Quantum Mechanics on Python:
Numerical Investigations of Fun(ky) Quantum Phenomena

Kartik Tiwari
Prof. Phookun
9 December 2021

Contents

Acknowledgements

I would like to thank Prof. Bikram Phookun for providing me with the initial stimulus and references to start working on this project. My exploration of numerical relativity in Summer of 2021 under the supervision of Prof. Miguel Alcubierre was of paramount importance in equipping me with the necessary numerical skills required for completion of this and many other coding endeavours. Vipasha Barot has been a reassuring presence on the days I felt incompetent. Finally, I appreciate Mihir Nath's logistical support on the two nights which were the most productive for this project.

1. Introduction

Schrodinger equation governs the time evolution of a quantum wave-function.

$$\boxed{i\hbar\frac{\partial}{\partial t}\Psi(x,t) = \hat{H}\Psi(x,t)}$$

Solving and studying the solutions of Schrodinger equation for various potential distributions is a significant part of learning introductory Quantum Mechanics. However, analytic closed form solutions of Schrodinger equation can be found for only a few cases. Even in these handful cases, finding the analytical solution frequently requires employing mathematical 'tricks' which make the exploration of Schrodinger equation an opaque endeavour for the (already burdened) undergrad. Therefore, a numerical scheme to solve Schrodinger equation for any arbitrary initial condition and potential distribution is of immense value as it enables students to simulate and investigate various quantum mechanical scenarios. In this report, I discuss my general python code for solving the Schrodinger equation in 1 Dimension and its applications to some well understood interesting quantum phenomena.

2. Numerical Method

The Time Dependant Schrodinger Equation (TDSE) in 1D is a partial differential equation. Therefore, clearly a finite difference scheme is required to numerical integrate TDSE. (Robertson) discusses why the numerical scheme of choice for solving TDSE is the *Crank-Nicholson* technique. Let us briefly discuss the algorithm that needs to be implemented in `python`.

2.1 *Crank Nicholson Technique*

Crank Nicholson technique is a member of the general family of ImEx (Implicit-Explicit) techniques. Crank-Nicholson has the advantage of being unconditionally stable while not having to solve complete matrix equations (Baumgarte and Shapiro). This provides us with - at least in some ways - the best features of both explicit and implicit methods. As we will see, at each time instance, the program has to only solve a tri-diagonal matrix and this is considerably more efficient than solving for an arbitrary matrix equation. Let me give a brief run down of how the integrator is designed for this problem. We start with the Schrodinger equation

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t) = \hat{H}\Psi(x,t) \tag{1}$$

Using the Taylor series expansion about a small step Δx, we can write the

$$\Psi(x+\Delta x) = \Psi(x) + \Delta x \frac{\partial \Psi(x)}{\partial x} + \frac{1}{2}\Delta x^2 \frac{\partial^2 \Psi(x)}{\partial x^2} + \mathcal{O}(\Delta x^3) \tag{2}$$

Similarly, we can also expand using Taylor series about x using $-\Delta x$ to get

$$\Psi(x-\Delta x) = \Psi(x) - \Delta x \frac{\partial \Psi(x)}{\partial x} + \frac{1}{2}\Delta x^2 \frac{\partial^2 \Psi(x)}{\partial x^2} - \mathcal{O}(\Delta x^3) \tag{3}$$

On combining both of these expansions, we get

$$\Psi(x+\Delta x) + \Psi(x-\Delta x) = 2\Psi(x) + \Delta x^2 \frac{\partial^2 \Psi(x)}{\partial x^2} + \mathcal{O}(\Delta x^4) \tag{4}$$

By dropping the higher order terms rearranging, we can isolate the second derivative as a finite approximation

$$\frac{\partial^2 \Psi(x)}{\partial x^2} \approx \frac{\Psi(x+\Delta x) + \Psi(x-\Delta x) - 2\Psi(x)}{\Delta x^2} \tag{5}$$

Now, let us switch to grid notation on our discrete mesh of $n\Delta t$ and $i\Delta x$ points. Ψ_i^n is indexed spatially with the index i and temporally with index n. This means $\Psi(t, x \pm \Delta x) = \Psi_{i\pm1}^n$ or $\Psi(t \pm \Delta t, x) = \Psi_i^{n\pm1}$. Note that we have a great deal of freedom in choosing how we wish to formulate the discrete equation - the only constraint being that limit $\Delta x = \Delta t \longrightarrow 0$ should reduce our formulation to Schrodinger Equation. So, we can equivalently write the Schrodinger equation using the average of wavefunction at two time steps

$$i\hbar \left(\frac{\Psi^{n+1} - \Psi^n}{\Delta t} \right) = \frac{1}{2}(\hat{H}\Psi^{n+1} + \hat{H}\Psi^n) \tag{6}$$

We can rearrange to isolate the Ψ^{n+1} term to get

$$\Psi^{n+1} = \left(\frac{1}{1 + \frac{i\Delta t}{2\hbar}\hat{H}} \right) \left(1 - \frac{i\Delta t}{2\hbar}\hat{H} \right) \Psi^n \tag{7}$$

Recall the Hamiltonian operator $\hat{H}$ is defined as

$$\hat{H} = \frac{-\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x) \tag{8}$$

