Computationally Solving the Time Dependent Schrodinger Equation

Gup Singh

Physics 77
University of California
Berkeley, CA
gupsingh@berkeley.edu

Abstract

The time dependent Schrodinger has been solved with powerful mathematical methods such as perturbation theory and varying principles to obtain approximations to solutions to the Schrodinger equation. With the increasing computational capabilities of modern-day computers, we were able to computationally solve the Time Dependent Schrodinger Equation for different potentials and wave functions. With the resulting wave functions we can plot the time evolution of the system. We chose to analyze aspects of the time step formulas to obtain some idea of where inaccuracies and faults come from. The first analysis was with the second derivative matrix being wrapped versus being unwrapped. The second piece of analysis focused on our dt or our time step. With the increasing number of iterations our wave functions values became numerically incorrect and chaotic.

1. Introduction

$$i\hbar\frac{\partial\Psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} + V\Psi.$$

A big part of introductory quantum mechanics is solving the Schrodinger equation for different potential and eventually applying perturbations. Solving the Time Dependent Schrodinger Equation tells us how the wave function will evolve over time. In quantum mechanics one learns to solve the time independent Schrodinger equation using separation of variables and applying that to various potentials such as the infinite square well. Later, methods for approximating solutions for perturbed potentials and time varying Hamiltonians are taught. Going through the process of learning these powerful and methods and then doing all of this in 100 lines of python made my project partners and I realize the power of computers.

2. Numerical Approaches

$$\mathrm{i}\hbar\frac{\partial\Psi}{\partial t} = H\Psi$$

The Schrodinger equation can be reduced to matrices multiplying each other. This makes the computational aspect straightforward.

```
array([[-2.,  1.,  0.,  0.,  1.],
       [ 1., -2.,  1.,  0.,  0.],
       [ 0.,  1., -2.,  1.,  0.],
       [ 0.,  0.,  1., -2.,  1.],
       [ 1.,  0.,  0.,  1., -2.]])
```

The second derivative matrix can be written as an n dimensional matrix as shown.

$$\begin{pmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_N \end{pmatrix}$$

The psi vector is a N dimensional column vector that is affected by our second derivative matrix and our Potential Matrix. We must compute multiple wave functions at different points in our discretized timeline or space. This is so that we can evolve the wave functions and animate it through code.

$$i\hbar\frac{\Psi(n+1)-\Psi(n)}{\Delta t} = H\Psi$$

Taking our partial wave function with respect to time we can numerically equate it to this substitution.

