Physics 465: Computing Project 8 (due T34, 11/14)

November 5, 2012

Particle in a 2-D box (Infinite 2-D Square Well)

In this project we're going to broaden our world to include two dimensions of space. Let's say we have a particle in a 2-D square box of side a and a potential that goes to infinity at the sides/boundaries of the box so that the wavefunction has to go to zero there. What happens if we start the particle out in one corner? How long does it take for the probability distribution to "revive" in that corner? Let's work it out, and then simulate that situation in python.

1) SWE in 2-D using cartesian coordinates

In two dimensions momentum can be described as a 2-D vector:

$$\vec{p} = p_x\hat{\imath} + p_y\hat{\jmath} \tag{1}$$

Generalizing the stationary state wavefunction to two dimensions is also straightforward:

$$\langle\vec{r}|\psi\rangle = \psi(\vec{r}) = \psi(x, y) \tag{2}$$

What happens to the time independent Schrödinger wave equation (TISWE)? It's still the same!

$$\hat{H}\,|\psi_n\rangle = E_n\,|\psi_n\rangle \tag{3}$$

Where $|\psi_n\rangle$ is an eigenstate of $\hat{H}$ with eigenvalue E_n, which in the position representation becomes:

$$\frac{\hat{p}^2}{2m}\psi_n(\vec{r}) + V(\vec{r})\psi_n(\vec{r}) = E_n\psi_n(\vec{r}) \tag{4}$$

How shall we generalize $\hat{p}$? Easy! Again in the position representation: $\hat{p}_x = -i\hbar\partial/\partial x$ and $\hat{p}_y = -i\hbar\partial/\partial y$. Putting these back into Eq. 4

$$-\frac{\hbar^2}{2m}(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})\psi_n(x,y) + V(x,y)\psi_n(x,y) = E_n\psi_n(x,y) \tag{5}$$

Since our "box" potential is zero everywhere except at the boundaries, we can separate $\psi(x,y)$ into parts that only depend on x and y separately: $\psi_x(x)$ and $\psi_y(y)$. Then we get *two* eigenvalue equations for those two parts:

$$-\frac{\hbar^2}{2m}\psi''_{n_x}(x) = E_{n_x}\psi_{n_x}(x) \tag{6}$$

$$-\frac{\hbar^2}{2m}\psi''_{n_y}(y) = E_{n_y}\psi_{n_y}(y) \tag{7}$$

Note that this is just like *two* 1-D infinite square wells! So we already know the solutions:

$$\psi_{n_x}(x) = \sqrt{\frac{2}{a}}\sin(\frac{n_x\pi x}{a}) \tag{8}$$

$$\psi_{n_y}(y) = \sqrt{\frac{2}{a}}\sin(\frac{n_y\pi y}{a}) \tag{9}$$

Note here that there are *two* eigenvalues, one for each dimension. So rather than just $|\psi_n\rangle$, we'll need to consider n_x and n_y to completely specify the overall state $|\psi_{n_x,n_y}\rangle$. The actual overall wavefunction is just the product of the two separate wavefunctions:

$$\psi_{n_xn_y}(x,y) = \frac{2}{a}\sin(\frac{n_x\pi x}{a})\sin(\frac{n_y\pi y}{a}) \tag{10}$$

Substituting these solutions back into Eq. 4 gives:

$$E_{n_x,n_y} = \frac{\hbar^2\pi^2}{2ma^2}(n_x^2 + n_y^2) \tag{11}$$

These are the energies used to predict the time evolution of the wavefunction given the fourier coefficients.

$$\Psi(x,y,t) = \langle xy| e^{-i\frac{\hat{H}}{\hbar}t} \sum_{n_x,n_y} |n_x,n_y\rangle \langle n_x,n_y|\psi(0)\rangle = \sum_{n_x,n_y} c_{n_x,n_y}\psi_{n_x,n_y}(x,y)e^{-i\frac{E_{n_x,n_y}}{\hbar}t} \tag{12}$$

where the fourier coefficients are computing in the "regular" way:

$$c_{n_x,n_y} = \langle \psi_{n_x,n_y} | \psi(0) \rangle \tag{13}$$

$$c_{n_x,n_y} = \int \frac{2}{a} \sin(\frac{n_x \pi x}{a}) \sin(\frac{n_y \pi y}{a}) \psi_0(x,y) \, dx \, dy \tag{14}$$

Representing 2-D wavefunctions in 3-D space

We got away with representing 1-D wavefunctions in 3-D using one dimension for *real* space and using the other two for the real and imaginary parts of the wavefunction. If we move to using up two dimensions on real space, we don't have enough dimensions left over for the real and imaginary parts of $\psi(x,y)$!