We can plug the finite difference approximation from Eq. 5 in the Hamiltonian operator and then plug the Hamiltonian operator back in Eq. 7 to get finite approximations both in space and time. After the substitutions and some algebra, we will get the following equation

$$\Psi_{i+1}^{n+1} + \Psi_{i-1}^{n+1} + A_i \Psi_i^{n+1} = B_i \tag{9}$$

where

$$A_i = -2 + \frac{4im\Delta x^2}{\hbar \Delta t} - \frac{2m\Delta x^2}{\hbar^2} V_i \tag{10}$$

and

$$B_i = -\Psi_{i+1}^n - \Psi_{i-1}^n + \Psi_i^n \left(2 + \frac{4im\Delta x^2}{\hbar \Delta t} + \frac{2m\Delta x^2}{\hbar^2} V_i \right) \tag{11}$$

Notice Eq. 9 has only Ψ^{n+1} on LHS and only Ψ^n on RHS. However, recall Crank Nicholson is not an explicit method and solving Eq. 9 requires solving a family equations. If we know the initial conditions Ψ_i^0 and the boundary conditions Ψ_0^n and Ψ_L^n, then we can formulate Eq. 9 in terms of a tridiagonal matrix equation of the following form

$$\begin{pmatrix} A_1 & 1 & 0 & 0 & ... \\ 1 & A_2 & 1 & 0 & ... \\ 0 & 1 & A_2 & 1 & ... \\ .. & .. & .. & .. & ... \\ 0 & 0 & .. & 1 & A_{L-1} \end{pmatrix} \begin{pmatrix} \Psi_1^{n+1} \\ \Psi_2^{n+1} \\ \Psi_3^{n+1} \\ ... \\ \Psi_{L-1}^{n+1} \end{pmatrix} = \begin{pmatrix} B_1 \\ B_2 \\ B_3 \\ ... \\ B_{L-1} \end{pmatrix} \tag{12}$$

This sort of tridiagonal matrix can be solved very efficiently using the Thomas algorithm which involves a forward sweep and a reverse sweep (**koonin**). The convenient thing about solving tridiagonal matrices is the fact that one does not even have to work with matrices in the first place. We can get by simply using vectors. In our case, the two off diagonals are simply 1s. So, effectively we only need to work with the main diagonal vector and the vector on the RHS to solve for the unknown Ψ^{n+1}. I provide a brief snapshot of the algorithm for solving family of equations for Ψ. First, we need two create two auxiliary arrays R and U. Now, we start with defining

$$U_1 = \frac{1}{A_1} \text{ and } R_1 = B_1 U_1 \tag{13}$$

from here, we start the forward sweep

$$U_i = \frac{1}{A_i - U_{i-1}} \text{ and } R_i = (B_i - R_{i-1}) U_i \tag{14}$$

for $i \in [2, 3...L-1]$. Now, we do the backward sweep to solve for Ψ^{n+1} using R and U. This would be done as follows

$$\Psi_i^{n+1} = \begin{cases} R_{L-1} & i = L-1 \\ R_i - U_i \Psi_{i+1}^{n+1} & i < L-1 \end{cases} \tag{15}$$

where i goes from $L-1, L-2, L-3...1$. In this way, we would have solved for Ψ^{n+1}. Now, we can simply repeat this procedure to get Ψ^{n+2} using Ψ^{n+1}. Also, note that we we are working with complex numbers in these simulations. To make a number complex in python we just puts a `j` next to the numeral. For example, $x = 5.6 + 3.1j$ would a complex number that can be decomposed as $x.real = 5.6$ and $x.imag = 3.1$. When generating arrays or plotting, one must ensure that the

complex datatype has been declared otherwise python might ignore the imaginary parts. The main python code that implements Crank Nicholson integration loop is the following -

```
for n in range(0, len(t)-2):                #Index n iterates of the time variables
    Psi[n, 0], Psi[n, -1] = 0, 0            #Fixed Boundary conditions (can be
    interpreted as infinite potential walls on each sides)

    for i in range(1, len(x)-2):            #Index i iterates over the space
    variables
        A[n, i] = -2  + (4j*m*dx**2)/(hbar*dt) - (2*m*dx**2)/(hbar**2)*V(x[i])
        B[n, i] = -Psi[n, i+1] - Psi[n, i-1] + Psi[n, i] * (2 + ((4j*m*dx**2)/(
    hbar*dt)) + ((2*m*dx**2)/(hbar**2))*V(x[i]))

    #Supplementary Matrices required for
    U[n, 1] = 1/A[n, 1]
    R[n, 1] = B[n, 1] * U[n, 1]

    #Foward Sweep
    for i in range(1, len(x)-2):
        U[n, i] = 1/(A[n, i] - U[n, i-1])
        R[n, i] = (B[n, i] - R[n, i-1])*U[n, i]

    N = len(x)-1
    i = N-1

    Psi[n+1, N] = R[n, N]

    #Backward Sweep
    while i>=1:
        Psi[n+1, i] = R[n, i] - U[n, i]*Psi[n+1, i+1]
        i -= 1
```

2.2 *Setting-Up the Problem*

To implement the Crank-Nicholson scheme for solving the Schrodinger equation, we need to set-up the problem appropriately. First we set up our spatial and temporal range. In the simulations that I run, I define $x \in [0, 3]$ and $t \in [0, 100]$ with $\Delta x = 0.01$ and $\Delta t = 0.001$. Next, we need to define the constant m corresponding to the mass in Schrodinger Equation and the reduced Planck's constant $\hbar$. Though the values of these constants can be found through a quick google search, the choice of appropriate units might require some thought (or some more googling). Taking into consideration the fact that we are working in quantum scales, an inappropriate choice of units can easily render our dynamics negligible. Conveniently for me, (Robertson) suggests a nice set of units to work with compatible with our range in x and t.

