Numerical Schrödinger Equation: Expanding on a Basis

Yaojia Huang & 01213992

November 10, 2018

Abstract

In quantum well problems, finding the analytical solutions of wave functions in wells of various shapes is a complex and time-consuming process. To avoid cumbersome calculations while enhance the accuracy of the solutions, we started solving one-dimensional potential well by using the solution of a infinite well / a set of plane waves as basis. Then we added one more dimension to the system and compressed the Hamiltonian matrix to two dimensions so that it can be diagonalised. Finally, we analysed the behaviour of a particle in 1D wells as well as in 2D wells.

1 Introduction

In 1926, Erwin Schrödinger, an Austrian physicist, published a groundbreaking mathematical equation that became the keystone of the study of quantum mechanics, the Schrödinger equation (SE).

Contrary to Newton's laws that describe motions of macroscopical objects, the SE effectively describes particles, in the quantum scale, as waves with possible locations and time evolution. Over decades, efforts have been devoted in finding the solution of the SE due to its wide applications in chemistry and physics, such as semiconductor hetero-structures with quantum wells[1]. Many methods have been devised to solve the SE, but only a few potentials are analytically solvable; therefore, many solutions are approximated by using numerical methods[2].

The aim of this project is to write a Python program that solves the time independent Schrödinger equation (TISE) with various potentials. The first attempt was made by constructing the solutions based on waves in an infinite potential well. Then, in order to reduce the error of the solution, the Fourier transform was adopted, and the solution has a set of plane wave as basis. In both approaches, diagonalising the Hamiltonian matrix was the key method to get the original wave function and its eigenenergy.

In the 2D situation, the Hamiltonian matrix, which was originally 4D, was truncated and compressed into 2D in order to make the diagonalisation possible. Then the solution was obtained by using the inverse Fourier transform.

2 Theory

The TISE can be expressed as

$$\hat{H}\boldsymbol{\Psi}_n = E_n\boldsymbol{\Psi}_n, \tag{1}$$

where $\hat{H}$ is the Hamiltonian operator that characterises the sum of the kinetic energy and potential energy of the wave function, $\boldsymbol{\Psi}_n$ is the wave function of the system, E_n is the energy state of the wave function. The Hamiltonian operator is given by

$$\hat{H} = -\frac{\hbar}{2\mu}\nabla^2 + \hat{V}, \tag{2}$$

where the first term on the RHS is the kinetic energy operator, in which μ is the mass of the particle, and the latter is the potential energy operator.

Assume that the basis is given by a set of functions denoted by $\boldsymbol{\Phi}$, then, in a certain potential well, any solution of the SE can be expressed as

$$\boldsymbol{\Psi}_n = \sum_{m=1}^{\infty} c_{nm}\boldsymbol{\Phi}_m. \tag{3}$$

Then $\boldsymbol{\Psi}_n$ can written in the vector form

$$\boldsymbol{\Psi}_n = (c_{n1}, c_{n2}, c_{n3}, ..., c_{nm}) \tag{4}$$

in the basis of $\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2, \boldsymbol{\Phi}_3, ...$

According to Equation(1), when the Hamiltonian operates on any wave function $\boldsymbol{\Psi}_n$, it returns the eigenenergy and the wave function itself. By combining Equation(1), Equation(4) and the Hamiltonian matrix, denoted as H_{nm}, where $n, m \in N^+$, the SE becomes

$$H_{nm} = \begin{bmatrix} c_{11} & c_{21} & c_{31} & \dots & c_{n1} \\ c_{12} & c_{22} & c_{32} & \dots & c_{n2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{1m} & x_{2m} & c_{3m} & \dots & c_{nm} \end{bmatrix}^{-1} \begin{bmatrix} E_1 & & & \\ & E_2 & & \\ & & \ddots & \\ & & & E_n \end{bmatrix} \begin{bmatrix} c_{11} & c_{21} & c_{31} & \dots & c_{n1} \\ c_{12} & c_{22} & c_{32} & \dots & c_{n2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{1m} & x_{2m} & c_{3m} & \dots & c_{nm} \end{bmatrix} \tag{5}$$

