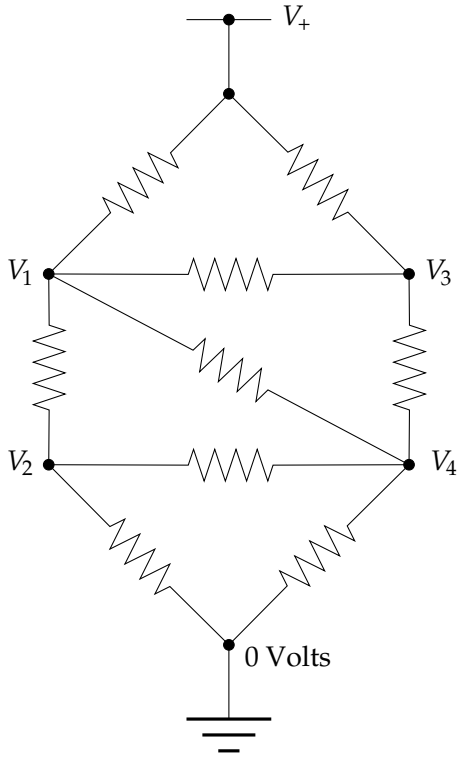
COMPUTATIONAL PHYSICS

EXERCISES FOR CHAPTER 6

Exercise 6.1: A circuit of resistors

Consider the following circuit of resistors:



All the resistors have the same resistance R. The power rail at the top is at voltage $V_+ = 5\,\mathrm{V}$. 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 (or you can modify a program you already have, such as the program `gausselim.py` in Example 6.1).

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) Modify the 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 the method of LU decomposition.

a) Starting, if you wish, with the program for Gaussian elimination in Example 6.1 on page 218, 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 of the calculation 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 the $\mathbf{L}$ and $\mathbf{U}$ you get and verifying that you recover the original matrix once more.

b) Build on your LU decomposition function to create a complete program to solve Eq. (6.2) by performing a double backsubstitution as described in this section. Solve the same equations using the function `solve` from the `numpy` package and verify that you get the same answer either way.

c) If you're 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 difference has no effect on the results, so it's fine to perform all the swaps at once if we wish.)

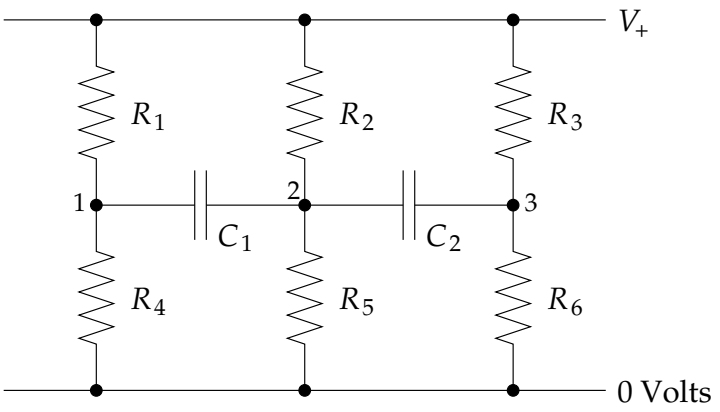
   Modify the function you wrote for part (a) to perform LU decomposition with partial pivoting. The function should return the matrices $\mathbf{L}$ and $\mathbf{U}$ for the LU decomposition of the swapped matrix, plus a list of the swaps made. Then modify the rest of your program

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`.

LU decomposition with partial pivoting is the most widely used method for the solution of simultaneous equations in practice. Precisely this method is used in the function `solve` from the `numpy` package. There's nothing wrong with using the `solve` function—it's well written, fast, and convenient. But it does nothing you haven't already done yourself if you've 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, you should check that you get the same answer both times.