1. Length - *nanometers* ($10^{-9}m$)

2. Time - *femtoseconds* $(10^{-15}s)$

3. Energy - *electron-volts* $(\approx 10^{-19}eV)$

In these units, reduced Planck's constant is $\hbar = 0.6582eV.fs$ and electron's mass is $m = 5.68$. Now, another non-trivial choice is that of the boundary conditions. The simplest boundary conditions we can choose for this problem are $\Psi(x = 0) = 0 = \Psi(x = L)$. If the domain L is sufficiently larger compared to the dynamically significant region, then this fixed boundary conditions work because the wavefunction has the property of vanishing in the spatial limit tending to infinity. On the other hand, if the wave-packet during its dynamical evolution interacts with the fixed boundaries in ways significant to the simulation results, we can treat these boundary conditions as the physical equivalent of having infinite potential barriers at both ends which constraint the wave-packet in a box. There are other, more sophisticated, treatments of the boundary conditions which one can explore (Alcubierre) (Thijssen) but for the purposes of this project, the fixed boundary conditions suffice.

3. Standard Quantum Phenomena

In this section, I explore some quantum phenomena in different potentials. I mention briefly what our analytical treatments would lead us to expect for each case and then demonstrate the results of my simulations. Though I have not explicitly derived the analytical results, the closed form solutions that exist can be easily found in standard QM textbooks (Griffiths) (Sakurai).

3.1 *Wave Packet Evolution*

The simplest thing numerically that we can do is studying the free particle i.e. setting the potential $V(x) = 0$ everywhere. A particle located at $x = \mu$ with an uncertainty in its position measurements of $\Delta x = \sigma$ can be simulated using a Gaussian wave-packet of mean μ and standard deviation σ. The initial wave-packet corresponding to the particle is defined as

$$\Psi(t = 0) = \frac{1}{\pi^{1/4}\sqrt{\sigma}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \exp\left(ip_0 x/\hbar\right) \tag{16}$$

where μ is the position of the particle, σ is the uncertainty in position and p_0 is the average momentum of the particle. The initial state of the system is represented in Fig. 1.

The time evolution we expect from such a wave-packet via our analytical treatment is a characteristic spreading of the probability distribution in such a way that the wavefunction remains normalized. On performing the time evolution using the Crank-Nicholson integrator, we do observe (in Fig. 2) the spreading of the probability distribution of our wave-packet. But does this

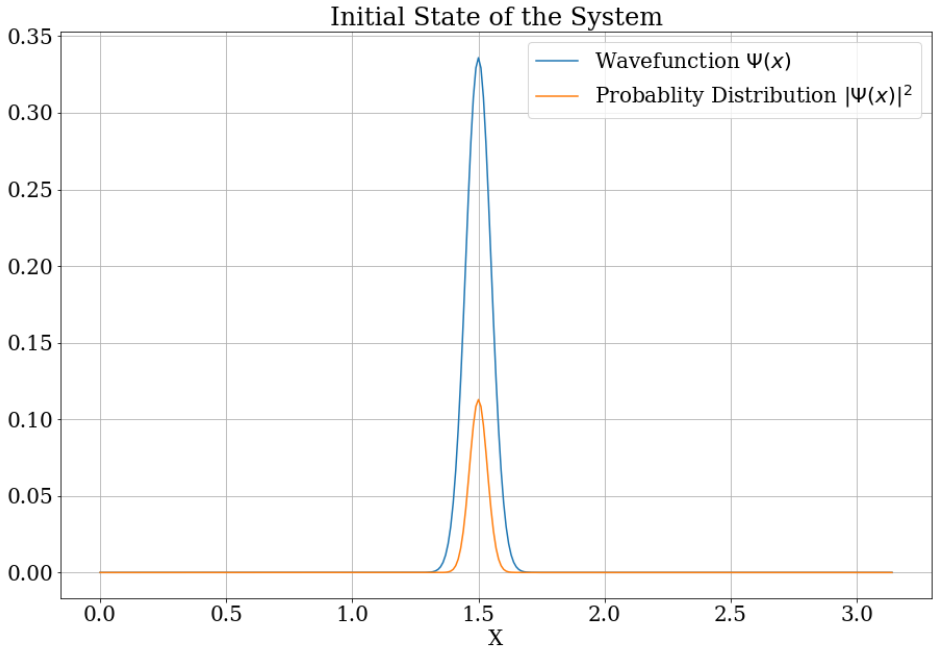

Figure 1: Initial Gaussian wave-packet representing a free particle (with $p_0 = 0$)

spreading meet the normalization constraint imposed by the physics of the system? We can check that explicitly at each time step by computing the sum of probability of finding the particle at each location. Fig. 3 shows clearly that the wavefunction indeed remains normalized during the time evolution. This normalization check was performed for each simulation and in each simulation a similar graph was achieved. To avoid redundancy, however, I took the wise decision of not putting the same graph four different times in the report.