Therefore, $\boldsymbol{\Psi}_n$ is the eigenvector and E_n is the eigenvalue of H_{nm}, which can be obtained by

$$H_{nm} = \langle\boldsymbol{\Phi}_n|\,\hat{H}\,|\boldsymbol{\Phi}_m\rangle = \langle\boldsymbol{\Phi}_n|\,\hat{H}_0 + \hat{V}\,|\boldsymbol{\Phi}_m\rangle \tag{6}$$

$$H_{nm} = E_n^{(0)}\delta_{nm} + \int_0^a {\boldsymbol{\Phi}^*}_n V(x)\boldsymbol{\Phi}_m dx, \tag{7}$$

where $\hat{H}_0$ is the Hamiltonian of the basis, $E_n^{(0)}$ is the eigenenergy of the basis, δ_{nm} is the Kronecker delta function and $V(x)$ can be any potentials between 0 and a.

The size of H_{nm} should be infinite, but taking a fairly large number of the first several terms (e.g. $n = m \leqslant 50$) can still give an approximate result. For H_{nm} of a finite size, the eigenvectors and the eigenenergies can be calculated by diagonalising H_{nm}. Since the coefficients of the components of each wave function are known, the wave function can be built by using Equation(3), and the probability function can be obtained by taking the square of the wave function.

2.1 1D infinite well basis

In a 1D infinite well centred at $x = \frac{a}{2}$ with width a, the wave functions are strictly confined such that the solutions can only have the following form:

$$\mathbf{\Phi}_n(x) = \begin{cases} \sqrt{\frac{2}{a}} \sin\left(\frac{n\pi x}{a}\right) & 0 \leq x \leq a \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

and the eigenenergy of the infinite well solution is

$$E_n = \frac{n^2\pi^2\hbar^2}{2\mu a^2}[3]. \tag{9}$$

2.2 1D Fourier transform

By using Fourier transform, the solutions of the wave function are based on a set of plane waves that is expressed as

$$\mathbf{\Phi}_n(x) = \frac{1}{\sqrt{a}} e^{-ik_n x}, \tag{10}$$

and the eigenenergy of the plane wave set is

$$E_n = \frac{\hbar^2 {k_n}^2}{2\mu}. \tag{11}$$

By using Equation(7) and Equation(10), the potential matrix, P, can therefore be written down as

$$P_{nm} = \langle \mathbf{\Phi}_n | \, V \, | \mathbf{\Phi}_m \rangle = \frac{1}{a} \int_0^a V(x) e^{-i(k_m - k_n)x} dx. \tag{12}$$

Equation(12) can be expressed in the form of the Fourier transform of the potential function, which is

$$P_{nm} = \frac{1}{a} \int_0^a V(x) e^{i\tilde{k}_l x} dx. \tag{13}$$

where $\tilde{k}_l = k_n - k_m$.

3 Implement

3.1 Solving the SE based on the solution of an infinite well in 1D

According to Equation(7), Equation(8) and Equation(9), for any proper potentials, $V(x)$, the Hamiltonian can be obtained by

$$H_{nm} = \frac{n^2\pi^2\hbar^2}{2\mu a^2}\delta_{nm} + \frac{2}{a}\int_0^a \sin\frac{n\pi x}{a} V(x) \sin\frac{m\pi x}{a} dx. \tag{14}$$

By choosing a proper range for n and m and using every possible combination of them, all the elements of the Hamiltonian matrix can be calculated, and their row and column positions

are determined by n and m respectively. Then, normalised eigenvectors and eigenvalues can be obtained by diagonalising the matrix by python.

The eigenvalues, or eigenenergies, returned by the program are not arranged in any specific order. For the purpose of finding the order of energy states, the eigenenergies were ranked from lowest to highest, and the eigenvectors were also sorted so that each of them corresponds to its original eigenenergy.

The eigenvectors obtained, Ψ_n, have the same form as Equation(4); therefore, the vectors could be converted to wave functions by applying Equation(3) and Equation(8). The probability functions are just the square of the waves functions. By using a loop in python, all the wave functions and probability functions were calculated in the same manner and shown by graphics.

3.2 Solving the SE based on plane waves in 1D

Using plane wave basis adopts a similar method as above, but requires the application of discrete Fourier transform.