Exercise 6.5: Here's 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

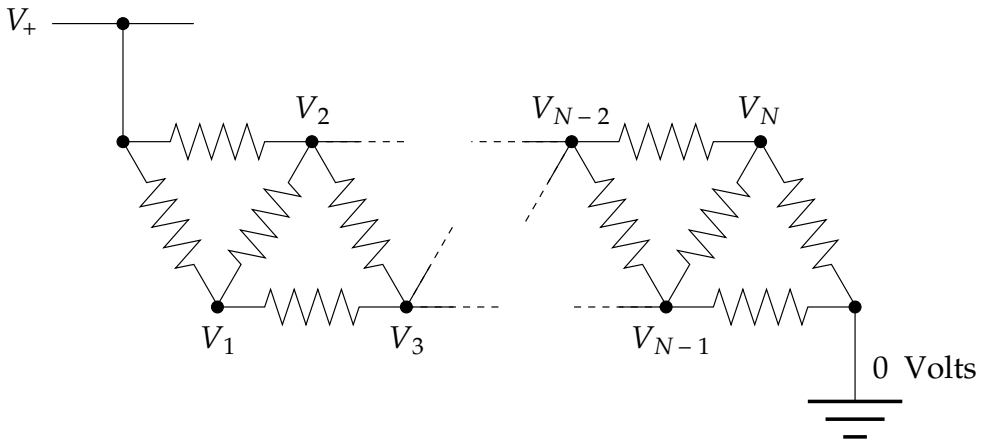
$$\begin{aligned} R_1 &= R_3 = R_5 = 1\,\text{k}\Omega, \\ R_2 &= R_4 = R_6 = 2\,\text{k}\Omega, \\ C_1 &= 1\,\mu\text{F}, \qquad C_2 = 0.5\,\mu\text{F}, \\ x_+ &= 3\,\text{V}, \qquad \omega = 1000\,\text{s}^{-1}. \end{aligned}$$

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 237 or `springsb.py` on page 238, 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. (6.53). 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 spheres. (Spheres of radius about 0.2 or 0.3 seem to work well.)

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 `banded` function from the file `banded.py`, discussed in Appendix E.

Exercise 6.8: The QR algorithm

In this exercise you'll 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 program that finds the QR decomposition of a matrix. Then we'll use that decomposition to find the eigenvalues.

As described above, 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. 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's worth multiplying out the matrices on the right to verify for yourself that you get the correct expressions for the $\mathbf{a}_i$.)

Notice 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

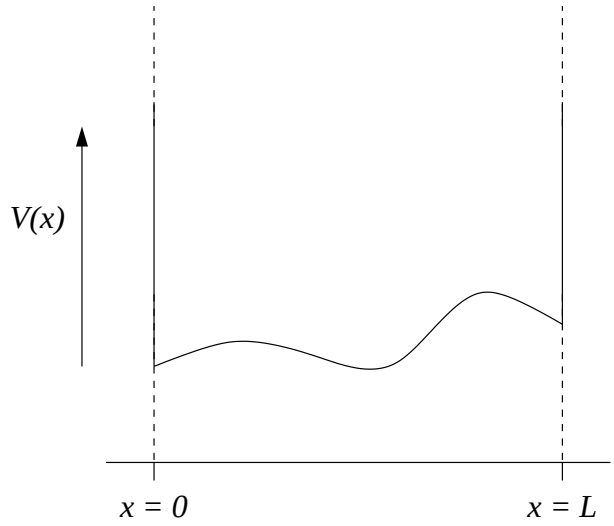
$$\mathbf{A} = \begin{pmatrix} 1 & 4 & 8 & 4 \\ 4 & 2 & 3 & 7 \\ 8 & 3 & 6 & 9 \\ 4 & 7 & 9 & 2 \end{pmatrix}.$$

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 magnitude of every off-diagonal element of the matrix is smaller than 10^{-6}. Test your program on the example matrix above. You should find that the eigenvalues are 1, 21, -3, and -8.

Exercise 6.9: 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 examples you've probably seen before. Suppose instead 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 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, let's 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 m, n positive integers

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

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

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

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'll probably find it useful to know that

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

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 width 5 Å, and $a = 10\,\mathrm{eV}$. (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) Now 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 $\int_0^L |\psi(x)|^2\,\mathrm{d}x = 1$. Is this true of your wavefunction?

