Ordinary differential equations
Partial differential equations

Homework 5: due on Monday November 28th at 12:30 pm

Problem 1: The Lotka–Volterra equations

The Lotka–Volterra equations are a mathematical model of predator–prey interactions between biological species. Let two variables x and y be proportional to the size of the populations of two species, traditionally called "rabbits" (the prey) and "foxes" (the predators). You could think of x and y as being the population in thousands, say, so that $x = 2$ means there are 2000 rabbits. Strictly the only allowed values of x and y would then be multiples of 0.001, since you can only have whole numbers of rabbits or foxes. But 0.001 is a pretty close spacing of values, so it's a decent approximation to treat x and y as continuous real numbers so long as neither gets very close to zero.

In the Lotka–Volterra model the rabbits reproduce at a rate proportional to their population, but are eaten by the foxes at a rate proportional to both their own population and the population of foxes:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy,$$

where α and β are constants. At the same time the foxes reproduce at a rate proportional the rate at which they eat rabbits—because they need food to grow and reproduce—but also die of old age at a rate proportional to their own population:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \gamma xy - \delta y,$$

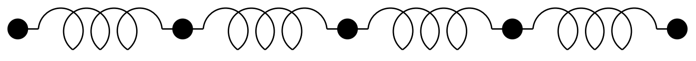
where γ and δ are also constants.

a) Write a program to solve these equations using the fourth-order Runge–Kutta method for the case $\alpha = 1$, $\beta = \gamma = 0.5$, and $\delta = 2$, starting from the initial condition $x = y = 2$. Have the program make a graph showing both x and y as a function of time on the same axes from $t = 0$ to $t = 30$. (Hint: Notice that the differential equations in this case do not depend explicitly on time t—in vector notation, the right-hand side of each equation is a function $f(\mathbf{r})$ with no t dependence. You may nonetheless find it convenient to define a Python function `f(r,t)` including the time variable, so that your program takes the same form as other programs given in Newman's Chapter 8. You don't have to do it that way, but it can avoid some confusion. Several of the following exercises have a similar lack of explicit.)

b) Describe in words what is going on in the system, in terms of rabbits and foxes.

Problem 2: Vibration in a one-dimensional system

Let's study the motion of a system of N identical masses (in zero gravity) joined by identical linear springs like this:



As shown in Newman's Example 6.2, the horizontal displacements ξ_i of masses $i = 1 \ldots N$ satisfy equations of motion

$$\begin{aligned} m\frac{\mathrm{d}^2\xi_1}{\mathrm{d}t^2} &= k(\xi_2 - \xi_1) + F_1, \\ m\frac{\mathrm{d}^2\xi_i}{\mathrm{d}t^2} &= k(\xi_{i+1} - \xi_i) + k(\xi_{i-1} - \xi_i) + F_i\,, \\ m\frac{\mathrm{d}^2\xi_N}{\mathrm{d}t^2} &= k(\xi_{N-1} - \xi_N) + F_N. \end{aligned}$$

where m is the mass, k is the spring constant, and F_i is the external force on mass i. Example 6.2 shows how these equations could be solved by guessing a form for the solution and using a matrix method. Here we'll solve them more directly.

Write a program to solve for the motion of the masses using the fourth-order Runge–Kutta method, where $m = 1$ and $k = 6$, and the driving forces are all zero except for $F_1 = \cos\omega t$ with $\omega = 2$. Plot your solutions for the displacements ξ_i of all the masses as a function of time from $t = 0$ to $t = 20$ on the same plot. Write your program to work with general N, but test it out for small values—$N = 5$ is a reasonable choice.

You will need first of all to convert the N second-order equations of motion into $2N$ first-order equations. Then combine all of the dependent variables in those equations into a single large vector **r** to which you can apply the Runge–Kutta method in the standard fashion.

Problem 3: Oscillating chemical reactions

The *Belousov–Zhabotinsky reaction* is a chemical oscillator, a cocktail of chemicals which, when heated, undergoes a series of reactions that cause the chemical concentrations in the mixture to oscillate between two extremes. You can add an indicator dye to the reaction which changes color depending on the concentrations and watch the mixture switch back and forth between two different colors for as long as you go on heating the mixture.

Physicist Ilya Prigogine formulated a mathematical model of this type of chemical oscillator, which he called the "Brusselator" after his home town of Brussels. The equations for the Brusselator are

$$\frac{\mathrm{d}x}{\mathrm{d}t} = 1 - (b+1)x + ax^2y, \qquad \frac{\mathrm{d}y}{\mathrm{d}t} = bx - ax^2y.$$

Here x and y represent concentrations of chemicals and a and b are positive constants.

Write a program to solve these equations for the case $a = 1$, $b = 3$ with initial conditions $x = y = 0$, to an accuracy of at least $\delta = 10^{-10}$ per unit time in both x and y, **using the adaptive Bulirsch–Stoer method** described in Newman's Section 8.5.6. Calculate a solution from $t = 0$ to $t = 20$, initially using a single time interval of size $H = 20$. Allow a maximum of $n = 8$ modified midpoint steps in an interval before you divide in half and try again.