Firstly, an odd number of samples are taken uniformly from the potential function. By passing the sample size, $2N+1$, and its range, a, to the *fftfreq* function of the numpy library in python, the frequency, k, of each possible base function is generated. The total number of the frequencies returned by the program is also $2N+1$ and the frequency array has the following form:

$$k = \Big(-\frac{N}{a}, -\frac{(N-1)}{a}, ..., 0, ..., \frac{(N-1)}{a}, \frac{N}{a} \Big) \quad (\text{let } k_i = i/a). \tag{15}$$

As the number of points gets more, the resultant wave function would be more accurate. Note that the problem would be complicated if an even number of points were chosen[4].

Since the frequencies are known, according to Equation(7) and Equation(11), the eigenenergies of the plane waves could be calculated and a $(2N+1)\times(2N+1)$ kinetic energy identity matrix, denoted as K, could be constructed by putting the eigenenergies along the diagonal.

The Hamiltonian matrix is the sum of K and the potential matrix, P; therefore, P must also have dimension $(2N+1)\times(2N+1)$. According to Equation(13), $\tilde{k}_l = k_n - k_m$, so the range of $\tilde{k}$ has $4N+1$ elements that can be listed as $[-\frac{2N}{a}, -\frac{(2N-1)}{a}, ..., \frac{2N}{a}]$. Let $\tilde{k}_i = i/a$, and because k_i is also defined as i/a in Equation(15), it can be shown that $\tilde{k}_{n-m} = k_n - k_m$. The values of P_{nm} corresponding to these $\tilde{k}$s need to be found.

In order to solve the problem efficiently, the *Fast Fourier Transform* method(FFT) was used. This method takes discrete values from the original function as inputs, makes a Fourier transform and returns a set of values in the k-space. Using the discrete form of Equation(13) and taking $4N+1$ samples from [0, a], the potential energy matrix becomes

$$P_{nm} = f(\tilde{k}_{n-m}) = \frac{1}{4N+1} \sum_{v=1}^{4N+1} V(x_v) e^{i\tilde{k}_{n-m}x_v}. \tag{16}$$

The equation can be solved through

$$F = \frac{1}{4N+1} FFT(V(x_1), V(x_2), ..., V(x_{4N+1})), \tag{17}$$

where FFT stands for the *Fast Fourier Transform* method in the numpy library.

The FFT would return a list of numbers:

$$F = \Big(f(\tilde{k}_{-2N}), f(\tilde{k}_{-2N+1}), ..., f(\tilde{k}_{2N-1}), f(\tilde{k}_{2N})\Big), \tag{18}$$

which has all the elements of the potential energy matrix.

The sum of the kinetic matrix and potential matrix gives the Hamiltonian matrix. The eigenvectors can be obtained by diagonalisation. However, the elements in eigenvectors are frequency components; therefore, the bases cannot be simply combined. In order to get the solution, the inverse Fourier transform was used on each eigenvector (e.g.*numpy.fft.ifft(vector)*). Then the wave functions and their probability functions could be obtained.

3.3 Solving the SE based on plane waves in 2D

In a 2D situation, the domain was defined as $a \times a$ and it was divided into grids with $2N+1$ points on each side. Two frequency arrays(k^x and k^y) of each side were given by the *fftfreq* method. Then, a matrix was obtained by using $meshgrid(k^x, k^y)$ to combine the arrays. The eigenenergy in 2D is

$$E_{nm} = \frac{\hbar^2[(k_n^x)^2 + (k_m^y)^2]}{2\mu}. \tag{19}$$

By using this equation to calculate every tuple in the matrix, the kinetic matrix could be obtained.

However, in order to take all possible combinations of k^x and k^y under a certain energy, some elements in the matrix must be discarded. Consider an arbitrary matrix of (k^x, k^y) below:

$$\begin{bmatrix} (-2,-2) & (-1,-2) & (0,-2) & (1,-2) & (2,-2) \\ (-2,-1) & (-1,-1) & (0,-1) & (1,-1) & (2,-1) \\ (-2,0) & (-1,0) & (0,0) & (1,0) & (2,0) \\ (-2,1) & (-1,1) & (0,1) & (1,1) & (2,1) \\ (-2,2) & (-1,2) & (0,2) & (1,2) & (2,2) \end{bmatrix}$$

Figure 1: An Arbitrary Matrix of (k^x, k^y)

If only basis wavefunctions whose eigenenergies are under or equal to $\frac{\hbar^2}{2\mu}$ were needed, then any elements outside the smaller parallelogram should be discarded; similarly, if the targeted eigenenergy is $\frac{4\hbar^2}{2\mu}$, then only elements within the large parallelogram should be kept. Therefore, the matrix is truncated so that frequencies whose energies are not higher than $E_{1,N+1}$ (use $E_{1,N+1}$ because all possible combinations of frequencies whose energies are less than it can be found in the given matrix) are kept. Then the truncated matrix is reduced to a 1D array by appending the $(2N+1)$-th row after the $2N$-th row, then the $2N$-th row after the $(2N-1)$-th row and so on. Denote this array as T and create a kinetic energy array that corresponds to T. A diagonal kinetic energy matrix can be constructed by putting the kinetic energy array along a zero square matrix of dimension $2N^2 + 2N + 1$, which is the number of elements in the parallelogram.