3.2 *Infinite Square Well - Eigenstate*

The infinite potential well of length a has a family of energy eigen-functions which are characterized by the initial wave-function

$$\Psi_n(t=0) = \sqrt{\frac{2}{a}} \sin\left(\frac{n\pi x}{a}\right) \text{ where } n = 1, 2, ... \tag{17}$$

These wave-functions have the key feature of a probability distribution that is constant in time. Further, the time evolution of such eigenstates demonstrate sinusoidal oscillations of the wavefunction. I simulate such an eigenstate (where $n = 2$) for infinite square well potential and noticed that the numerical results are in conjunction with our theoretical understanding. I cannot create an infinite potential well using the finite memory of a computer. However, this potential can be approximated by having a difference in potential which is of a much much greater order of magnitude

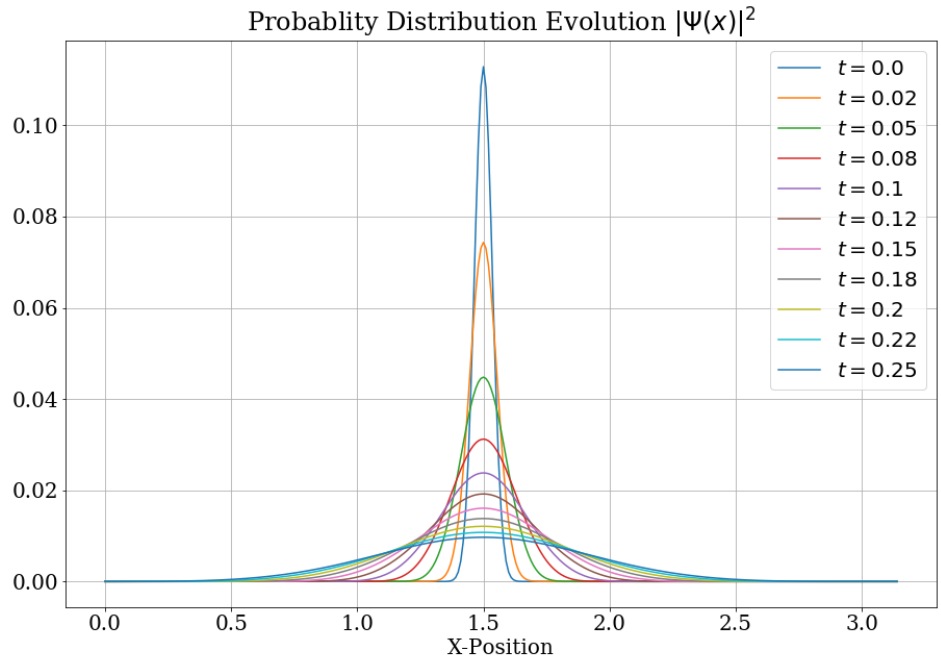

Figure 2: Time evolution of the wave packet in absence of any potential

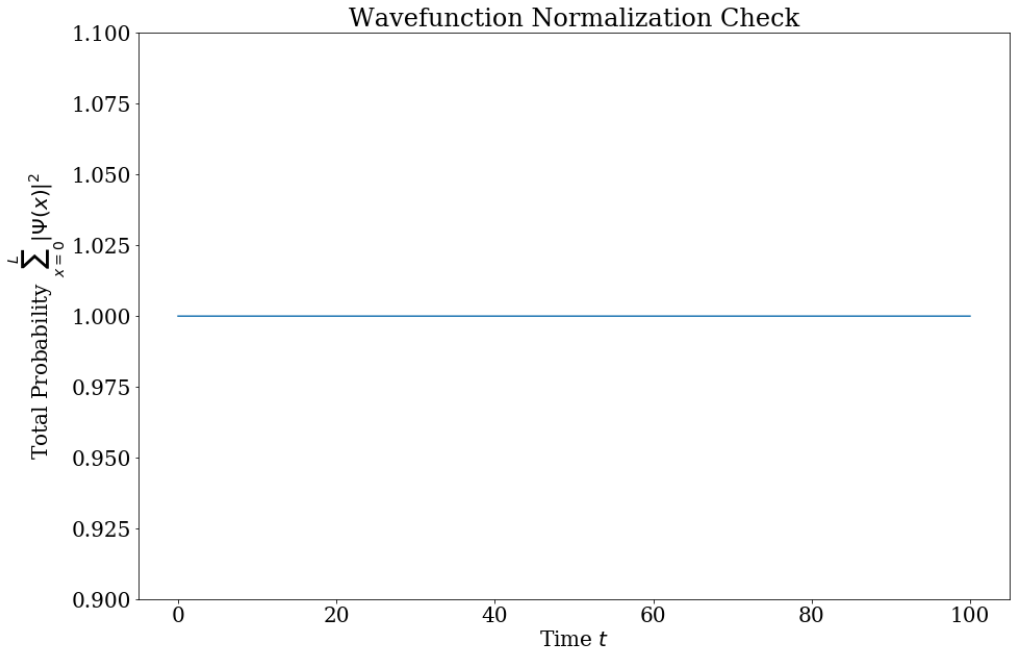

Figure 3: Explicit check for normalization of the wavefunction during time evolution

than that of the wavefunction. Fig. 4 displays the initial state of the system that was evolved in time and Fig. 5 and Fig. 6 are the results of the simulations. The probability distribution indeed stays constant in time (apart from minor numerical errors due to the finite difference approximation). Further, I also observed the sinusoidal oscillations of the real and imaginary parts of the wavefunction through time within the square potential well.

