Quantum Tunneling in Two Dimensions

Junjiang Li and Thomas Li

ABSTRACT

We numerically solve the time-dependent Schrödinger equation in two dimensions using the Split-Step Fourier Method (SSFM) to observe the quantum tunneling effect that occurs when a particle (modeled by a Gaussian wave packet) hits a potential barrier. Two additional types of potential energies, a smiling-face potential, and a charged-particle potential are provided to investigate the qualitative behavior of tunneling.

I. INTRODUCTION

At the dawn of the 20th century, physics underwent revolutionary changes. Among them was the development of quantum mechanics, which greatly improved our understandings of nature at a small scale. In quantum mechanics, everything is described as a matter wave defined by a wave function $\Psi(\vec{r},t)$ with spatial component $\vec{r}$ evolving with time t .

For many microscopic phenomena unexplainable by classical theories, such as alpha decay and long-range electron transfer in the light reaction of photosynthesis, quantum theory triumphed with the prediction of the peculiar "quantum tunneling" phenomenon, which refers to the particles' abilities of overcoming a high-potential-energy barrier that is classically forbidden.

Although the quantum tunneling behavior of a stream of particles in steady-state is well-understood, the one-particle dynamics is not. We aim to numerically solve the Schrödinger equation to visualize the time-evolution of a free-particle (modeled by a Gaussian wave packet) as it impinges on a potential wall. We also added different types of potential to qualitatively investigate their effects on the particle's tunneling behavior.

II. METHODS

In this section we outline the differential equation of our system and the underlying algorithm implemented in our program to solve the differential equation.

A. The Differential Equation

The differential equation governing out system is time-dependent Schrödinger equation. Written in two dimensions, the equation assumes the form of

$$\left[-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)+U(x,y,t)\right]\Psi(x,y,t) = i\hbar\frac{\partial}{\partial t}\Psi(x,y,t). \tag{1}$$

Part of solving the Schrödinger equation is to identify $U(x,y,t)$. In our problem, the potential energy is not time-dependent, so specifying $U(x,y)$ is enough. We prepared three types of potential energies — a potential barrier, a smiling-face potential, and a charged-particle potential. With the smiling-face potential, we hoped to see the effect of adding curvature to the potential wall on the tunneling behavior. With the charged-particle potential, we are modeling a physical situation where two charged particles with the same sign approach each other.

To solve the Schrödinger equation, we used the numerical method called the "Split-Step Fourier Method" (SSFM). It is numerically stable, relatively efficient, and easy to implement — it only requires forward integration. Below we give a brief discussion of the SSFM algorithm.

B. The SSFM Algorithm

To understand SSFM, It is helpful to first only consider the 1D case of (1), since the 2D algorithm is simply an extension to the 1D case. In 1D, equation (1) can be re-written as

$$\hat{\mathcal{H}}\Psi(x,t) = i\hbar\frac{\partial}{\partial t}\Psi(x,t), \tag{2}$$

where $\hat{\mathcal{H}} = \hat{\mathcal{T}} + \hat{\mathcal{V}}$ is the Hamiltonian, and $\hat{\mathcal{T}}$ and $\hat{\mathcal{V}}$ are kinetic and potential operators, respectively. Although $\hat{\mathcal{H}}$ is not a number, we ignore the fact for now and treat it as one. With that assumption, equation (2) can be integrated from t to $t+\Delta t$, leading to the evolution equation

$$\Psi(x,t+\Delta t) = e^{-i\Delta t\hat{\mathcal{H}}/\hbar}\,\Psi(x,t). \tag{3}$$

So far we have neglected the fact that $\hat{\mathcal{H}}$ is an operator. The meaning of exponentiating an operator is brought clear through a Taylor expansion, which leads to

$$\left[1-\frac{1}{1!}\left(\frac{i\Delta t}{2\hbar}\right)^1\hat{\mathcal{H}}^1+\frac{1}{2!}\left(\frac{i\Delta t}{2\hbar}\right)^2\hat{\mathcal{H}}^2+\dots\right]\Psi(x,t). \tag{4}$$