Similar to Equation(12), the equation for the 2D potential matrix can be expressed as

$$P = \frac{1}{a^2}\iint_A V(x,y)e^{i[(k_n^x-k_q^x)x+(k_m^y-k_p^y)y]}dxdy \quad (20)$$

$$= \frac{1}{a^2}\iint_A V(x,y)e^{i[(k_n^x x+k_m^y y)-(k_q^x x+k_p^y y)]}dxdy, \quad (21)$$

where $m,n,p,q \in N^+$. Define (k_n^x, k_m^y) and (k_q^x, k_p^y) as $\vec{k}_u$ and $\vec{k}_v$, then $\vec{k}_u - \vec{k}_v$ must have $(4N+1)\times(4N+1)$ possibilities(the argument is similar to the one in the last section, but in this case there are two dimensions).

$(4N+1)\times(4N+1)$ samples are selected uniformly from the region of the potential and Equation(21) is turned into the discrete form:

$$P = \frac{1}{(4N+1)^2}\sum_{i=1}^{4N+1}\sum_{j=1}^{4N+1} V(\vec{r}_{ij})e^{i(\vec{k}_u-\vec{k}_v)\vec{r}_{ij}}, \vec{r}_{ij} = (x_i, y_j). \quad (22)$$

Then the *2D Fast Fourier Transform* method(FFT2) is applied. FFT2 is similar to FFT but takes a matrix instead of an array as the input, and outputs a matrix of values in the frequency space. The value of the potential function at each sample site is calculated. FFT2 takes the matrix of the values of the potential function and returns a $(4N+1)\times(4N+1)$ matrix. However, the base frequency vectors, $\vec{k}_u - \vec{k}_v$, actually has less possibilities because both of $\vec{k}_u$ and $\vec{k}_v$ are confined in the parallelogram discussed previously. In fact, $\vec{k}_u - \vec{k}_v$ is confined in a larger parallelogram in the output matrix illustrated by Figure(2). Therefore, many elements in the matrix must be set to 0 except for those within the parallelogram.

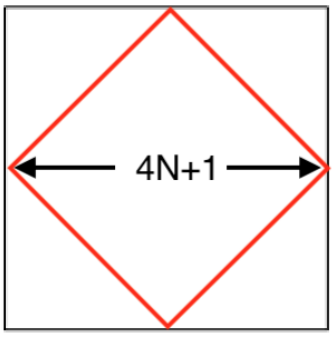


Figure 2: The Matrix returned by FFT2

Only elements inside the parallelogram are kept; others are set to be 0 because they are not calculated from the targeted combinations of (k^x, k^y).

After the above process, the $(2N^2+2N+1)\times(2N^2+2N+1)$ potential energy matrix needs to be filled by the elements in the parallelogram in order. Remember that all valid $\vec{k}_u$ or $\vec{k}_v$ are stored in T, the valid $\vec{k}_u - \vec{k}_v$ can be indicated by $T[i]-T[j]$. Since for each $\vec{k}_u - \vec{k}_v$, there is a value for potential energy according to Equation(22), it can be shown that

$$P_{ij} = \frac{1}{(4N+1)^2}\sum_{i=1}^{4N+1}\sum_{j=1}^{4N+1} V(\vec{r}_{ij})e^{i(T[i]-T[j])\vec{r}_{ij}}.\vec{r}_{ij} = (x_i, y_j). \quad (23)$$

The number of elements in T is $2N^2+2N+1$, so the potential energy matrix is fully filled.