So.. how do we do it? I've played around with several representations of the imaginary part, and this is the one I like the best: Rather than arrows, use cylinders to represent the wavefunction in 2-D. For the real part, use the height above the plane. For the imaginary part use the radius of the cylinder for the magnitude, and the color for the sign. Fig. 1 is a screen shot of the look of the $nx = 2$ and $ny = 1$ state alone:

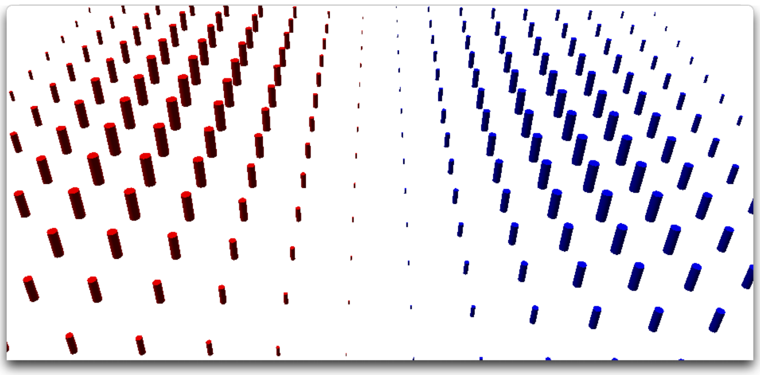

Figure 1: (nx=2, ny=1) state at one point in time

Here is the modified "safcnP8.py" file that handles this case:

```
from visual import color

def SetArrowFromCN( cn, a):
    """
    SetArrowWithCN takes a complex number ‘cn’ and an object ‘a’ .
    This version assumes ‘a’ is a cylinder and sets the height of
    the cylinder based on the real part, and the color/radius based
    on the imaginary part. The radius is never set to less than 5% of the
    magnitude of the complex number.
    """
    a.axis.z = cn.real
    a.radius = max(0.05*abs(cn), abs(cn.imag)/6.0)
    if cn.imag<0:
        a.color=color.blue
    else:
        a.color=color.red
```

2) Initializing a 2-D array to represent wavefunctions

There are a few python tricks we'll need to make this easy. One is an easy way to set up 2-D arrays for computing 2-D wavefunctions. Let's say we want to define a wavefunction over a 2-D area where x goes from 0 to a, and y goes from 0 to a as well (obviously a square area!). We'd like to be able to write something like:

```
psinxny = sin(nx*pi*x/a)*sin(ny*pi*y/a)          # compute the nx,ny energy eigenstate
psinxny = psinxny/sqrt((abs(psinxny)**2).sum())  # normalize it.
```

and have `psinxny` be a 2-D array with the values of the wavefunction. For this to work, `x` needs to be a 2-D array with appropriate x values and `y` needs to be a similar array with the appropriate y values. One easy way to get this is to use the numpy `mgrid` constructor. The `mgrid` constructor takes index-like objects and uses them to create a set of grid arrays that do what we need. Here's the way it works:

```
x, y = mgrid[0:NA:NA*1j, 0:NA:NA*1j]*(a/NA) # trick to make x and y arrays
```

This example makes an `x` grid and a `y` grid, which are really just 2-D arrays, that hold x and y values respectively. The x and y values go from 0 to a and there are `NA` of them. It's sort of like `linspace`, but in two dimensions.

3) Managing eigenstates and related information

Another trick is that of using a dictionary to hold a set of arrays that we can use as eigenfunctions. We could of course use an array of arrays, as we've done before, but I wanted to have a flexible way of specifying exactly which eigenstates I wanted to work with as a pair of lists of nx and ny values. A dictionary is an ideal way to do this. It's basically a mapping type that takes a set of 'keys', and associates them with an object (in this case an array). Our 'keys' will be the tuples (nx, ny) and the objects will be the normalized eigenfunctions that correspond to that key. It's easy to set up like this:

```
# compute the eigenstates and store them in a dictionary 'eigenstates'

NX=[1,2,3,5,6,7]               # which fourier terms in x direction
NY=[1,2,3,5,6,7]               # which fourier terms in y direction

eigenstates = {}               # dictionary for precomputed eigenstates

for nx in NX:
    for ny in NY:
        psinxny = sin(nx*pi*x/a)*sin(ny*pi*y/a)             # the nx,ny eigenstate
        psinxny = psinxny/sqrt((abs(psinxny)**2).sum())     # normalize it.
        eigenstates[(nx, ny)] = psinxny                     # save it
```

This same trick can be used to keep track of fourier coefficients and energies for each term in a very similar way.

