VPython Class 9: Fitting wavefunctions into "boxes" with a computer

1. Introduction

In class last week we talked about using Schrödinger's equation to find special-case quantum wave functions with definite energies. Today we introduce technique using a computer, called numerical integration, to *construct* such special-case wavefunctions point by point.

Recall that Schrödinger's equation is

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + U(x)\psi(x) = E\psi(x). \tag{1}$$

I'm now going to use a standard trick to make things easier for computer work: I'm going to pick a set of dimensionless variables. In the first part of the lab we're going to look at the one-dimensional "particle in a box" problem, or square-well potential, with the well having a length L, but I don't want to worry about whether $L = 1\,\text{nm}$, or $L = 0.3\,\text{nm}$, or something else, so I choose a new variable

$$X \equiv \frac{x}{L}. \tag{2}$$

The left end of the "box" will then be at $X = 0$, and the right end will be at $X = 1$. This choice also affects the expression for the second derivative. Since $x = XL$, we have

$$\frac{d^2\psi}{dx^2} = \frac{1}{L^2}\frac{d^2\psi}{dX^2}. \tag{3}$$

Using this, and some re-arrangement, give Schrödinger's equation as

$$-\frac{d^2\psi(X)}{dX^2} + \frac{2mL^2}{\hbar^2}\,U(X)\psi(X) = \frac{2mL^2}{\hbar^2}\,E\psi(X). \tag{4}$$

or

$$-\frac{d^2\psi(X)}{dX^2} + \frac{8mL^2\pi^2}{h^2}\,U(X)\psi(X) = \frac{8mL^2\pi^2}{h^2}\,E\psi(X). \tag{5}$$

This suggests defining new dimensionless variables for the energies E and U in terms of the energy $h^2/8mL^2$:

$$E' = \frac{E}{\frac{h^2}{8mL^2}} = E\,\frac{8mL^2}{h^2} \qquad \text{and} \qquad U' = \frac{U}{\frac{h^2}{8mL^2}} = U\,\frac{8mL^2}{h^2}. \tag{6}$$

Using these expressions, Schrödinger's equation becomes

$$-\frac{d^2\psi(X)}{dX^2} + \pi^2 U'(X)\,\psi(X) = \pi^2 E'\,\psi(X). \tag{7}$$

NOTE: I could just as easily have incorporated the factor of π^2 within my definition of E' and U', but by not doing so some things come out a little cleaner in this exercise.

This is now in a form that a computer can evaluate, without the messy details of any constants hanging around. Of course at the end, all dimensionless quantities like E' will have to be converted back to energies with units. That's easy though: just multiply E' by $\frac{h^2}{8mL^2}$.

2. Solving Schrödinger's equation numerically

Rearrangement of Eq. (7) highlights the fact that Schrödinger's equation gives us information about the second derivative of the wavefunction:

$$\frac{d^2\psi(x)}{dx^2} = -\pi^2 \left[E - U(x)\right] \psi(x), \tag{8}$$

where I have switched back to conventional symbols x, U and E, but with the understanding that they represent the dimensionless variables X, E', and U'.

For a computer to build a solution to this equation we need a numerical approximation to the second derivative that we can use at a set of discrete points. We wll use an equally spaced set of value of x given by

$$x_i = x_0 + i\Delta, \tag{9}$$

where i is an integer. A set of such x values can be created using the `linspace` function from `numpy`. The corresponding values of the function will be labeled

$$y_i = f(x_i). \tag{10}$$

You are already familiar with a numerical approximation to the first derivative:

$$\left.\frac{dy}{dx}\right|_{x_i} \simeq \frac{y_{i+1} - y_i}{\Delta}. \tag{11}$$

For the second derivative we can use this approximation for the first derivative in the intervals to the right and left of x_i:

$$\begin{aligned}
\left.\frac{d^2y}{dx^2}\right|_{x_i} &\simeq \frac{\left.\frac{dy}{dx}\right|_{\text{right}} - \left.\frac{dy}{dx}\right|_{\text{left}}}{\Delta} \\
&\simeq \frac{\frac{y_{i+1} - y_i}{\Delta} - \frac{y_i - y_{i-1}}{\Delta}}{\Delta} \\
&= \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta^2}
\end{aligned} \tag{12}$$

Solving this for y_{i+1} gives

$$y_{i+1} = 2y_i - y_{i-1} + \Delta^2 \left.\frac{d^2y}{dx^2}\right|_{x_i}. \tag{13}$$

If we know y_0 and y_1, and we know the second derivative from Schrödinger's equation, we can calculate y_2. Then we (i.e., the computer) can repeat the process over and over to calculate y_3, y_4, etc.

2.1 Particle in a box

- Locate a graphing program you have written, and make sure that it still works. In the following I'm assuming that you have have a Python function, and that it has a definition beginning with a statement like

```
def function_name:
    ...
```

- Add a potential energy function at the top of your program. This is ridiculously simple for the first part of this exercise when we're considering a particle in a box:

```
def u(x):
        return 0
```

- Make the left endpoint for your graph be $x = 0$, and your right endpoint be $x = 1$.
- Choose to graph at least 101 points.
- Create the array of x values using

```
x = np.linspace(lep, rep, n)
```

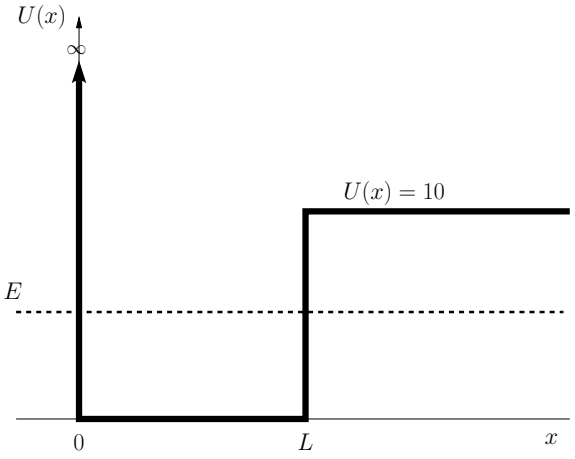
  where I have used the variables `lep`, `rep`, and `n` for my left endpoint, right endpoint, and number of points respectively. (Remember that for `n` points, there are `np - 1` intervals.)

- Calculate the distance between your points and define this to be `dx`.
- Modify your function so that it includes a "guess" for the dimensionless energy, the declaration of an array for your y values (initially all set to zero), an initial value for `y[1]`, and a `for` loop to calculate the rest of the y values:

```
def integrate(x):
    '''Function to perform numerical integration'''
    e = 0.8                    # Value of dimensionless energy
    y = np.zeros(len(x))       # Create array for y values
    y[1] = dx                  # Initialize y[1]
    for i in range(1,np-1):    # Calculate y[2], y[3], y[4], etc.
       YOUR CODE HERE
    return y
```

- At this point you should have the plot of a wavefunction corresponding to your first energy "guess" `e = 0.8`. Is this a valid wavefunction for a particle in a box? If not, what do you have to adjust to make it a valid wavefunction.
- Use your program to find the three lowest energies of the particle in a box. Convert the values you get to quantities with dimensions. Is this what you expect?

2.2 Particle in a semi-infinite square well



- Modify your potential energy function to correspond to that of a semi-infinite square well potential:

```
def u(x):
    if x<1:
        return 0
    else:
def u(x):
    if x<1:
        return 0
    else:
        return 10
```

- Make the right endpoint for your graphs be three, and increase the number of points accordingly.
- Adjust the y limits for your graph to be ± 2.
- Do your previous values of the energy work?
- Find the lowest two allowed energies in this potential.