The kinetic and potential matrices are already known, so the Hamiltonian can be obtained by adding them together. By diagonalisation, the eigenvectors could be calculated. Take one of the eigenvectors, place the i-th element of the vector at the coordinate that $T[i]$ originally belongs to in a zero $(2N^2 + 2N + 1) \times (2N^2 + 2N + 1)$ matrix, and use inverse FFT2 to solve this matrix to get the wave function. Squaring the wave function in 2D also gives its probability function.

4 Error, Results and Discussion

4.1 Infinite well basis

The probability functions of an electron in a finite well obtained by using infinite well basis are displayed in Figure(3).

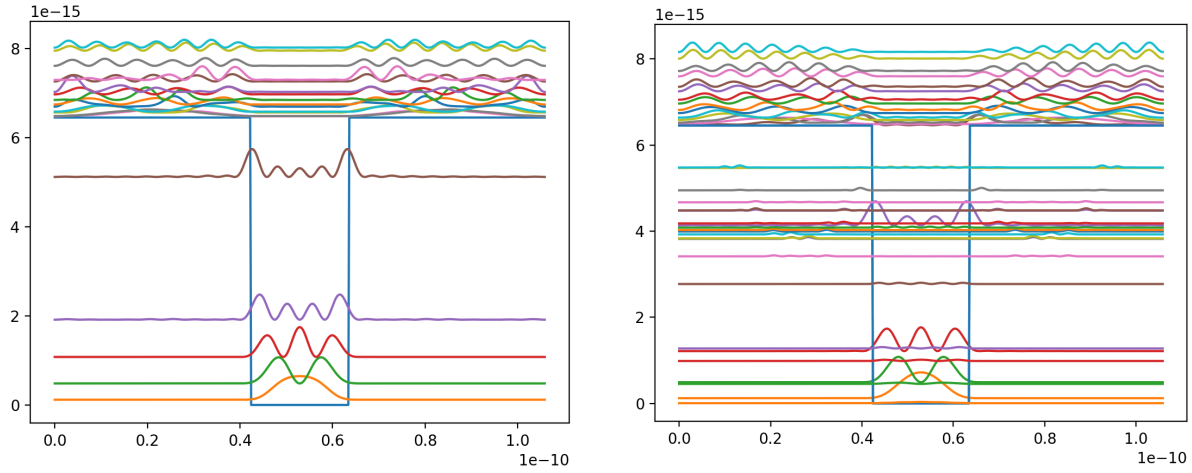
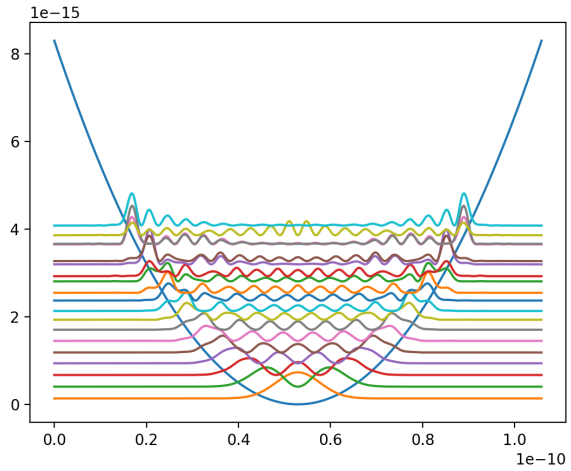
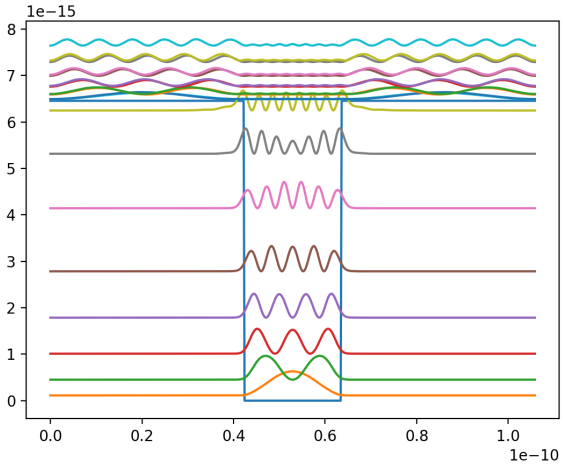


Figure 3: Probability Function of an Electron in a Finite Potential Well
The probability functions on the left are based on 24 solutions of the infinite well, while those on the right have 50 bases.

