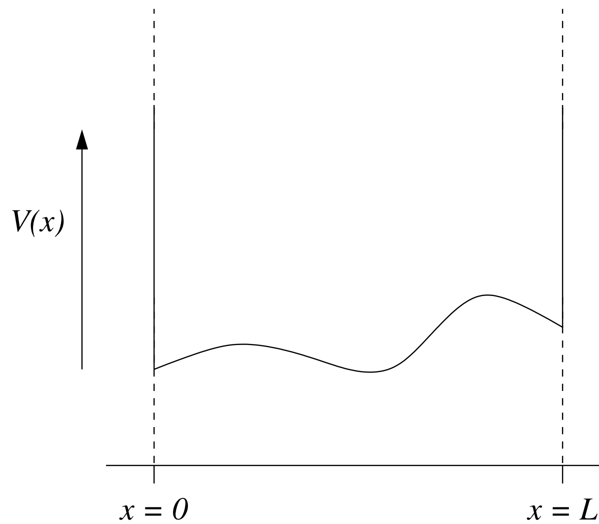
PHYS 7411 Spring 2015
Computational Physics
Homework 4

Due by 3:00pm in Nicholson 447 on 30 March 2015

(Any late assignments will be penalized in the amount of 25% per day late. Any copying of computer programs will be penalized with no credit.)

Exercise 1: 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

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

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'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 2: 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 3: Consider the following semidefinite program:

$$\min x$$

subject to

$$\begin{bmatrix} x & 1 \\ 1 & y \end{bmatrix} \geq 0.$$

a) Draw the feasible set. Is it convex?

b) Write the dual SDP.

c) Is the primal strictly feasible? Is the dual strictly feasible?

d) What can you say about strong duality?

Exercise 4: Fast Fourier transform

Write your own program to compute the fast Fourier transform for the case where N is a power of two, based on the formulas given in Section 7.4.1. As a test of your program, use it to calculate the Fourier transform of the data in the file `pitch.txt`, which can be found in the on-line resources. A plot of the data is shown in Fig. 7.3. You should be able to duplicate the results for the Fourier transform shown in Fig. 7.4.

This exercise is quite tricky. You have to calculate the coefficients $E_k^{(m,j)}$ from Eq. (7.43) for all levels m, which means that first you will have to plan how the coefficients will be stored. Since, as we have seen, there are exactly N of them at every level, one way to do it would be to create a two-dimensional complex array of size $N \times (1 + \log_2 N)$, so that it has N complex numbers for each level from

zero to $\log_2 N$. Then within level m you have 2^m individual transforms denoted by $j = 0 \ldots 2^m - 1$, each with $N/2^m$ coefficients indexed by k. A simple way to arrange the coefficients would be to put all the $k = 0$ coefficients in a block one after another, then all the $k = 1$ coefficients, and so forth. Then $E_k^{(m,j)}$ would be stored in the $j + 2^m k$ element of the array.

This method has the advantage of being quite simple to program, but the disadvantage of using up a lot of memory space. The array contains $N \log_2 N$ complex numbers, and a complex number typically takes sixteen bytes of memory to store. So if you had to do a large Fourier transform of, say, $N = 10^8$ numbers, it would take $16N \log_2 N \simeq 42$ gigabytes of memory, which is much more than most computers have.

An alternative approach is to notice that we do not really need to store all of the coefficients. At any one point in the calculation we only need the coefficients at the current level and the previous level (from which the current level is calculated). If one is clever one can write a program that uses only two arrays, one for the current level and one for the previous level, each consisting of N complex numbers. Then our transform of 10^8 numbers would require less than four gigabytes, which is fine on most computers.

(There is a third way of storing the coefficients that is even more efficient. If you store the coefficients in the correct order, then you can arrange things so that every time you compute a coefficient for the next level, it gets stored in the same place as the old coefficient from the previous level from which it was calculated, and which you no longer need. With this way of doing things you only need one array of N complex numbers—we say the transform is done "in place." Unfortunately, this in-place Fourier transform is much harder to work out and harder to program. If you are feeling particularly ambitious you might want to give it a try, but it's not for the faint-hearted.)

Exercise 5: Image deconvolution

You've probably seen it on TV, in one of those crime drama shows. They have a blurry photo of a crime scene and they click a few buttons on the computer and magically the photo becomes sharp and clear, so you can make out someone's face, or some lettering on a sign. Surely (like almost everything else on such TV shows) this is just science fiction? Actually, no. It's not. It's real and in this exercise you'll write a program that does it.