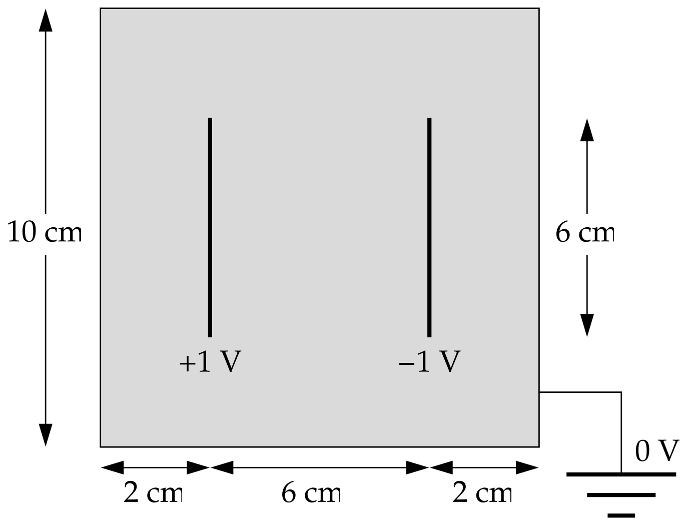
Make a plot of your solutions for x and y as a function of time, both on the same graph, and have your program add dots to the curves to show where the boundaries of the time intervals lie. You should find that the points are significantly closer together in parts of the solution where the variables are changing rapidly.

Hint: The simplest way to do this calculation is to make use of recursion, the ability of a Python function to call itself. Write a user-defined function called, say, `step(r,t,H)` that takes as arguments the vector $\mathbf{r} = (x, y)$ at a starting time t and an interval length H, and returns the new value of $\mathbf{r}$ at time $t + H$. This function should perform the modified midpoint/Richardson extrapolation calculation described in Newman's Section 8.5.5 until either the calculation converges to the required accuracy or you reach the maximum number $n = 8$ of modified midpoint steps. If it fails to converge in eight steps, have your function call itself, twice, to calculate separately the solution for the first then the second half of the interval from t to $t + H$, something like this:

```
r1 = step(r,t,H/2)
r2 = step(r1,t+H/2,H/2)
```

(Then *these* functions can call themselves, and so forth, subdividing the interval as many times as necessary to reach the required accuracy.)

Problem 4: Consider the following 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 solve Laplace's equation and 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*, so they are part of the boundary condition in this case: the capacitor plates behave the same way as the walls of the box, with potentials that are fixed at a certain value and cannot change.

Problem 5: 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, $A = 10°\mathrm{C}$ and $B = 12°\mathrm{C}$. 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 $D = 0.1\,\mathrm{m}^2\,\mathrm{day}^{-1}$.

Write a program, or modify one of the ones given 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, 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.

Problem 6: 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 the class exercise, a single particle in one dimension in a box of length L with impenetrable walls. The wavefunction in such a box necessarily goes to zero on the walls and hence one possible (unnormalized) solution of the equation is

$$\psi_k(x,t) = \sin\left(\frac{\pi k x}{L}\right) \mathrm{e}^{\mathrm{i}Et/\hbar},$$

where the energy E can be found by substituting into the Schrödinger equation, giving

$$E = \frac{\pi^2\hbar^2k^2}{2ML^2}.$$

As with the vibrating string of Newman's Section 9.3.4, we can write a full solution as a linear combination of such individual solutions, which on the grid points $x_n = nL/N$ takes the value

$$\psi(x_n,t) = \frac{1}{N}\sum_{k=1}^{N-1} b_k \sin\left(\frac{\pi k n}{N}\right) \exp\left(\mathrm{i}\frac{\pi^2\hbar k^2}{2ML^2}t\right),$$

where the b_k are some set of (possibly complex) coefficients that specify the exact shape of the wavefunction and the leading factor of $1/N$ is optional but convenient.

Since the Schrödinger equation (unlike the wave equation) is first order in time, we need only a single initial condition on the value of $\psi(x,t)$ to specify the coefficients b_k, although, since the coefficients are in general complex, we will need to calculate both real and imaginary parts of each coefficient.

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

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

where

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

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

a) Write a program to calculate the values of the coefficients b_k, which for convenience can be broken down into their real and imaginary parts as $b_k = \alpha_k + i\eta_k$. Divide the box into $N = 1000$ slices and create two arrays containing the real and imaginary parts of $\psi(x_n, 0)$ at each grid point. Perform discrete sine transforms on each array separately and hence calculate the values of the α_k and η_k for all $k = 1 \ldots N-1$.

   To perform the discrete sine transforms, you can use the fast transform function `dst` from the package `dcst`, which you can find in the on-line resources in the file named `dcst.py`. The function takes an array of N real numbers and returns the discrete sine transform as another array of N numbers.

   (Note that the first element of the input array should in principle always be zero for a sine transform, but if it is not the `dst` function will simply pretend that it is. Similarly the first element of the returned array is always zero, since the $k = 0$ coefficient of a sine transform is always zero. So in effect, the sine transform really only takes $N-1$ real numbers and transforms them into another $N-1$ real numbers. In some implementations of the discrete sine transform, therefore, though not the one in the package `dsct` used here, the first element of each array is simply omitted, since it's always zero anyway, and the arrays are only $N-1$ elements long.)

b) Putting $b_k = \alpha_k + i\eta_k$ in the solution above and taking the real part we get

$$\operatorname{Re}\psi(x_n,t) = \frac{1}{N}\sum_{k=1}^{N-1}\left[\alpha_k \cos\left(\frac{\pi^2\hbar k^2}{2ML^2}t\right) - \eta_k \sin\left(\frac{\pi^2\hbar k^2}{2ML^2}t\right)\right]\sin\left(\frac{\pi k n}{N}\right)$$

   for the real part of the wavefunction. This is an inverse sine transform with coefficients equal to the quantities in the square brackets. Extend your program to calculate the real part of the wavefunction $\psi(x,t)$ at an arbitrary time t using this formula and the inverse discrete sine transform function `idst`, also from the package `dcst`. Test your program by making a graph of the real part of the wavefunction at time $t = 10^{-16}$ s.

c) Extend your program further to study the evolution of the wavefunction over time, similar to the class exercise (a suitable time interval for each frame/plot is about 10^{-18} s). Try to explain in physics terms what is going on in the system.