Exercise 6.10: 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 6.11: Overrelaxation

If you did not already do Exercise 6.10, 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 nonlinear 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 iteration of the modified equation

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

(keeping the definition of Δx the same). If the parameter 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 the value of x would have been changed and changes by a little more. Using $\Delta x = f(x) - x$, we can also write x' as

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

which is the form in which it is usually written.

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 typically also give good results. 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. (6.81) for the overrelaxation method and hence show that the error on x', the equivalent of Eq. (6.83), is given by

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

b) Consider again the equation $x = 1 - e^{-cx}$ that we solved in Exercise 6.10. 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 $x = 1 - e^{-cx}$ 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 about 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 6.12: The biochemical process of *glycolysis*, the breakdown of glucose in the body to release energy, can be modeled by the equations

$$\frac{dx}{dt} = -x + ay + x^2y, \qquad \frac{dy}{dt} = b - ay - x^2y.$$

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

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

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 6.13: 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*:

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

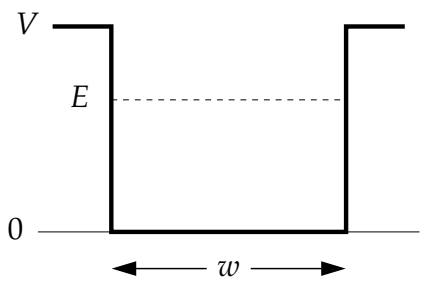
where the so-called *Wien displacement constant* is $b = hc/k_B x$, and x is the solution to the nonlinear equation

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

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 $\lambda = 502\,\text{nm}$. From the equations above and your value of the displacement constant, estimate the surface temperature of the Sun.

Exercise 6.14: 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 6.15: 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.

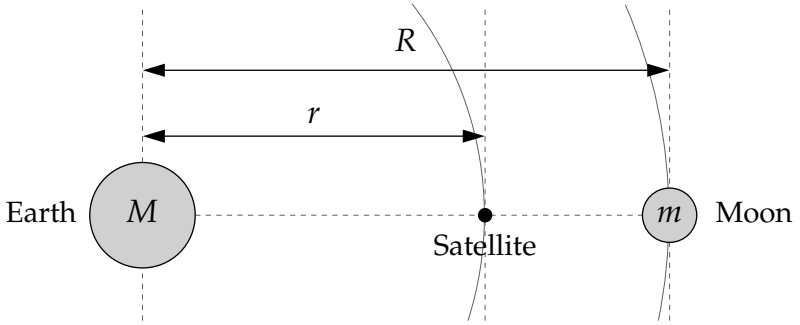
a) Make a plot of $P(x)$ from $x = 0$ to $x = 1$ and by inspecting it find rough values for the six roots of the polynomial—the points at which the function is zero.

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.

Exercise 6.16: 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. Here's the setup:



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 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 fifth-order polynomial equation in r (also called a quintic equation). Such equations cannot be solved exactly in closed form, but it's 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:

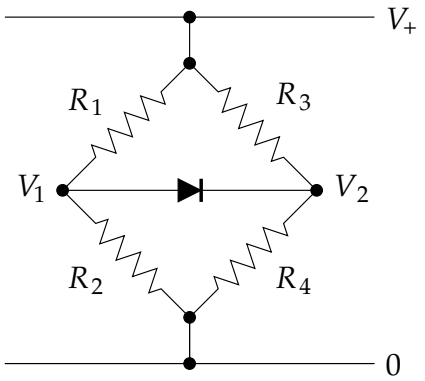
$$\begin{aligned} G &= 6.674 \times 10^{-11}\,\mathrm{m^3 kg^{-1} s^{-2}}, \\ M &= 5.974 \times 10^{24}\,\mathrm{kg}, \\ m &= 7.348 \times 10^{22}\,\mathrm{kg}, \\ R &= 3.844 \times 10^{8}\,\mathrm{m}, \\ \omega &= 2.662 \times 10^{-6}\,\mathrm{s^{-1}}. \end{aligned}$$

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

Exercise 6.17: 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

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

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 6.18: 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 $\int_{\lambda_1}^{\lambda_2} I(\lambda)\,d\lambda$ and the total energy at all wavelengths is $\int_0^\infty I(\lambda)\,d\lambda$. 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:

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

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

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

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?