compute a solution accurate to at least four significant figures.

The values of the various parameters are:

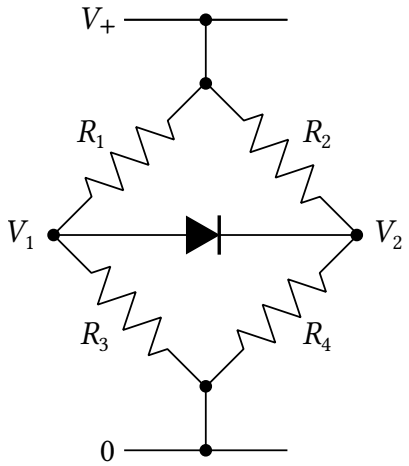
$$\begin{aligned} G &= 6.674 \times 10^{-11}\,\mathrm{m^3 kg^{-1} s^{-2}}, \\ M &= 5.974 \times 10^{24}\,\mathrm{kg}, \\ m &= 7.348 \times 10^{22}\,\mathrm{kg}, \\ R &= 3.844 \times 10^{8}\,\mathrm{m}, \\ \omega &= 2.662 \times 10^{-6}\,\mathrm{s^{-1}}. \end{aligned}$$

You will also need to choose a suitable starting value for r, or two starting values if you use the secant method.

Exercise 6.18: Nonlinear circuits

Exercise 6.1 used regular simultaneous equations to solve for the behavior of circuits of resistors. Resistors are linear—current is proportional to voltage—and the resulting equations we need to solve are therefore also linear and can be solved by standard matrix methods. Real circuits, however, often include nonlinear components. To solve for the behavior of these circuits we need to solve nonlinear equations.

Consider the following simple circuit, a variation on the classic Wheatstone bridge:



The resistors obey the normal Ohm law, but the diode obeys the diode equation:

$$I = I_0(e^{V/V_T} - 1),$$

where V is the voltage across the diode and I_0 and V_T are constants.

a) The Kirchhoff current law says that the total net current flowing into or out of every point in a circuit must be zero. Applying the law to voltage V_1 in the circuit above we get

$$\frac{V_1 - V_+}{R_1} + \frac{V_1}{R_2} + I_0\left[e^{(V_1-V_2)/V_T} - 1\right] = 0.$$

Derive the corresponding equation for voltage V_2.

b) Solve the two nonlinear equations for the voltages V_1 and V_2 with the conditions

$$\begin{aligned} &V_+ = 5\,\text{V}, \\ &R_1 = 1\,\text{k}\Omega, \qquad R_2 = 4\,\text{k}\Omega, \qquad R_3 = 3\,\text{k}\Omega, \qquad R_4 = 2\,\text{k}\Omega, \\ &I_0 = 3\,\text{nA}, \qquad V_T = 0.05\,\text{V}. \end{aligned}$$

You can use either the relaxation method or Newton's method to solve the equations. If you use Newton's method you can solve Eq. (6.108) for $\Delta\mathbf{x}$ using the function `solve()` from `numpy.linalg` if you want to, but in this case the matrix is only a 2×2 matrix, so it's easy to calculate the inverse directly too.

c) The electronic engineer's rule of thumb for diodes is that the voltage across a (forward biased) diode is always about 0.6 volts. Confirm that your results agree with this rule.

Exercise 6.19: The temperature of a light bulb

An incandescent light bulb is a simple device—it contains a filament, usually made of tungsten, heated by the flow of electricity until it becomes hot enough to radiate thermally. Essentially all of the power consumed by such a bulb is radiated as electromagnetic energy, but some of the radiation is not in the visible wavelengths, which means it is useless for lighting purposes.

Let us define the efficiency of a light bulb to be the fraction of the radiated energy that falls in the visible band. It's a good approximation to assume that the radiation from a filament at temperature T obeys the Planck radiation law, meaning that the power radiated per unit wavelength λ obeys

$$I(\lambda) = 2\pi A h c^2 \frac{\lambda^{-5}}{e^{hc/\lambda k_B T} - 1},$$

where A is the surface area of the filament, h is Planck's constant, c is the speed of light, and k_B is Boltzmann's constant. The visible wavelengths run from $\lambda_1 = 390\,\text{nm}$ to $\lambda_2 = 750\,\text{nm}$, so the total energy radiated in the visible window is $\int_{\lambda_1}^{\lambda_2} I(\lambda)\mathrm{d}\lambda$ and the total energy at all wavelengths is $\int_0^\infty I(\lambda)\mathrm{d}\lambda$. Dividing one expression by the other and substituting for $I(\lambda)$ from above, we get an expression for the efficiency η of the light bulb thus:

$$\eta = \frac{\int_{\lambda_1}^{\lambda_2} \lambda^{-5}/(e^{hc/\lambda k_B T} - 1)\,\mathrm{d}\lambda}{\int_0^\infty \lambda^{-5}/(e^{hc/\lambda k_B T} - 1)\,\mathrm{d}\lambda},$$

where the leading constants and the area A have canceled out. Making the substitution $x = hc/\lambda k_B T$, this can also be written as

$$\eta = \frac{\int_{hc/\lambda_1 k_B T}^{hc/\lambda_2 k_B T} x^3/(e^x - 1)\,\mathrm{d}x}{\int_0^\infty x^3/(e^x - 1)\,\mathrm{d}x} = \frac{15}{\pi^4} \int_{hc/\lambda_2 k_B T}^{hc/\lambda_1 k_B T} \frac{x^3}{e^x - 1}\,\mathrm{d}x,$$

where we have made use of the known exact value of the integral in the denominator.

a) Write a Python function that takes a temperature T as its argument and calculates the value of η for that temperature from the formula above. The integral in the formula cannot be done analytically, but you can do it numerically using any method of your choice. (For instance, Gaussian quadrature with 100 sample points works fine.) Use your function to make a graph of η as a function of temperature between 300 K and 10 000 K. You should see that there is an intermediate temperature where the efficiency is a maximum.

b) Calculate the temperature of maximum efficiency of the light bulb to within 1 K using golden ratio search. (Hint: An accuracy of 1 K is the equivalent of a few parts in ten thousand in this case. To get this kind of accuracy in your calculation you'll need to use values for the fundamental constants that are suitably accurate, i.e., you will need values accurate to several significant figures.)

c) Is it practical to run a tungsten-filament light bulb at the temperature you found? If not, why not?

Exercise 6.20: Roots of a polynomial via the companion matrix:

When finding the roots of a function, in the specific case where that function is a polynomial $f(x) = a_0 + a_1 x + \ldots + a_N x^N$, we can take a completely different approach and make use of the *companion*

matrix, which is the $N \times N$ matrix

$$\mathbf{A} = \frac{1}{a_N}\begin{pmatrix} 0 & a_N & 0 & \cdots & 0 \\ 0 & 0 & a_N & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_N \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{N-1} \end{pmatrix}.$$

If λ_i is the ith root of the polynomial, then the vector $\mathbf{v}_i = \left(1, \lambda_i, \lambda_i^2, \ldots, \lambda_i^{N-1}\right)$ is an eigenvector of the companion matrix:

$$\begin{aligned} \frac{1}{a_N}\begin{pmatrix} 0 & a_N & 0 & \cdots & 0 \\ 0 & 0 & a_N & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_N \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{N-1} \end{pmatrix}\begin{pmatrix} 1 \\ \lambda_i \\ \vdots \\ \lambda_i^{N-2} \\ \lambda_i^{N-1} \end{pmatrix} &= \frac{1}{a_N}\begin{pmatrix} a_N\lambda_i \\ a_N\lambda_i^2 \\ \vdots \\ a_N\lambda_i^{N-1} \\ -\left(a_0 + a_1\lambda_i + \ldots + a_{N-1}\lambda_i^{N-1}\right) \end{pmatrix} \\ &= \begin{pmatrix} \lambda_i \\ \lambda_i^2 \\ \vdots \\ \lambda_i^{N-1} \\ \lambda_i^N \end{pmatrix} = \lambda_i\begin{pmatrix} 1 \\ \lambda_i \\ \vdots \\ \lambda^{N-2} \\ \lambda_i^{N-1} \end{pmatrix}, \end{aligned}$$

where in the second line we have made use of the fact that λ_i is a root, so that

$$a_0 + a_1\lambda_i + \ldots + a_{N-1}\lambda_i^{N-1} = -a_N\lambda_i^N.$$

Thus $\mathbf{v}_i$ is not merely an eigenvector, but its eigenvalue is also equal to the root λ_i. This is true for any root and, since there are generically N roots of the polynomial and N eigenvalues of an $N \times N$ matrix, all the eigenvalues must be roots. Thus, if we can calculate the eigenvalues of the companion matrix, we have the roots of the polynomial.

This is not the fastest way to calculate polynomial roots. Newton's method and binary search are both faster. But if we already have a way to calculate matrix eigenvalues then the companion matrix method is simple to implement, and it is relatively bulletproof: it cannot fail to converge like Newton's method and it does not require us to bracket the roots like binary search. Moreover, it works with complex roots as well as real ones—the companion matrix is non-symmetric and hence can have complex eigenvalues.