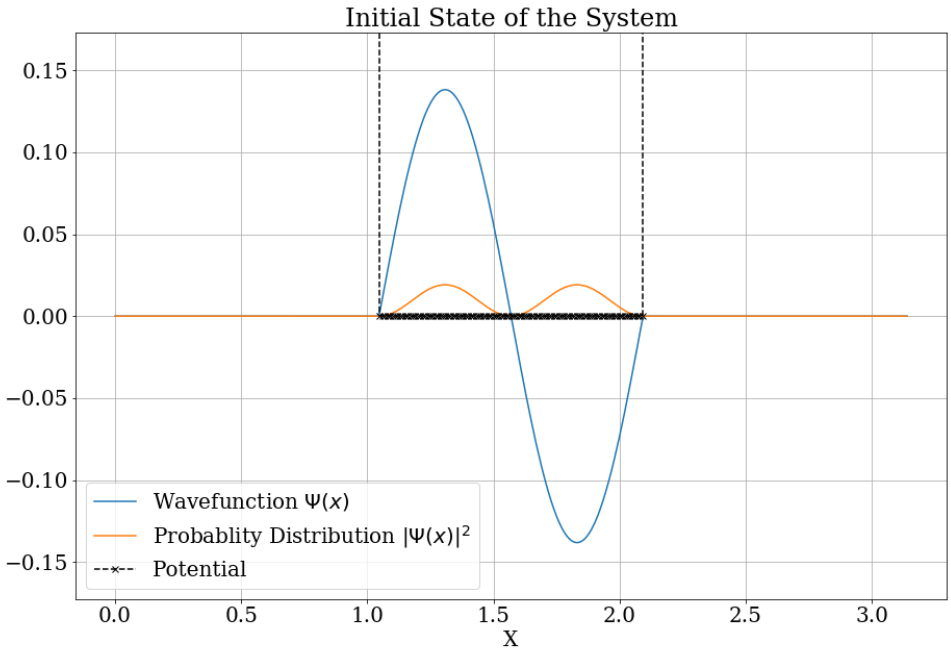

Figure 4: Initial state of energy eigen function (n=2) for infinite square well

3.3 *Quantum Tunneling through Potential Bump*

Another interesting phenomena studied in introductory QM courses is quantum tunneling of the wavefunction through various potentials. Classically, if a potential barrier is greater than the energy of an incident wave or particle, we expect no transmission. However, quantum mechanically, a part of the wavefunction can still seep through the potential barrier, thereby, rendering a non-zero probability of finding the particle having transmitted to the other side of the potential barrier. To start the simulation, we require a Gaussian potential bump and a Gaussian wave-packet. The energy of the Gaussian wave-packet is defined as

$$E = \frac{p_0^2}{2m} \tag{18}$$

In this simulation, we are taking $p_0 = 0$, which implies the $E = 0eV$ for the wave-packet incident on the potential barrier. On the other hand, as demonstrated in Fig. 7, the potential barrier is

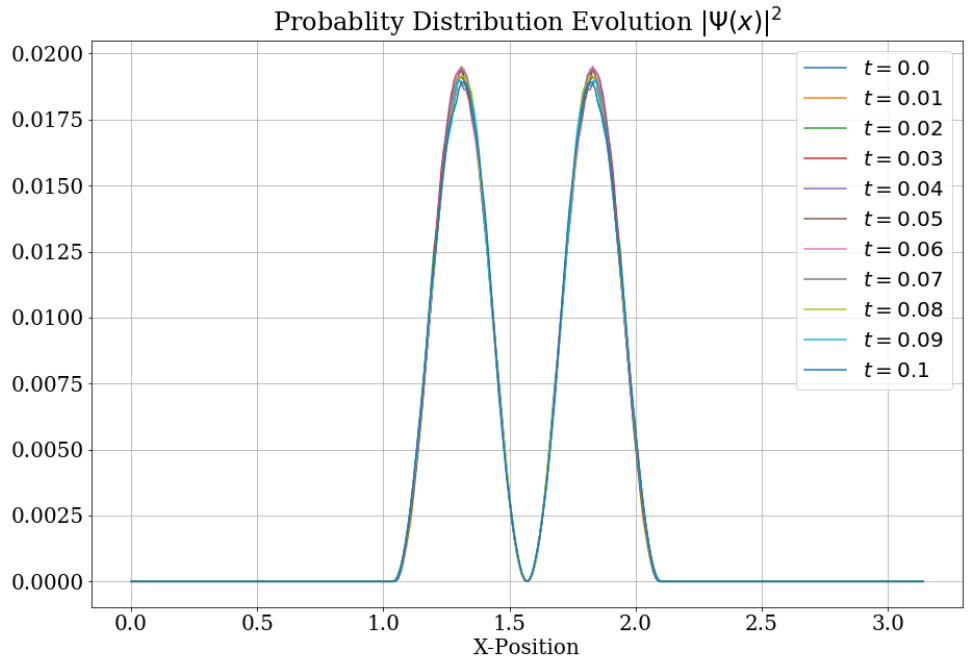

Figure 5: Time invariant probability distribution for infinite well potential

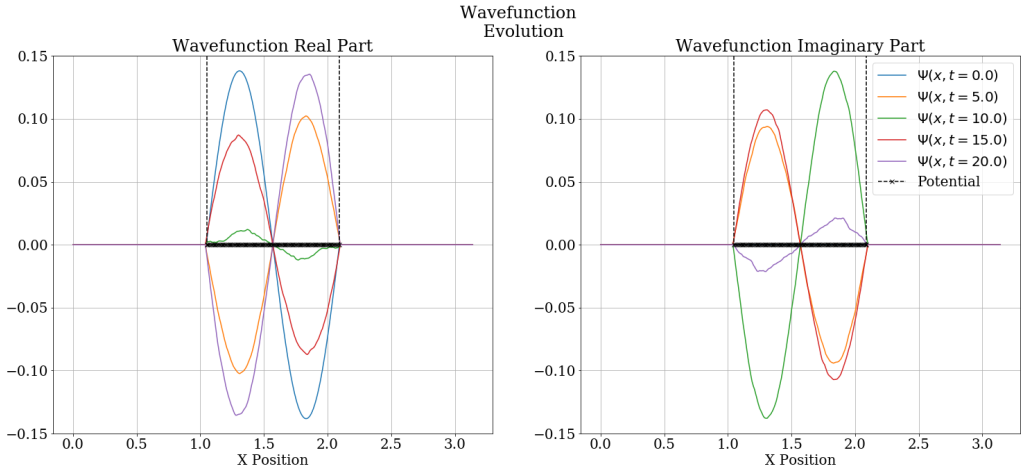

Figure 6: Sinusoidal oscillations of the eigenstate wavefunction

$0.75eV$,

As expected from the analytical solutions, a part of the wavefunction transmits through the potential bump (as seen in Fig. 9) that leads to a non zero probability of finding the particle 'behind' the potential barrier (as seen in Fig. 8)

