python_intro_part_03_schroedinger_1D

February 10, 2016

1 Case study: integrating 1D Schroedinger equation over time

1.1 Goal

Invent a potential, create a wave packet, watch it evolve (and tunnel) under the time-dependent Schroedinger equation.

1.2 Physics

Time-dependent Schroedinger equation in 1D

$$i\hbar\frac{\partial\Psi(x,t)}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi(x,t)}{\partial x^2} + V(x)\Psi(x,t) \qquad x,t \in \mathbb{R}$$

Separating real and imaginary part, omitting constants, discretizing in space and time (naively) gives

$$\frac{\Psi_i(x,t+\Delta t) - \Psi_i(x,t)}{\Delta t} = \frac{1}{2}\Psi_r(x,t) * [1,-2,1] - V(x)\Psi_r(x,t) \qquad x \in \mathbb{N}, t \in \mathbb{R}$$
$$\frac{\Psi_r(x,t+\Delta t) - \Psi_r(x,t)}{\Delta t} = -\frac{1}{2}\Psi_i(x,t) * [1,-2,1] + V(x)\Psi_i(x,t) \qquad x \in \mathbb{N}, t \in \mathbb{R}$$

where Ψ_r and Ψ_i are the real and imaginary parts of Ψ, respectively.

This is the simplest possible discretization. This "forward time difference" method is easiest to understand, but behaves poorly for all but the smallest Δt. A better integration scheme is the Leapfrog algorithm (e.g. http://helium.bradley.edu/PICUP/doc/tdse/Numerical_Strategy.pdf).

1.3 Step by step

1. The potential
   - Write a function that returns a vector (a 1D numpy array) holding a parabola. The parabola should be centered in the middle of your vector. The function should take the curvature of the parabola and the length of the vector as arguments. Plot an example result.
   - Write a function that returns a parametric potential landscape. I recommend zero potential in most places, and barriers and valleys with a height or depth of ˜1/10. The sole argument to the function should be the spatial resolution. I have created something like this: Plot an example result.
2. The wave packet
   - Write a function that returns a complex sinusoid, modulated by a Gaussian, that is,

     $$(cos(2\pi kx) + isin(2\pi kx))\exp(-(x-\mu)^2/2/\sigma^2)$$

     The function should take the following arguments: number of discretization points; and location, standard deviation relative to your potential landscape (that is, $\in [0,1]$); and the wave number k. You may ignore normalization constants. While numpy understands complex numbers, I have preferred to explicitly represent the real and imaginary parts. Plot an example result.

3. Integrating the TDSE
    - Really, it's just a loop and two lines of code. Use circular convolution to let your wave packet live on a ring. I have used the following parameters: $\sigma \sim 0.02, k < 30$, spatial resolution $> 400, $ time step $< 1/2$.
    - Store all intermediate results, make suitable plots of the temporal evolution.
    - Marvel at visual quantum mechanics, implemented all by yourself!
    - Repeat with different potential.
    - Now try and break it: see what happens when you make the temporal or spatial discretization too coarse, when you inject too much energy into the system, etc.

1.4 Hints

- do not intersperse constants in your code. Instead, define constants in the header of your script, or pass them as arguments to functions.
- use periodic boundary conditions like so:

```
ndimage.filters.convolve1d(..., ..., mode='wrap')
```

- to create an animation, I used

```
fig = plt.figure()
ax = fig.add_subplot(111)
for t in range(number_of_steps):
print(t)
ax.clear()
ax.set_xlim(0, n)
ax.set_ylim(-1, 1)
ax.plot(pot)

< update wave packet >

ax.plot(wpr, '-b')
ax.plot(wpi, '-r')
ax.figure.canvas.draw()
plt.pause(0.001) # deprecated; but if omitted, plotting stalls on windows machine
```

- for debugging purposes, the variable explorer (top right corner in spyder) is immensely useful.
- Try the debug mode (`Ctrl-F5`). In the ipython console, type `?` for a list of options, or `n` to execute the next line. See how the variable explorer becomes populated as you step through your program, and how the values assigned to variables change. Note you can double-click on arrays in the variable explorer.