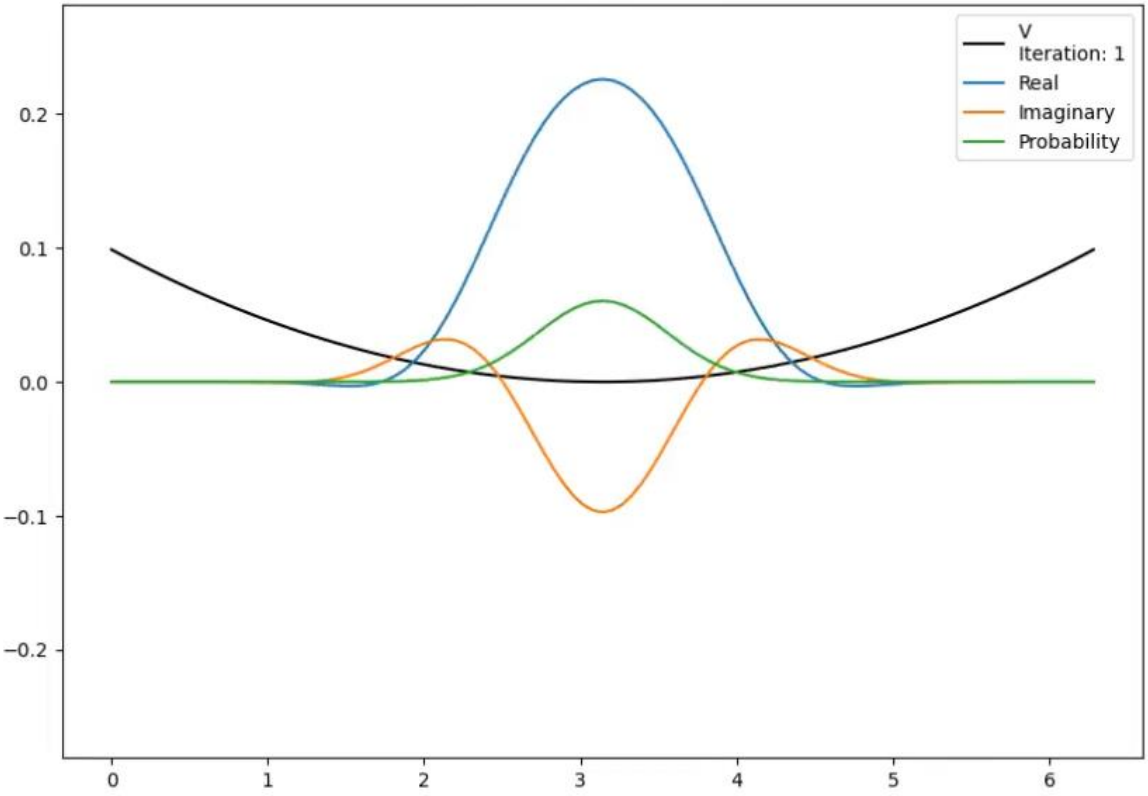
$$\Psi(n+1) = \frac{\Delta t}{ih}(H\Psi) + \Psi(n)$$

We must solve this psi(n+1) equation numerous times and store these values into a big array to animate and watch the wave function.

2. Results

In this project we successfully computationally solved the time dependent Schrodinger equation for wave functions varying with time given a specified potential. We then animated the time varying wave functions. The following are the resulting animations.