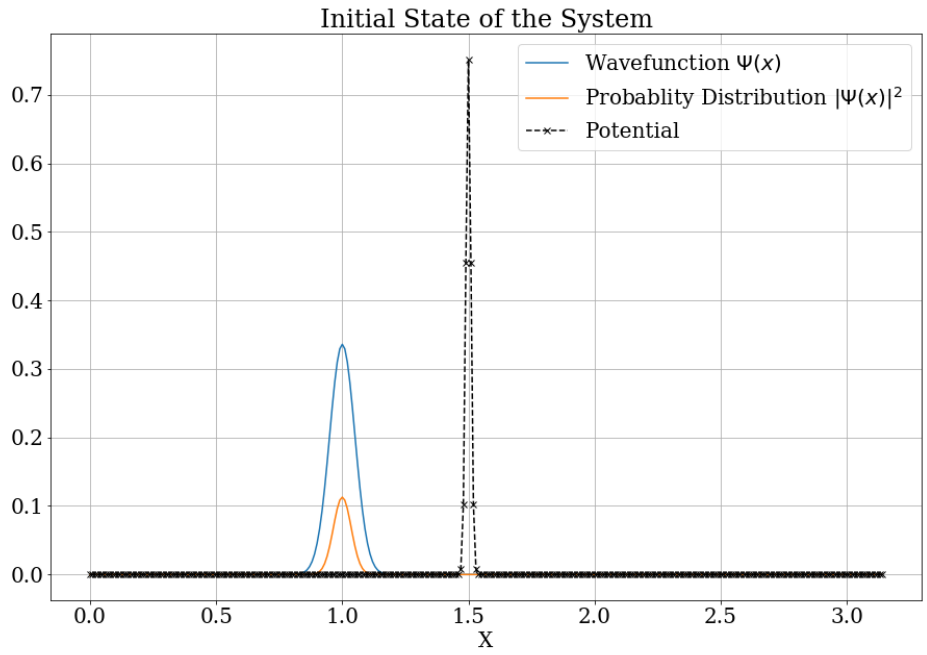

Figure 7: Initial condition to observe QM tunnelling through a Gaussian Bump. At $t = 0$, $\mu = 1$, $\sigma = 0.1$ and $p_0 = 0$ for the wave packet.

3.4 *Potential Step*

The final case that I consider is that of the finite potential step. In this simulation, I create a one-sided potential step of $V = 0.05eV$ and I give the initial wave-packet a forward momentum of $p_0 = 0.53eV.fs/nm$ which corresponds to an energy value of $E = 0.015eV$. Clearly, the energy of the incident wave-packet is less than that of the potential step. Classically, we would expect a complete reflection of the wavefunction with no transmission inside the region. However, as was the case in the last section, we observe quantum tunnelling in this case as well. The initial configuration of the simulation has been displayed in Fig. 10. Notice in Fig. 11, the probability amplitude decreases exponentially inside the potential step. The deeper inside the potential step we go, the less likely it is for the particle to be present there.

4. **Scope for Further Work**

Obviously, there are infinitely many potentials and initial conditions that one can solve Schrodinger equation for using the Crank-Nicholson integrator I programmed. An infinitely many of these configurations would also be infinitely boring to study. However, some interesting class of problems that I could not explore due to time constraints but would like to explore (and would urge to

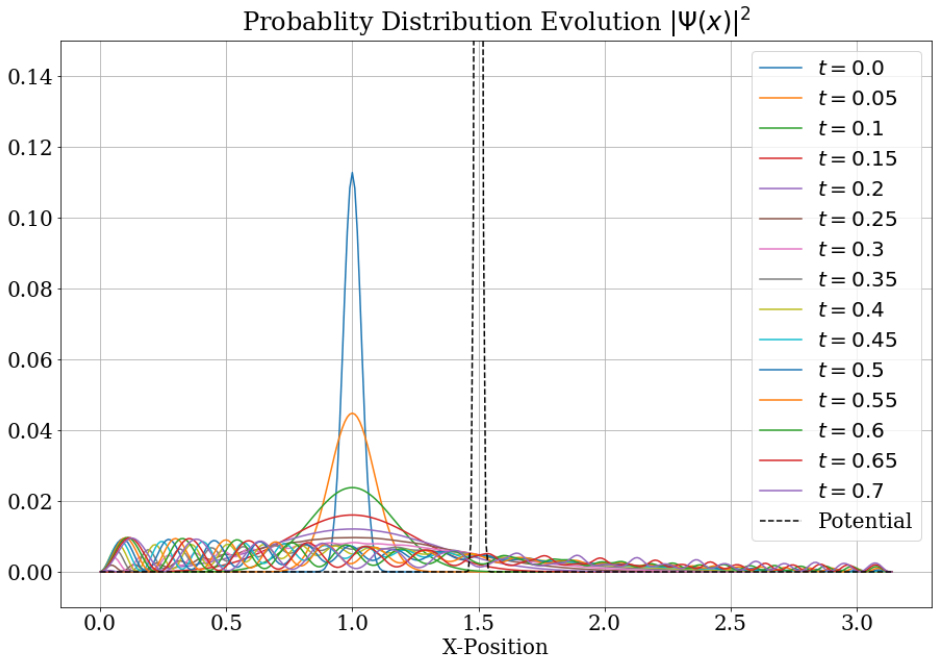

Figure 8: Minute disturbances in the probability distribution function to the right of the potential barrier demonstrate a non zero probability of finding the particle there.