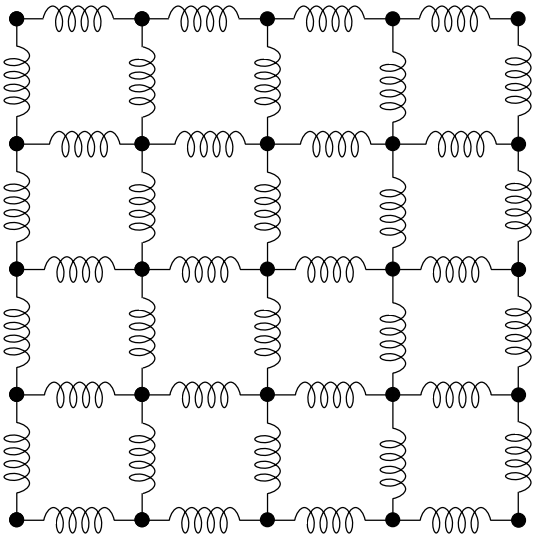
Write a Python program that uses the companion matrix method to calculate the roots of the polynomial from Exercise 6.16:

$$P(x) = 924x^6 - 2772x^5 + 3150x^4 - 1680x^3 + 420x^2 - 42x + 1.$$

If you previously did Exercise 6.16, check that your answers agree. Hint: The function `eigvals` from `numpy.linalg` calculates the eigenvalues of a non-symmetric matrix.

Exercise 6.21: Vibration modes of a 2D lattice:

In this problem we look at a two-dimension version of the vibrating masses and springs from Example 6.2 on page 233. Consider an $n \times n$ square lattice with masses m at each lattice point and identical springs with spring constant k on each bond:



If we denote the integer coordinates of a lattice site by $i, j = 0 \ldots n-1$ and the vibration amplitude of the corresponding mass by x_{ij}, then the equivalent of Eq. (6.56b) on page 234 for a site in the interior of the lattice is

$$-m\omega^2 x_{ij} = k(x_{i+1,j} - x_{ij}) + k(x_{i-1,j} - x_{ij}) + k(x_{i,j+1} - x_{ij}) + k(x_{i,j-1} - x_{ij}),$$

which we can rewrite more simply as

$$x_{i+1,j} + x_{i-1,j} + x_{i,j+1} + x_{i,j-1} - 4x_{ij} = -\frac{m\omega^2}{k} x_{ij}.$$

For sites on the edges and corners of the lattice there are similar equations, but with fewer terms:

$$x_{i+1,0} + x_{i-1,0} + x_{i,1} - 3x_{i,0} = -\frac{m\omega^2}{k} x_{i,0} \qquad \text{for an edge site,}$$

$$x_{1,0} + x_{0,1} - 2x_{0,0} = -\frac{m\omega^2}{k} x_{0,0} \qquad \text{for a corner site.}$$

We can combine all of these equations into a single matrix equation by defining a site index $u = i + jn$, so that

$$x_{u+1} + x_{u-1} + x_{u+n} + x_{u-n} - 4x_u = -\frac{m\omega^2}{k} x_u$$

in the interior of the lattice, and similarly for the equations at the edges and corners. Then all of the equations together can be written in matrix form as an eigenvalue equation $\mathbf{Lx} = -\lambda\mathbf{x}$, where $\lambda = m\omega^2/k$ and $\mathbf{L}$ is the *graph Laplacian matrix*, which is symmetric with elements

$$L_{uv} = \begin{cases} 1 & \text{if sites } u \text{ and } v \text{ are neighbors,} \\ -z_u & \text{if } u = v, \\ 0 & \text{otherwise,} \end{cases}$$

where z_u is the coordination number of site u, i.e., the number of neighbors it has, which will be 2, 3, or 4 depending on the location.

a) Write a program to construct the 16×16 Laplacian matrix $\mathbf{L}$ for a 4×4 lattice, then compute its eigenvalues using the function `eigvalsh` from `numpy.linalg`. The values $\lambda = m\omega^2/k$ tell us the allowed frequencies of vibration of the lattice—the so-called vibration modes. In units where $m = k = 1$, have your program print out the frequencies.

b) What is the lowest possible frequency? Physically, what does this vibration mode represent?