Harmonic Oscillator

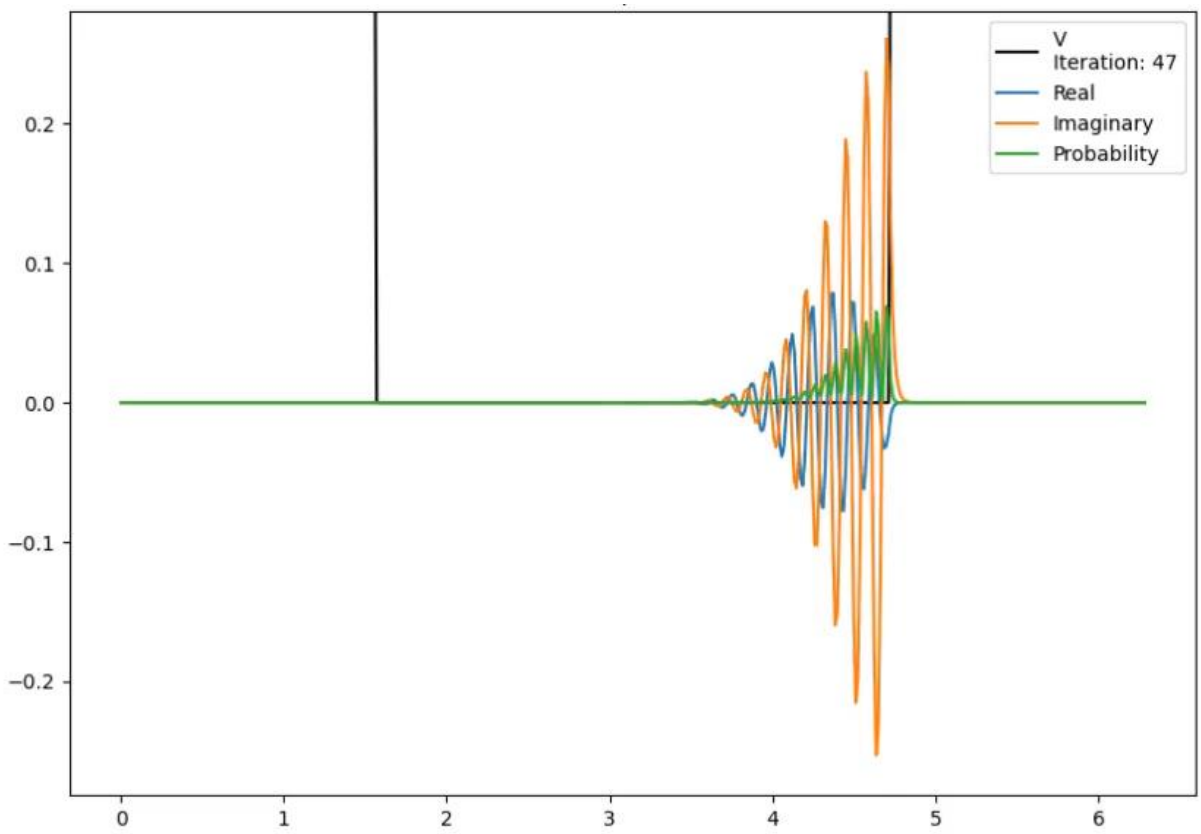


$$V(x) = \frac{1}{2}m\omega^2 x^2$$

This is a snapshot the Harmonic Oscillator potential wave function varying in time. In the animation the real and imaginary parts oscillate and the probability remains gaussian but

squishes and narrows slightly. We presume the squishing and narrowing is due to numerical complications in the code. If this were truly the harmonic oscillator potentially varying in time, the probabilistic wave (green) would remain stationary and not perturbate.