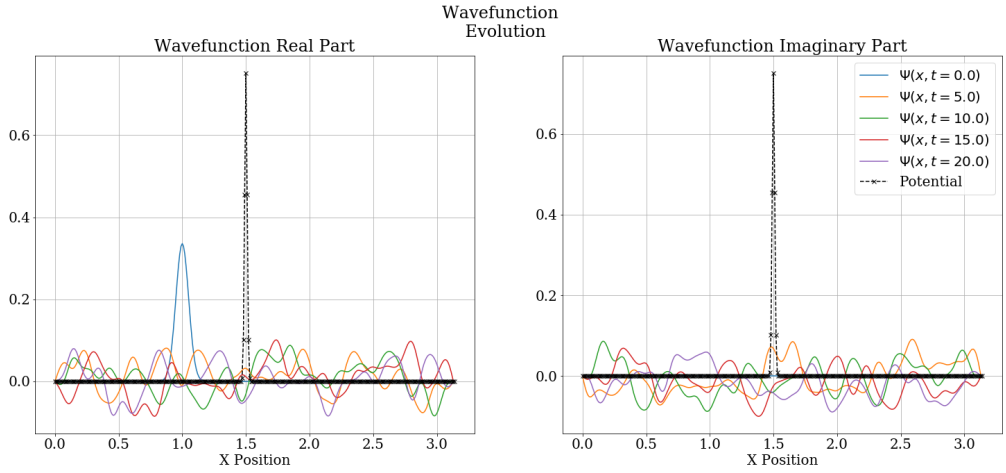

Figure 9: Initial wave-packet seeping through the potential barrier

reader to do so too) involve the Quantum harmonic oscillator potential with and without periodic perturbations. Analytical solutions for the Quantum harmonic oscillator exist against which the numerical results can be compared, should one wish to do so. Apart from this, a more in-depth numerical analysis of error, numerical dissipation, grid-size dependence, etc. can be performed for the program. I doubt though any of it would be as interesting as watching a nice animation of

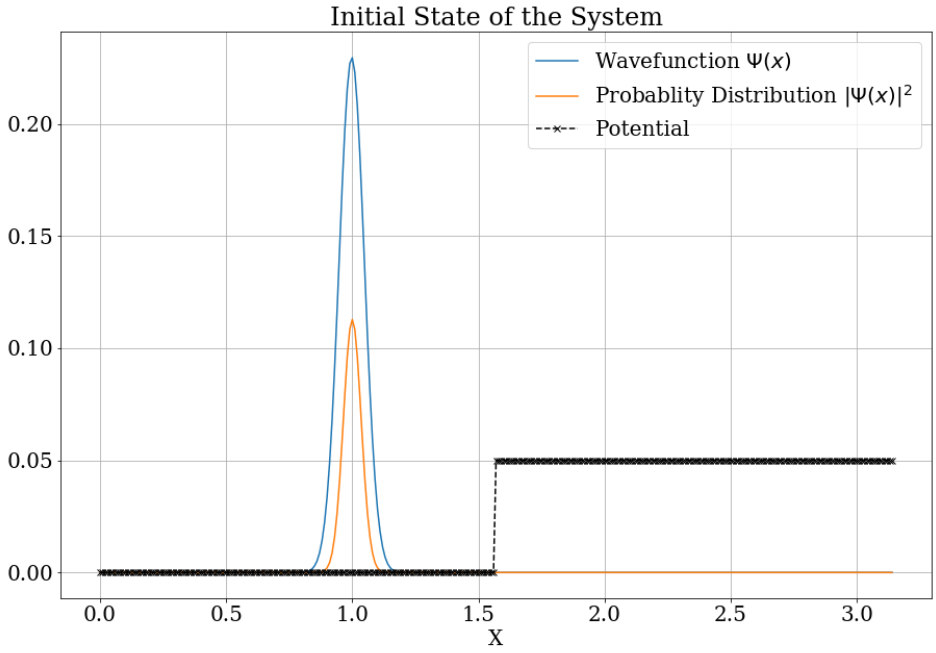

Figure 10: Initial condition to observe QM tunnelling through a Potential Step ($V = 0.05eV$)