When a photo is blurred each point on the photo gets smeared out according to some "smearing distribution," which is technically called a *point spread function*. We can represent this smearing mathematically as follows. For simplicity let's assume we're working with a black and white photograph, so that the picture can be represented by a single function $a(x, y)$ which tells you the brightness at each point (x, y). And let us denote the point spread function by $f(x, y)$. This means that a single bright dot at the origin ends up appearing as $f(x, y)$ instead. If $f(x, y)$ is a broad function then the picture is badly blurred. If it is a narrow peak then the picture is relatively sharp.

In general the brightness $b(x, y)$ of the blurred photo at point (x, y) is given by

$$b(x, y) = \int_0^K \int_0^L a(x', y') f(x - x', y - y') \, \mathrm{d}x' \, \mathrm{d}y',$$

where $K \times L$ is the dimension of the picture. This equation is called the *convolution* of the picture with the point spread function.

Working with two-dimensional functions can get complicated, so to get the idea of how the math works, let's switch temporarily to a one-dimensional equivalent of our problem. Once we work out the details in 1D we'll return to the 2D version. The one-dimensional version of the convolution above would be

$$b(x) = \int_0^L a(x') f(x - x') \, \mathrm{d}x'.$$

The function $b(x)$ can be represented by a Fourier series as in Eq. (7.5):

$$b(x) = \sum_{k=-\infty}^{\infty} \tilde{b}_k \exp\biggl(\mathrm{i}\frac{2\pi kx}{L}\biggr),$$

where

$$\tilde{b}_k = \frac{1}{L}\int_0^L b(x) \exp\biggl(-\mathrm{i}\frac{2\pi kx}{L}\biggr)\,\mathrm{d}x$$

are the Fourier coefficients. Substituting for $b(x)$ in this equation gives

$$\begin{aligned}
\tilde{b}_k &= \frac{1}{L}\int_0^L \int_0^L a(x')f(x-x') \exp\biggl(-\mathrm{i}\frac{2\pi kx}{L}\biggr)\,\mathrm{d}x'\,\mathrm{d}x \\
&= \frac{1}{L}\int_0^L \int_0^L a(x')f(x-x') \exp\biggl(-\mathrm{i}\frac{2\pi k(x-x')}{L}\biggr) \exp\biggl(-\mathrm{i}\frac{2\pi kx'}{L}\biggr)\,\mathrm{d}x'\,\mathrm{d}x.
\end{aligned}$$

Now let us change variables to $X = x - x'$, and we get

$$\tilde{b}_k = \frac{1}{L}\int_0^L a(x') \exp\biggl(-\mathrm{i}\frac{2\pi kx'}{L}\biggr) \int_{-x'}^{L-x'} f(X) \exp\biggl(-\mathrm{i}\frac{2\pi kX}{L}\biggr)\,\mathrm{d}X\,\mathrm{d}x'.$$

If we make $f(x)$ a periodic function in the standard fashion by repeating it infinitely many times to the left and right of the interval from 0 to L, then the second integral above can be written as

$$\begin{aligned}
\int_{-x'}^{L-x'} f(X) \exp\biggl(-\mathrm{i}\frac{2\pi kX}{L}\biggr)\,\mathrm{d}X &= \int_{-x'}^{0} f(X) \exp\biggl(-\mathrm{i}\frac{2\pi kX}{L}\biggr)\,\mathrm{d}X \\
&\qquad + \int_0^{L-x'} f(X) \exp\biggl(-\mathrm{i}\frac{2\pi kX}{L}\biggr)\,\mathrm{d}X \\
&= \exp\biggl(\mathrm{i}\frac{2\pi kL}{L}\biggr) \int_{L-x'}^{L} f(X) \exp\biggl(-\mathrm{i}\frac{2\pi kX}{L}\biggr)\,\mathrm{d}X + \int_0^{L-x'} f(X) \exp\biggl(-\mathrm{i}\frac{2\pi kX}{L}\biggr)\,\mathrm{d}X \\
&= \int_0^L f(X) \exp\biggl(-\mathrm{i}\frac{2\pi kX}{L}\biggr)\,\mathrm{d}X,
\end{aligned}$$

which is simply L times the Fourier transform $\tilde{f}_k$ of $f(x)$. Substituting this result back into our equation for $\tilde{b}_k$ we then get

$$\tilde{b}_k = \int_0^L a(x') \exp\biggl(-\mathrm{i}\frac{2\pi kx'}{L}\biggr) \tilde{f}_k\,\mathrm{d}x' = L\,\tilde{a}_k \tilde{f}_k.$$

In other words, apart from the factor of L, the Fourier transform of the blurred photo is the product of the Fourier transforms of the unblurred photo and the point spread function.