Infinite Square well

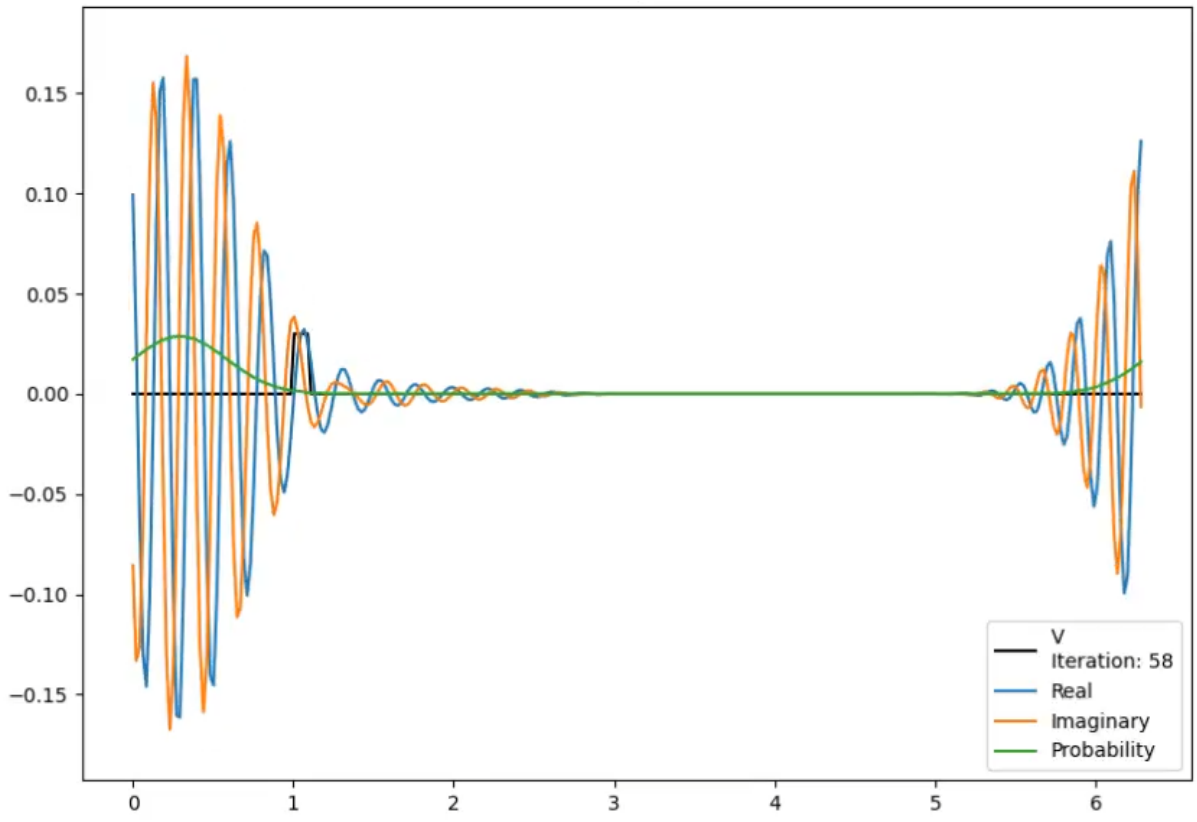


x=0.493 y=-0.064

$$V(x) = \begin{cases} 0, & 0 \le x \le a, \\ \infty, & \text{otherwise} \end{cases}$$

The animations are of a gaussian wave packet in a seemingly infinite square well. The wave packet oscillates back and forth as expected for particles in wells.

Small Bump



This is by far my favorite example of the various potentials we have simulated. A wave packet can be seen interacting with the small bump. In the animation a small part of the wave packet bounces back as well as tunnel through the bump as expected from the Schrodinger equation solution. This highlighted the

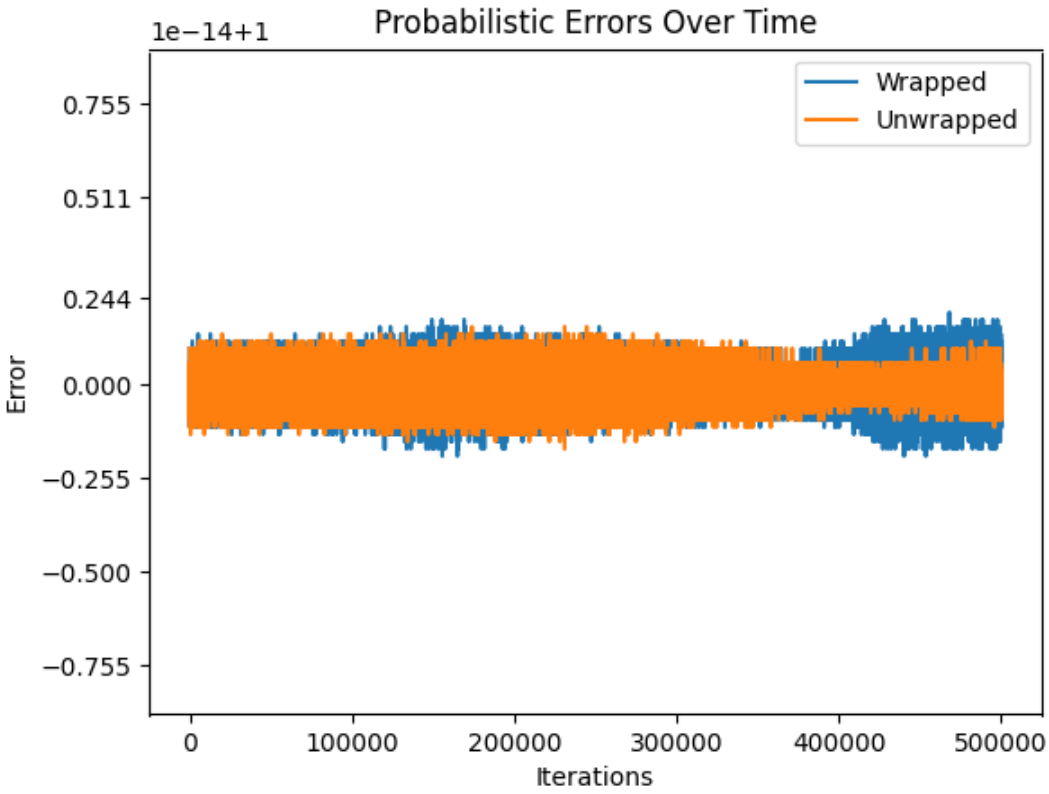
3. Analysis:

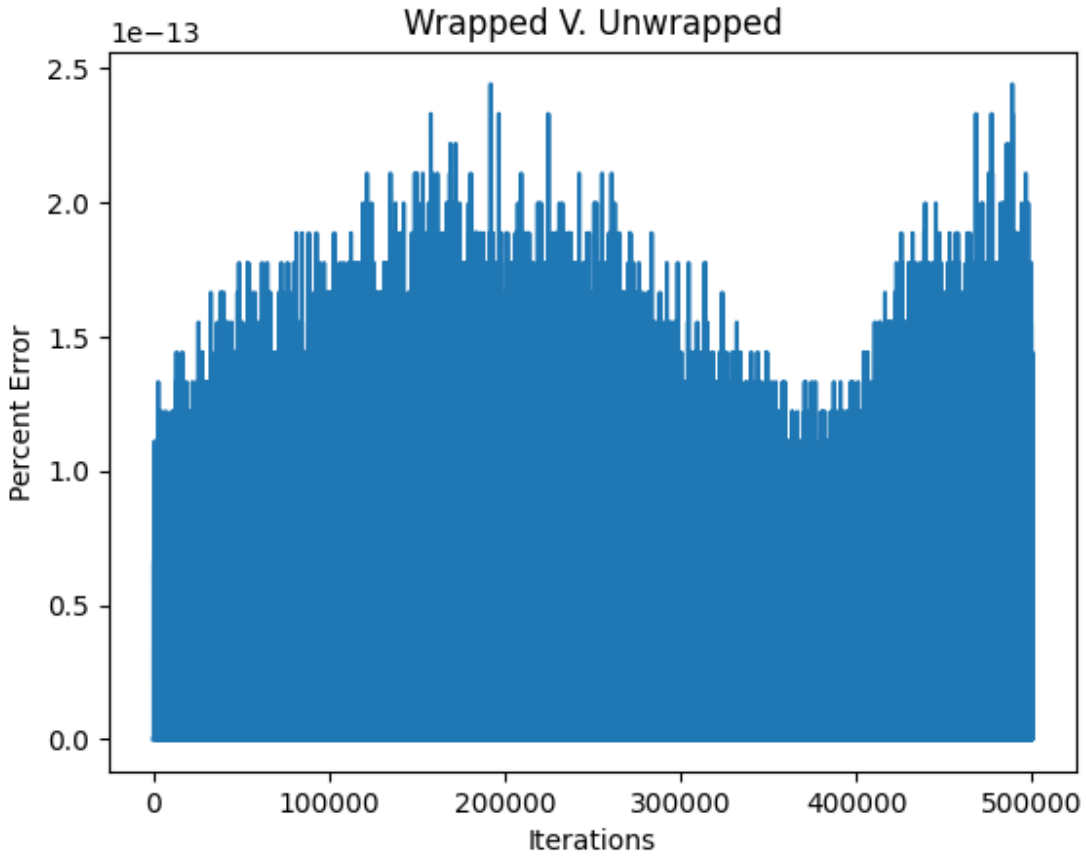
Once again, our first piece of analysis was looking at the numerical difference that the second derivative matrix has on our wave function values given that we remove the 0's in the top right and bottom left corner. We called these matrices wrapped and unwrapped, respectively.

```
array([[-2.,  1.,  0.,  0.,  1.],
       [ 1., -2.,  1.,  0.,  0.],
       [ 0.,  1., -2.,  1.,  0.],
       [ 0.,  0.,  1., -2.,  1.],
       [ 1.,  0.,  0.,  1., -2.]])
```