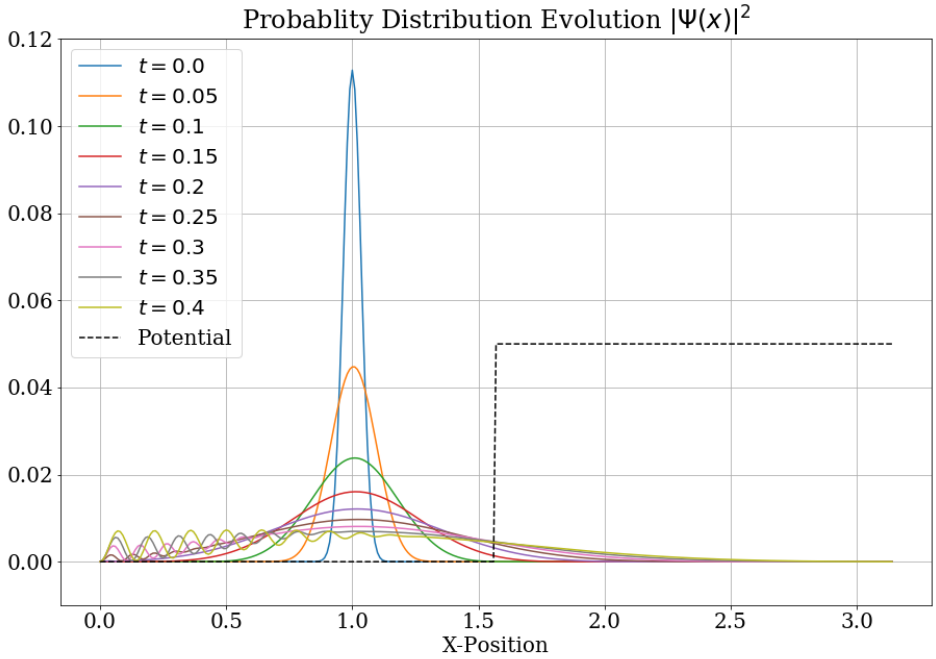

Figure 11: Probability distribution of a wave-packet quantum tunnelling through a potential step

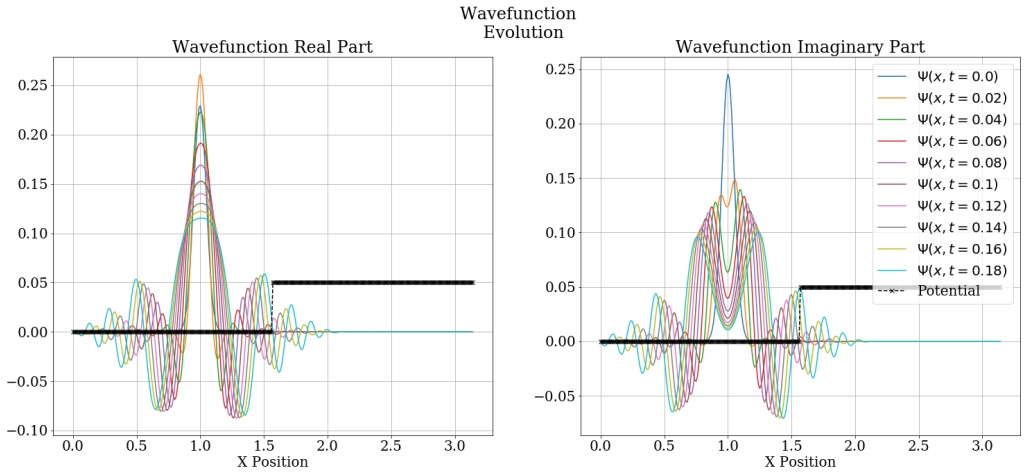

Figure 12: Time evolution of the wave-function scattering in the potential step simulation

quantum tunnelling generated by the program (but who am I to judge the interests of others!). As (Robertson) points out, one can also write programs using explicit methods and observe what is it that goes wrong there. Most importantly, the one thing that just might be within my current competence as a programmer is writing an efficient script to solve the Schrodinger in 3 spatial dimensions instead of just 1. This seems to be a nice computational challenge that one might want to take up in the summers. Finally, one can go the CS major route from here and try to translate the code in a more object-oriented fashion and make a nice library/package for doing quantum mechanics on python.

Works Cited

[1] Miguel Alcubierre. *Introduction to 3+1 numerical relativity*. International series of monographs on physics. Oxford: Oxford Univ. Press, 2008. DOI: 10.1093/acprof:oso/9780199205677.001.0001. URL: https://cds.cern.ch/record/1138167.

[2] Thomas W. Baumgarte and Stuart L. Shapiro. *Numerical Relativity: Solving Einstein's Equations on the Computer*. 2010.

[3] David Griffiths. *Introduction of Quantum Mechanics*. Prentice Hall, Inc., 1995.

[4] David G. Robertson. "Solving the Time-Dependent Schrodinger Equation." In: (Oct. 2011).

[5] Jun John Sakurai. *Modern quantum mechanics; rev. ed.* Reading, MA: Addison-Wesley, 1994. URL: https://cds.cern.ch/record/1167961.

[6] Jos Thijssen. *Computational Physics*. 2nd ed. Cambridge University Press, 2007. ISBN: 0521833469. URL: http://www.amazon.com/Computational-Physics-Jos-Thijssen/dp/0521833469/ref=ntt_at_ep_dpi_1.

Appendix: Animations

To better observe the dynamical evolution of wave-function, I generated some animations. There is no meaningful method of embedding the GIFs I had generated within this PDF report but if someone is interested in seeing them, they can write to me at kartik.tiwari9194@gmail.com. A sample python script used for generating the animations is listed below.

```
import matplotlib.animation as animation
import os

fig = plt.figure()
plt.xlabel('Position [x]')
plt.ylabel('Probablity Distribution [U]')
plt.grid()

plts = []              # get ready to populate this list the Line artists to be plotted
# plt.hold()
for i in range(0, int(len(t)/30), 1):
    p, = plt.plot(x, mod(Psi[i,:]**2), 'r')   # this is how you'd plot a single line...
    plts.append( [p] )           # ... but save the line artist for the animation

ani = animation.ArtistAnimation(fig, plts, interval=1, repeat_delay=3000)   # run the animation
ani.save('test.mp4', fps=24)          #Save the animation
os.system("ffmpeg -i test.mp4 test.gif") #Convert mp4 to GIF
```