In Figure(3), it's obvious that the probability functions become more disordered as more bases are used, which does not seem right because each function should converge as the number of bases increases. This eccentric behaviour is probably due to the error made by integration in the program. Python uses continuous functions to approximate the integrant; therefore, any discontinuous integrant would cause larger error than a continuous one. Figure(4a) shows functions in a harmonic potential well that also have 50 bases, but they are not so much distorted because the harmonic potential is continuous.



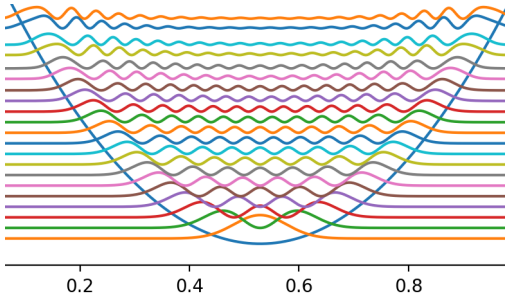
(a) Probability Function of an Electron in a Harmonic Well
The graphs in this figure use 50 bases. Functions at lower energy states are closer to the actual solutions.



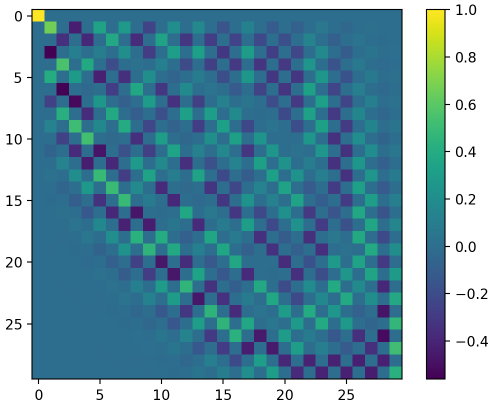
(b) The graphes is also plotted by using 50 bases, but the integration is improved by integrating over 3 intervals in which the potential function is continuous.

Figure 4

In order to improve the accuracy of the figure, the discontinuity was eliminated through integrating by 3 parts where the potential is continuous. The resultant functions in Figure(4b) look much better than those in Figure(3).



(a) The graph shows a number of probability function in a harmonic well



(b) The x axis from left to right correspond to the eigenstates from the bottom to top in the left figure. The y axis labels the basis function and the colour represents the coefficient of each basis

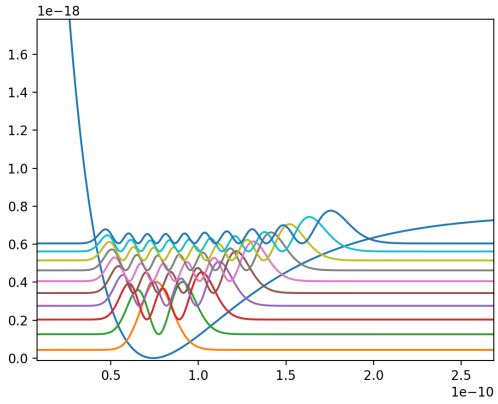
Figure 5: Probility Functions in Harmonic Potential and Its Eigenstates

However, it hard to tell the boundary between waves that are “accurate enough” and those of unsatisfactory accuracy. In order to find such boundary, the eigenstates are plotted with colour as shown in Figure(5b).

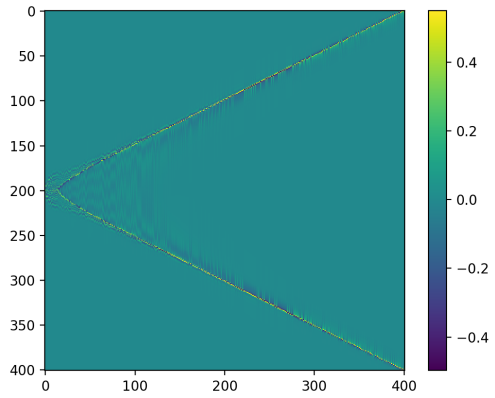
Figure(5b) shows that as the energy state increases, the corresponding eigenstate have more elements that are numerically significant. Eigenstates before the 13th one include all important elements; therefore, the 13th solution should be the boundary. Figure(5a) agreed with the conclusion because the 14th function indeed become flatter than the previous one.

4.2 Plane wave basis in 1D

Generally, using Fourier transform generates much better result than the previous method does. FFT method does not evaluate any integration but calculate discrete points, which greatly enhance the efficiency of the program and makes calculation of more bases possible.