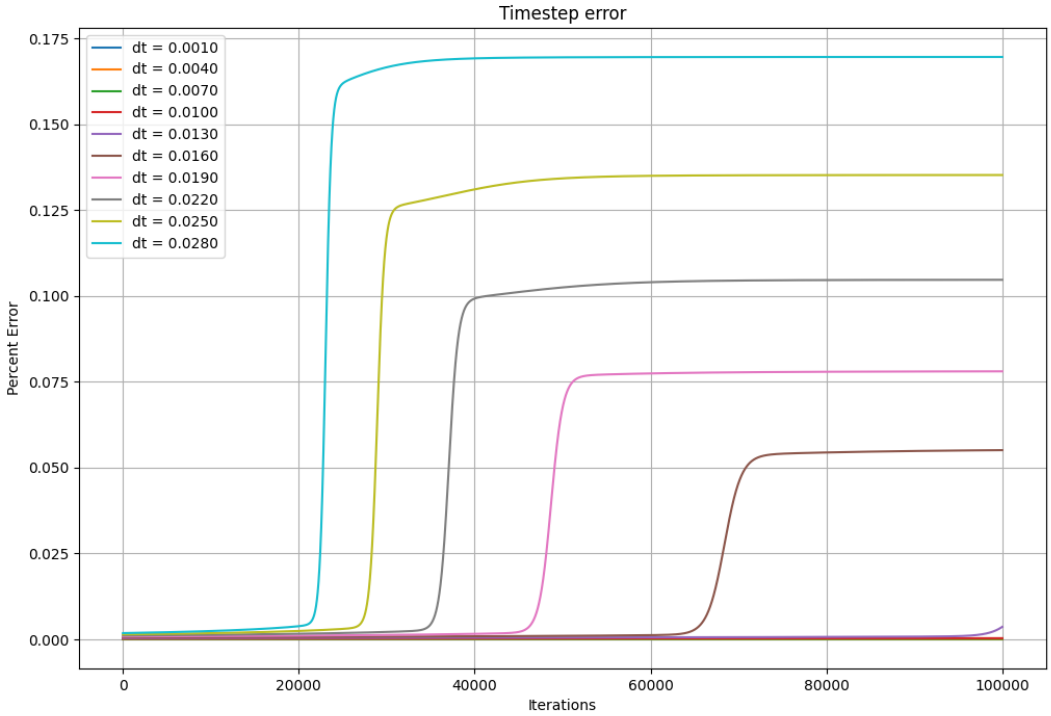
```
array([[-2.,  1.,  0.,  0.,  0.],
       [ 1., -2.,  1.,  0.,  0.],
       [ 0.,  1., -2.,  1.,  0.],
       [ 0.,  0.,  1., -2.,  1.],
       [ 0.,  0.,  0.,  1., -2.]])
```

Applying these second derivative matrices to our psi values and comparing the probability density for each interaction yielded the probabilistic errors shown below.


Probabilistic Errors Over Time
1e−14+1
0.755
0.511
0.244
0.000
−0.255
−0.500
−0.755
Error
0
100000
200000
300000
400000
500000
Iterations
Wrapped
Unwrapped



Percent Error of Wrapped and Unwrapped psi^2 values. The values were slightly more than 1 and the difference varied over the number of interactions as shown in the figure.



We also compared increasing values of our timestep dt. We found that as we increased out timestep the sooner the error jumps as interactions increases. To combat this issue, we had to tinker with our timestep (as well as our other parameters).

4. Conclusion

In summary, we successfully solved the time dependent Schrodinger equation for a specified wave function or wave packet with varying potentials. The resulting outputs we get from matrix multiplications we store in arrays and animate using the SESolver python library. The animation accurately simulates the behavior of quantum mechanics systems given a tweaked set of parameters and a small-time step.

5. Acknowledgements

I am grateful to Kenny Vetter, and Professor Yury Kolomensky

6. References:

Krueger, E. (2021, March 6). *A Simple Way To Time Code in Python*. Medium. https://towardsdatascience.com/a-simple-way-to-time-code-in-python-a9a175eb0172.

Griffiths, D. J., & Schroeter, D. F. (2019). *Introduction to quantum mechanics*. Cambridge University Press.

2016/03/30, D. (2017, March 17). *Python: Differentiation matrix by numpy*. Gappy Facets. http://gappyfacets.com/2016/03/30/python-differentiation-matrix/.