Now it is clear how we deblur our picture. We take the blurred picture and Fourier transform it to get $\tilde{b}_k = L\,\tilde{a}_k \tilde{f}_k$. We also take the point spread function and Fourier transform it to get $\tilde{f}_k$. Then we divide one by the other:

$$\frac{\tilde{b}_k}{L\tilde{f}_k} = \tilde{a}_k$$

which gives us the Fourier transform of the *unblurred* picture. Then, finally, we do an inverse Fourier transform on $\tilde{a}_k$ to get back the unblurred picture. This process of recovering the unblurred picture from the blurred one, of reversing the convolution process, is called *deconvolution*.

Real pictures are two-dimensional, but the mathematics follows through exactly the same. For a picture of dimensions $K \times L$ we find that the two-dimensional Fourier transforms are related by

$$\tilde{b}_{kl} = KL\tilde{a}_{kl}\tilde{f}_{kl}\,,$$

and again we just divide the blurred Fourier transform by the Fourier transform of the point spread function to get the Fourier transform of the unblurred picture.

In the digital realm of computers, pictures are not pure functions $f(x, y)$ but rather grids of samples, and our Fourier transforms are discrete transforms not continuous ones. But the math works out the same again.

The main complication with deblurring in practice is that we don't usually know the point spread function. Typically we have to experiment with different ones until we find something that works. For many cameras it's a reasonable approximation to assume the point spread function is Gaussian:

$$f(x, y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right),$$

where σ is the width of the Gaussian. Even with this assumption, however, we still don't know the value of σ and we may have to experiment to find a value that works well. In the following exercise, for simplicity, we'll assume we know the value of σ.

a) On the web site you will find a file called `blur.txt` that contains a grid of values representing brightness on a black-and-white photo—a badly out-of-focus one that has been deliberately blurred using a Gaussian point spread function of width $\sigma = 25$. Write a program that reads the grid of values into a two-dimensional array of real numbers and then draws the values on the screen of the computer as a density plot. You should see the photo appear. If you get something wrong it might be upside-down. Work with the details of your program until you get it appearing correctly. (Hint: The picture has the sky, which is bright, at the top and the ground, which is dark, at the bottom.)

b) Write another program that creates an array, of the same size as the photo, containing a grid of samples drawn from the Gaussian $f(x, y)$ above with $\sigma = 25$. Make a density plot of these values on the screen too, so that you get a visualization of your point spread function. Remember that the point spread function is periodic (along both axes), which means that the values for negative x and y are repeated at the end of the interval. Since the Gaussian is centered on the origin, this means there should be bright patches in each of the four corners of your picture, something like this:

c) Combine your two programs and add Fourier transforms using the functions `rfft2` and `irfft2` from `numpy.fft`, to make a program that does the following:

   i) Reads in the blurred photo
   ii) Calculates the point spread function
   iii) Fourier transforms both

iv) Divides one by the other
v) Performs an inverse transform to get the unblurred photo
vi) Displays the unblurred photo on the screen

When you are done, you should be able to make out the scene in the photo, although probably it will still not be perfectly sharp.

Hint: One thing you'll need to deal with is what happens when the Fourier transform of the point spread function is zero, or close to zero. In that case if you divide by it you'll get an error (because you can't divide by zero) or just a very large number (because you're dividing by something small). A workable compromise is that if a value in the Fourier transform of the point spread function is smaller than a certain amount ϵ you don't divide by it—just leave that coefficient alone. The value of ϵ is not very critical but a reasonable value seems to be 10^{-3}.

d) Bearing in mind this last point about zeros in the Fourier transform, what is it that limits our ability to deblur a photo? Why can we not perfectly unblur any photo and make it completely sharp?

We have seen this process in action here for a normal snapshot, but it is also used in many physics applications where one takes photos. For instance, it is used in astronomy to enhance photos taken by telescopes. It was famously used with images from the Hubble Space Telescope after it was realized that the telescope's main mirror had a serious manufacturing flaw and was returning blurry photos—scientists managed to partially correct the blurring using Fourier transform techniques.