2D eigensolver

Daniel Huantes

June 2022

1 What: What are we doing

We are creating a 2D eigensolver in python. Specifically, we are applying a similar method to our 1D eigensolver in an attempt to model a particle confined to some 2D box.

$$\hat{H}\left|\psi_n\right\rangle = E_n \left|\psi_n\right\rangle \tag{1}$$

Again, we observe the schrodinger equation in Hamiltonian form, but this time we expand it slightly to explain where the '2D' comes in

$$\begin{aligned}
\hat{H}\left|\psi_n\right\rangle &= (\hat{V}(x,y) + \hat{T})\left|\psi_n\right\rangle & (2)\\
&= (\hat{V}(x,y) + \frac{\vec{p}^2}{2m})\left|\psi_n\right\rangle & (3)\\
&= (\hat{V}(x,y) + \frac{\mathbf{p}_x^2}{2m} + \frac{\mathbf{p}_y^2}{2m})\left|\psi_n\right\rangle & (4)
\end{aligned}$$

2 Why: Why are we doing it

In theory we could expand to a full 3D, but the ultimate problem we are concerned with is the propagation of a beam through variable refractive index. Treating the changes in refractive index, the Helmholtz equation has the same form as the schrödinger equation, so the 2 spacial dimensions of each 'slice' are the 2 spacial dimensions the 'particle' finds itself in. The beam propagation direction is represented by time.

3 How: What is the design method for this

the state vector $|\psi\rangle$ can be cast to multiple different forms, most commonly that of the probability amplitude function $\psi(x) = \langle x|\psi\rangle$. In the 1D case, we discretize the domain of this function, restricting it to some finite range, and representing it as a 1D numpy array. The utility of using a 1D numpy array is that the Hamiltonian operator can be represented as a 2D matrix, and we can take advantage of existing libraries and methods to calculate it's eigenvectors.

We want to find a representation for a 2D version of our state vector, some code representation that can allow us to keep the hamiltonian in a form we can reasonably calculate the eigenvectors of. My initial idea was to use a 2D grid,

$$\psi(x,y) = \begin{bmatrix} \psi(x_0,y_0) & \psi(x_0,y_0+\Delta y) & \psi(x_0,y_0+2\Delta y) & ... \\ \psi(x_0+\Delta x,y_0) & \psi(x_0+\Delta x,y_0+\Delta y) & \psi(x_0+\Delta x,y_0+2\Delta y) & ... \\ \psi(x_0+2\Delta x,y_0) & \psi(x_0+\Delta x,y_0+\Delta y) & \psi(x_0+2\Delta x,y_0+2\Delta y) & ... \\ \vdots & \vdots & \ddots & \ddots \end{bmatrix} \tag{5}$$

This allows for the $\mathbf{p}_x^2$ to remain the same.

$$\mathbf{p}_x^2 = -\hbar^2\nabla_x^2 = -\hbar^2\frac{\partial^2}{\partial x^2} \tag{6}$$

In the 1D case we wrote the x-momentum operator as a matrix taking advantage of the finite difference method to write the 2nd derivative

$$\mathbf{p}_x^2 = -\frac{\hbar^2}{\Delta x^2}\begin{bmatrix} -2 & 1 & 0 & 0 & \dots \\ 1 & -2 & 1 & 0 & \dots \\ 0 & 1 & -2 & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots & \ddots \end{bmatrix} \tag{7}$$

using a more concise α and β as shorthand

$$\mathbf{p}_x^2 \left|\psi\right\rangle = \begin{bmatrix} \alpha & \beta & 0 & 0 & \dots \\ \beta & \alpha & \beta & 0 & \dots \\ 0 & \beta & \alpha & \beta & \dots \\ \vdots & \vdots & \vdots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \psi(x_0) \\ \psi(x_0+\Delta x) \\ \psi(x_0+2\Delta x) \\ \vdots \\ \psi(x_f) \end{bmatrix} \tag{8}$$

we can observe then that this matrix ALSO works for $\psi(x,y)$

$$\mathbf{p}_x^2\psi(x,y) =$$
$$\begin{bmatrix} \alpha & \beta & 0 & 0 & \dots \\ \beta & \alpha & \beta & 0 & \dots \\ 0 & \beta & \alpha & \beta & \dots \\ \vdots & \vdots & \vdots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \psi(x_0,y_0) & \psi(x_0,y_0+\Delta y) & \psi(x_0,y_0+2\Delta y) & ... \\ \psi(x_0+\Delta x,y_0) & \psi(x_0+\Delta x,y_0+\Delta y) & \psi(x_0+\Delta x,y_0+2\Delta y) & ... \\ \psi(x_0+2\Delta x,y_0) & \psi(x_0+\Delta x,y_0+\Delta y) & \psi(x_0+2\Delta x,y_0+2\Delta y) & ... \\ \vdots & \vdots & \ddots & \ddots \end{bmatrix}$$

At least in the case of $\mathbf{p}_x^2$, this representation works. The trouble beings when inspecting the case of $\mathbf{p}_y^2$. By observation, all I've found in the way of $\mathbf{p}_y^2$ is the following result

$$\mathbf{p}_y^2 \left|\psi\right\rangle = (\mathbf{p}_x^2\boldsymbol{\psi}(x,y)^T)^T \tag{9}$$
$$= \boldsymbol{\psi}(x,y)\mathbf{p}_x^{2\,T} \tag{10}$$

But at least with the methods we are using, I don't believe we can apply post-multiplication of matrices as an operator, as our Hamiltonian will be interpreted as pre-multiplying. so, instead I propose another method.

Similar to how matrices are stored in memory, we can instead encod all of our 2D data into a single 1D eigenvector (numpy array), keeping track of the 'stride'.

4 How: How is this implemented in code

5 How: How can this be used in the future