We see that exponentiating $\hat{\mathcal{H}}$ really is just asking us to apply the operator infinitely many times. But, this is not an easy task, especially since $\hat{\mathcal{T}}$ involves partial derivatives. We begin simplifying this task by separating $\hat{\mathcal{T}}$ and $\hat{\mathcal{V}}$. In general, $\hat{\mathcal{H}}$ does not commute, which means $e^{-i\Delta t\hat{\mathcal{H}}/\hbar} \neq e^{-i\Delta t\hat{\mathcal{T}}/\hbar}e^{-i\Delta t\hat{\mathcal{V}}/\hbar}$. Nonetheless, we can approximate the expression with

$$e^{-i\Delta t\hat{\mathcal{H}}/\hbar} \approx e^{-i\Delta t\hat{\mathcal{V}}/2\hbar}e^{-i\Delta t\hat{\mathcal{T}}/\hbar}e^{-i\Delta t\hat{\mathcal{V}}/2\hbar}. \tag{5}$$

The error introduced by this approximation goes as $\mathcal{O}(\Delta t^3)$. We can again take the taylor expansion of each exponential term, and the results will be (4) where each occurance of $\hat{\mathcal{H}}$ will be replaced by $\hat{\mathcal{V}}/2$, or $\hat{\mathcal{T}}$.

We know $\hat{\mathcal{V}} = U(x)$ in position space. Therefore, $e^{i\Delta t\hat{\mathcal{V}}/2\hbar} = e^{-i\Delta tU(x)/2\hbar}$. $\hat{\mathcal{T}}$ is not so simple in the real space, however, since it includes partial derivatives. But, $\hat{\mathcal{T}}$ will assumes a similar form as $\hat{\mathcal{V}}$ does in real space, if we switch to momentum space. This switching is done by a Fourier Transform of $\Psi(x,t)$, since the product of the Fourier transform gives the wavenumber composition of the wavefunction, which is directly proportional to momentum ($\vec{p} = \hbar\vec{k}$). It is fairly

straightforward to show that KE $= \hbar^2 k^2/2m$, therefore the operation $e^{-i\Delta t\hat{\mathcal{T}}/\hbar}\Psi(x,t)$ can be expressed as

$$e^{-i\Delta t\hbar^2 k^2/2m\hbar}\mathscr{F}[\Psi(x,t)] \tag{6}$$

where $\mathscr{F}$ represents the Fourier transform. In summary, to apply $\hat{\mathcal{V}}$, we have to move the wavefunction to real space, and replace $\hat{\mathcal{V}}$ by $U(x)$. To apply $\hat{\mathcal{T}}$, we have to move to momentum space, which is done by taking the Fourier transform of $\Psi(x,t)$, and replace $\hat{\mathcal{T}}$ by $\hbar^2 k^2/2m$. The algorithm for propagating $\Psi(x,t)$ through time Δt is therefore

$$\Psi(x,t+\Delta t) = e^{-i\Delta t U(x)/2\hbar} \times \mathscr{F}^{-1}\left\{e^{-i\Delta t\hbar^2 k^2/2m\hbar} \times \mathscr{F}\left[e^{-i\Delta t U(x)/2\hbar} \times \Psi(x,t)\right]\right\} \tag{7}$$

Extending the algorithm to 2D is straightforward. Namely, we replace $U(x) \mapsto U(x,y)$, and $\hbar^2 k^2/2m \mapsto \hbar^2(k_x^2 + k_y^2)/2m$.

III. CODE IMPLEMENTATION

First, a 2-D rectangular space of size [-L, L]×[-L, L] is discretized into cells of size dx. Our particle wave function, modeled by a Gaussian wave packet (given below), is then sampled over the discretized space.

$$\Psi(x,0) = e^{-(x^2+y^2)/(2\epsilon)^2} e^{ik_0 x}$$

The user can control the spread of the wave function by modifying the standard deviation (ϵ), labeled in program as eps. The initial momentum along the x direction can also be specified using the parameter k0. Depending on which type of potential did the user select, a $U(x,y)$ matrix (denoted in the program as U), also discretized according to the discretization of space, will be generated by one of the routines below. The program prompts the selection of the type of $U(x,y)$.