4) So... what are we supposed to do?

Your mission is to produce two deliverables:

1) A screenshot of the wavefunction representation evolving in the (2,1) state. It should look something like Fig. 2.

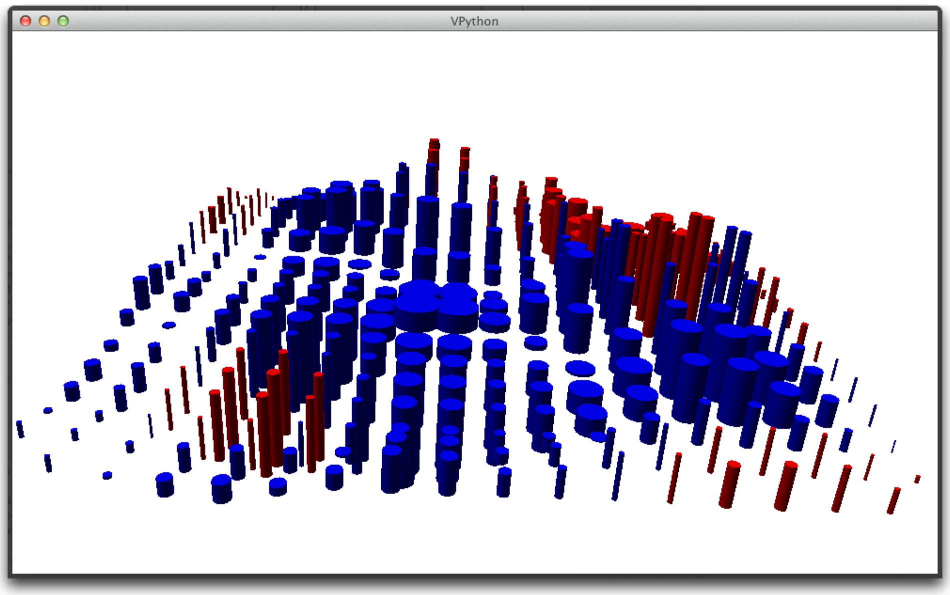

Figure 2: 2-D representation of $\langle \vec{r} | \psi(0) \rangle$ much later than $t = 0$.

2) A graph of the probability of finding the particle in the lower left corner, which should

look something like Fig. 3

$$0 \leq x \leq a/2 \tag{15}$$
$$0 \leq y \leq a/2 \tag{16}$$

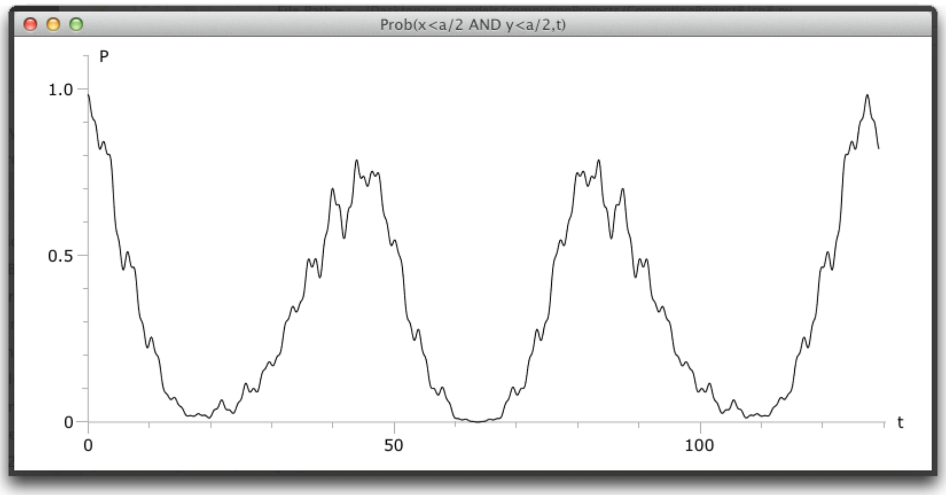

Figure 3: Probability of being in lower left corner vs. time

There is a “Starter Program” on the ‘K’ drive (cp8.py) that you can use to get started.

Questions

Please include your answers to these questions in your report.

1) Using the theory from section 1, predict the time it takes for the system to ‘revive’ to it’s initial state. Show that this prediction corresponds to the apparent revival time on your graph from section 4.

2) Unlike the 1-D infinite square well, the frequencies of the excited states in the 2-D case are not integer multiples of the lowest energy eigenstate. What affect does this fact have on the revival time?