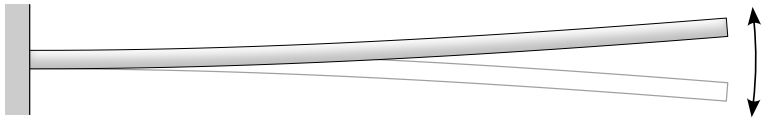
c) Modify your program to also calculate the eigenvectors and then make a density plot of the first excited state of the lattice, i.e., the eigenvector with the second-smallest eigenvalue. To do this, you will have to map the indices $u = i + jn$ back to lattice coordinates i, j and create a 4×4 array with the elements of the eigenvalues arranged appropriately, then make a density plot of that array using `imshow`.

d) Modify your program to make an equivalent density plot for the first excited state of a 50×50 lattice.

e) If you want more of a challenge, make a visualization that separately shows the first 25 excited states of the 50×50 system with 25 separate density plots arranged in a 5×5 grid. You can do this in `pyplot` using the `subplot` function, which allows you to create many separate subplots in the same larger graphic. You may also find useful the functions `axis("off")`, which turns off the labels on the axes of a graph, and `tight_layout`, which gets rid of the unused white space around the edges of the graphics window.

We will encounter the graph Laplacian matrix again in Chapter 9 when we study the solution of partial differential equations like Laplace's equation and the Schrödinger equation.

Exercise 6.22: Vibration of a rigid bar: A rigid bar or rod, clamped and immobile at one end and free at the other, will vibrate when plucked, like this:



Such bars are used as the basis for a variety of musical instruments, including the east African mbira (or kalimba), the Wurlitzer electric piano, the mouth harp, and the lowly tuning fork. The equation of motion for the transverse displacement $\psi(x,t)$ of such a bar at time t and position x along the bar is a variant of the wave equation:

$$\frac{\partial^4\psi}{\partial x^4} + \left(\frac{12\rho}{Eh^2}\right)\frac{\partial^2\psi}{\partial t^2} = 0,$$

where h is the thickness of the bar in the direction of vibration and ρ and E are its density and Young's modulus respectively. (The width of the bar does not enter the equation. A wider bar has greater rigidity, but it also has greater mass, and the two effects cancel exactly.)

For a vibration mode with angular frequency ω, a solution to the above equation has the form

$$\psi(x,t) = \Psi(x)\,\mathrm{e}^{\mathrm{i}\omega t}$$

for some function $\Psi(x)$, and substituting into the equation of motion and canceling a factor of $\mathrm{e}^{\mathrm{i}\omega t}$ gives

$$\frac{\mathrm{d}^4\Psi}{\mathrm{d}x^4} = \frac{12\rho\omega^2}{Eh^2}\,\Psi(x).$$

The form $\Psi(x) = \mathrm{e}^{kx}$ solves this equation with $k^4 = 12\omega^2\rho/Eh^2$ or $k^2 = \pm(2\omega/h)\sqrt{3\rho/E}$. With the positive sign this implies real k and an exponential solution. With the negative sign it implies imaginary k

and an oscillatory solution. The general solution is (the real part of) the linear combination of the four separate possibilities:

$$\Psi(x) = ae^{kx} + be^{-kx} + ce^{ikx} + de^{-ikx},$$

where k is now the positive constant satisfying

$$k^2 = \frac{2\omega}{h}\sqrt{\frac{3\rho}{E}}.$$

Alternatively, since $e^{\pm x} = \cosh x \pm \sinh x$ and $e^{\pm ix} = \cos x \pm i \sin x$, we can write the real solution as

$$\Psi(x) = A\cosh kx + B\sinh kx + C\cos kx + D\sin kx,$$

where A, B, C, and D are constants.

a) The values of A, B, C, and D are set by the boundary conditions at the ends of the bar. There are two boundary conditions at each end. Suppose the bar is of length L and runs from $x = 0$ to $x = L$. One end, say the end at $x = 0$, is clamped and immovable, which means its position is zero at this point, so $\Psi = 0$, and its slope is also zero $d\Psi/dx = 0$. Show that these conditions imply $A + C = 0$ and $B + D = 0$, and hence eliminate the constants C and D from the expression for $\Psi(x)$.

b) The other end of the bar at $x = L$ is free to move and has a different two boundary conditions. The first is that there is no bending at the end of the bar, which means $d^2\Psi/dx^2 = 0$, and the second is that there is no external force upward or downward on the end of the bar, which means $d^3\Psi/dx^3 = 0$. Show that these conditions imply that the two remaining constants A and B satisfy

$$-\frac{B}{A} = \frac{\cosh kL + \cos kL}{\sinh kL + \sin kL} = \frac{\sinh kL - \sin kL}{\cosh kL + \cos kL},$$

and hence, using appropriate trigonometric identities, show that

$$\operatorname{sech} kL + \cos kL = 0.$$

c) Make a plot of the function $f(x) = \operatorname{sech} x + \cos x$ from $x = -10$ to 10.