1. **Potential Barrier** All potential barriers are centered at $x = 0$ and aligned parallel to the y axis for simplicity. Upon the selection of this type of potential, the program will ask for half of the total length of the potential wall, denoted in program as LWall, and fill all cells with corresponding x coordinates between -LWall and LWall with the height of the wall, U0. The parameter U0 is a part of the function call to the main program. This type of potential is built in to the main program and requires no extra functions to be generated.

2. **Smiley Face** To generate the two eyes, we filled grids with coordinates satisfying the relationship $(x - x_0)^2 + (y \pm y_0)^2 \leq r^2$ where $x \leq x_0$. x_0, y_0 and r are predetermined. The mouth are the region enclosed by ellipses $x^2/a^2 + y^2/c^2 = 1$ and $x^2/b^2 + y^2/c^2 = 1$, where $x \geq 0$. Upon the selection of this type of potential, the program will prompt the user to input parameters a and b that control the width of the mouth. All regions (the eyes and the mouth) will be set to a height of U_0 as specified in the function call to main (the main program).

3. **Point Charge Potential** The potential energy between a point charge q_0 centered at the origin and a test charge q is given by $U = k_e q_0 q/r$. We simplify this in our code as $U = k/r$, where, upon the selection of this type of potential, U0 will be used as the k in $U = k/r$. With this type of potential, we made sure that there is not a cell corresponding to $x = 0$ and $y = 0$, otherwise $U \to \infty$. Although MATLAB supports this type of data, it causes error in other calculations. If the parameters supplied by the user causes a $(x,y) = (0,0)$ cell to be created, the program will terminate and prompt the user to change the input.

Because position space is discretized, so will the momentum space (referred to as k-space) after Fourier Transform. Specifically, the spacing between cells in k-space is dk=pi/(N*dx), and the range in k-space is [-N*dk, N*dk]×[-N*dk, N*dk], where 2N+1 is the total number of grids along each axis in position space. From this, a matrix of $k_x^2 + k_y^2$, as a part of the SSFM algorithm, can be easily calculated.

Translation of the SSFM algorithm given in equation (7) to MATLAB code is simple. Every multiplication indicates an element-wise multiplication, and MATLAB by defaults handles exponentiation of matrix as element-wise exponentiation, which is the desired behavior. However, it is worth noting that the fft2 function, which is the fast-fourier transform for matrices, do not put the 0 frequency element in the center. Therefore, the function fftshift is called after each Fourier transform to reposition the elements. Conversely, ifftshift is called before the inverse Fourier transform, so that MATLAB can correctly transform the wave function back to position space.

The SSFM algorithm might not conserve normalization, and therefore the wave function is normalized after each iteration of equation (7). A subfunction normalize is defined to perform this task.

It is important to note that the SSFM algorithm naturally incorporates Periodic Boundary Conditions. Therefore, the wave function will "loop back" to itself if the run time is too large and the space size (L) is too small. The final time of simulation, tf, should be kept at a moderate value so that no "wrapping around" occurs.

The output of this program is a movie, which plots the time-evolution of the probability density profile. Due to technical restrictions, the figure window has to be continually refreshed so that MATLAB samples the correct frame to build the movie. Therefore, while the program is running, there will be figure window constantly popping in and out. A progress bar, not written by us, is therefore provided to help keep track of the estimated remaining time of the program. In general, with the parameters we have tested (details below), the program takes at most 10 minutes to finish, which is not unreasonable.

IV. RESULTS

To begin, we tested the validity of our code by specifying a free-particle wave situation, with the function call main(10, 0.05, 0, 1, 1, 3, -5, 0.5e-3, 3);.

Notice that the third entry (which will be U0, see help main) is 0. When the program prompted to select a type of potential, we chose 1 and set LWall to 0. It is expected that the wave packet will simply travel with dispersion, and this is indeed what we observe. This output, along with all outputs, will be stored in the google drive folder with link https://drive.google.com/drive/folders/1sLQXX6K_uRcqw_AXwlW5rQMGpD6snszG?usp=sharing This shows that our code handles the SSFM algorithm correctly.

We further tested a free-space with a potential energy of 10 everywhere. This is done by calling the function with arguments main(10, 0.05, 10, 1, 1, 3, -5, 0.5e-3, 3);, selecting 1 at the list of potentials, and setting LWall to 10. The result (named Free-Particle with Global Potential) also showed a free-wave traveling with dispersion, which is the desired output since biasing space by a constant potential will not change anything. This, however, shows that our program handles potential energy correctly.

We first investigated the simplest situation — a potential wall centered at $x = 0$. The function call to the program is main(10, 0.05, 10, 1, 1, 2, -3, 1e-3, 3);, with the type of potential being 1 and LWall being 0.05. The potential generated along with the time-evolution movie are stored in the same folder described above with the name "Potential Wall (U0=10, LWall=0.05)". Qualitatively, the behavior looks like what is expected of a tunneling behavior. We subsequently changed LWall to 0.1 and 0.15, while keeping U0 unchanged. The subsequent outputs are uploaded as well. We see that the tunneling probability is highly non-linear as a function of LWall — significant probabilities lie at $x > 0$ (after the wall) when LWall$= 0.05$, while virtually none is left at LWall$= 0.1$. This is further seen from LWall$= 0.15$, where essentially tunneling does not occur.

At LWall$= 0.05$, we also varied U0 to 15 and 20 to see the effect of wall height on tunneling probabilities. As expected, raising U0 decreases the probability of tunneling, but not by the same extreme extent as seen in raising LWall.

The smiling-face potential, called with parameters

```
main(20, 0.05, 5, 1, 1, 3, 0, 1e-3, 5);
The semi-major radius of lower lip >> 5
The semi-major radius of upper lip >> 4.3
```

produced an output with the name time_evolution_smile. Qualitatively, the behavior made sense, because the mouth is nothing much but a curved potential wall. It can also be seen that the part of wave function that tunneled through the potential barrier has a curvature impinged, which is due to its interaction with the potential wall. Towards the end of the movie, the interaction of the wave function with the eyes can also be seen, where interference developed due to the close proximity between the eyes.

Finally, we called the point-charge potential with parameters main(10, 20/401, 1, 1, 1, 3, -3, 0.5e-3, 2.5);, and the result is of the name of time_evolution_point_charge. We saw that the wave function is sliced in half, a rather interesting behavior since it does not correspond to our intuition that at least some will be reflected back. We invoked the function again with a much smaller eps, in hope to make the wave nature of the particle less pronounced. The function call is main(15, 20/401, 1, 1, 0.1, 3, -0.5, 0.25e-3, -1), with result stored under the name time_evolution_point_smalleps, and indeed more reflection was observed — the wave packet behaves more classically as expected. A smaller dt is used to avoid error in calculation, which is present if the function is invoked with the same parameters except dt=0.5e-3.

Overall, our code was accurate and obtained results that are reasonable and are expected. They enabled us to visualize the dynamics of tunneling, and also to investigate qualitatively the effects relevant parameters might have (such as the height and width of the potential wall, or the curvature of it) on the tunneling phenomenon. In the future we could modify our code so that it runs several simulations in parallel with the same parameters but one, and we can thus juxtapose the videos to better see the effect that a variable might have.

Also, the run time of our code can be improved. Right now, for decently complex calculations, the run time takes 5-10 minutes, which is not fast. The computation time increases quickly with grid size, which makes simulation of more complex processes expensive. Improvements to the standard SSFM algorithm, or even a new algorithm, can be adopted to accelerate the calculation. Making sure grid sizes are powers of 2 may also speed up the computation.

V. CONCLUSION

In conclusion, we numerically solved the Schrödinger equation under various physical conditions to investigate the quantum tunneling effect. Specifically, we qualitatively studied the behavior of tunneling under various factors (such as width of wall and height of wall), and also looked at more-fun potentials and how they might influence the behavior. The outputs, by and large, agree with intuition and previous theoretical analysis.

VI. CONTRIBUTIONS

The introduction, parts of code implementation, and results section of this report are written by Thomas Li, and the rest are written by Junjiang Li. The main function is written by Junjiang Li, and the other functions are written by Thomas Li. We feel we have equally contributed to the project.