d) Let x_n be the nth positive root of $f(x)$, with the roots numbered in increasing order. Show that the angular frequency of the nth mode of vibration of the bar is given by

$$\omega_n = \frac{hx_n^2}{2L^2}\sqrt{\frac{E}{3\rho}}.$$

Why is it safe to ignore the negative roots?

e) Write a program to find the values of x_1, x_2, and x_3 to six decimal places using binary search.

f) A bar (or "tine") on a kalimba is 7 cm long and made of steel with thickness 0.5 mm, density 7900 kg/m^3, and Young's modulus 200×10^9 N/m^2. What note does it play when plucked, i.e., what is the frequency of the first mode of vibration? What are the frequencies of the next two modes?

Exercise 6.23: The variational method for quantum mechanics:

The variational method in quantum mechanics is a method for placing an *upper bound* on the ground-state energy of a system. It involves guessing a parametrized form for the ground-state wavefunction and then minimizing the energy of that form with respect to its parameters. Since the true ground state is the lowest-energy state of the system, no other state can be lower. Thus any energy you find for your guessed wavefunction must be greater than or equal to the ground-state energy. In other words, the method allows you to calculate a number E_{var} such that the true ground-state energy is definitely less than or equal to E_{var}. And you do this without ever actually solving the Schrödinger equation.

As an example, consider the quantum simple harmonic oscillator, meaning a single particle of mass m in a one-dimensional quadratic potential well $V(x) \propto x^2$, and let us guess a possible form for the ground-state wavefunction $\psi(x)$. Since the system is left-right symmetric and has its lowest potential energy at $x = 0$, the wavefunction should be symmetric and peaked in the middle. We will guess a Gaussian:

$$\psi(x) = \frac{1}{\sqrt[4]{\pi\sigma^2}}\, e^{-x^2/2\sigma^2},$$

where σ is the width of the Gaussian.

a) Show that the wavefunction is correctly normalized, meaning that $\int_{-\infty}^{\infty} |\psi(x)|^2 dx = 1$, by calculating the integral numerically using Gaussian quadrature with 100 points for the case $\sigma = 10^{-10}$ m. You will have to perform a change of variables to cover the infinite range of the integral. Think carefully about what change of variables you should use. All the weight of the Gaussian wavefunction is in the region within a few σ of the origin, so to get a good value for the integral most of your integration points x need to fall in this range as well. A standard change of variables such as $x = \tan z$ (Eq. (5.106) on page 172) will not work because with 100-point Gaussian quadrature the integration points will all fall too far from the origin. Think about how you can modify a formula like $x = \tan z$ to fix this problem and get a good value for the integral, while still integrating over the entire range from $-\infty$ to $+\infty$.

b) Using your knowledge of quantum mechanics, show that the expected energy of a particle of mass m with this wavefunction in potential $V(x)$ is

$$\langle E \rangle = \frac{1}{\sqrt{\pi\sigma^2}} \int_{-\infty}^{\infty} \left[\frac{\hbar^2}{2m\sigma^2} \left(1 - \frac{x^2}{\sigma^2} \right) + V(x) \right] e^{-x^2/\sigma^2}\, dx.$$

c) Write a Python function that takes the value of σ as its argument and uses Gaussian quadrature on 100 points to calculate the value of $\langle E \rangle$ for an electron in the quadratic potential $V(x) = V_0(x/x_0)^2$ with $x_0 = 10^{-10}$ m and $V_0 = 10$ eV. Use your function to make a plot of $\langle E \rangle$ against σ for values of σ up to 10^{-9} m. You should find that the curve goes down and then back up again, with a minimum in the middle. Estimate the approximate position of the minimum energy by eye from your plot.

d) Combine your function with a golden ratio search to find the value of σ that gives the minimum energy. Calculate the value of σ to an accuracy of at least 10^{-15} m, then use this value to calculate the minimum energy itself, in electron volts.

e) This minimum energy is an upper bound on the true ground-state energy of the particle. It is known that the exact ground-state energy E_0 of the quantum harmonic oscillator, in the notation used here, is

$$E_0 = \frac{\hbar}{x_0}\sqrt{\frac{V_0}{2m}}.$$

How does your result compare?

f) You should find that your program gives exactly the correct ground-state energy in this case (near enough). This is because a Gaussian is actually the correct functional form for the ground-state wavefunction of the quantum harmonic oscillator. Now modify your program to calculate an upper bound on the ground-state energy of the *anharmonic* oscillator having $V(x) = V_0[(x/x_0)^2 + (x/x_0)^4]$, with the same values of V_0 and x_0 as before and using the same Gaussian approximation for the wavefunction.