(a) The graph shows a number of probability function in the Morse potenital well. Each wave function is composed of 200 bases, so the plot displays high accuracy.



(b) The colour plot diverges as the energy level increases. The waves at higher energy levels show that the particle is almost free.

Figure 6: Probility Functions in Morse Potential and Its Eigenstates

Figure(6) represents the probability functions of an hydrogen atom in the potential well produced by another hydrogen atom, which is called the Morse potential. By observing the eigenstate plot, one can assume that the probability functions are very accurate. The "boundary" between a free particle and a bounded particle is around the 110th energy states in Figure(6) because the basis functions between two divergent colour lines begin to vanish. The solution can be more accurate if more discrete points and bases are used, but there is no such need because the solutions are good enough.

4.3 Plane wave basis in 2D

Figure(7) demonstrates the probability function of an electron in its ground state in a harmonic potential well, and it was drawn by using 50 bases.

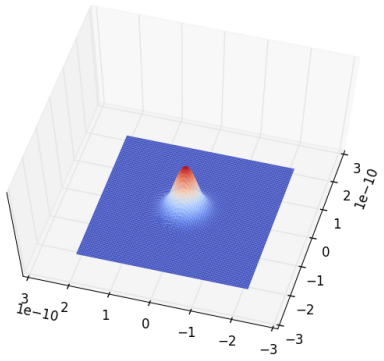
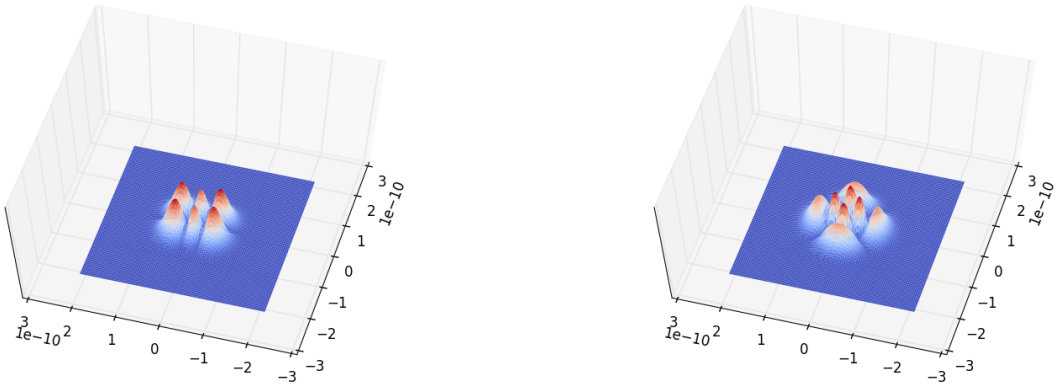


Figure 7: An Electron of Ground Energy State in a 2D Harmonic Well
This is the probability function in 2D. The electron most likely to be found into the red region and least likely found in the blue region.



(a) (b)

Figure 8: Probility Functions of Electrons in Higher Energy States in the Harmonic Potential

As the energy state rises, the frequency of the wave increases in different direction. In Figure(8), the frequency increases in the radial, angular, x and y direction. Figure(8a) shows that the wave increases frequency along x and $y$axis; Figure(8b) shows that the wave increases frequency along radial and angular directions. The phenomenon implies that for the same energy state, the wave function/probability function can have different shape, which makes sense because the (k^x, k^y) array contains some combinations of k^x and k^y that can generate the same energy.

The plot can also be improved by increasing the number of grid points and bases, but the time taken to solve the problem would also increase greatly.

5 Conclusion

The project is to make a program in python to solve the SE by expanding the solution on a set of basis. The basis of the solution of a infinite well and a set of plane wave basis were used. The result based on the plane wave basis turned to be more accurate by using discrete Fourier transform. The accuracy can be further improved by using more grid points and bases. The program has the ability to solve all potential wells in both 1D and 2D.

References

[1] Udala, A., Reedera, R., Velmrea, E. and Harrison, P. (2006).Comparison of methods for solving the Schrdinger equation for multiquantum well heterostructure applications. Proc. Estonian Acad. Sci. Eng., 2006, 12, 3-2, 246?261

[2] Eleuch, H. and Rostovtsev, Y. (2010). Analytical solution for 3D stationary Schrdinger equation: implementation of Huygens' principle for matter waves. Journal of Modern Optics, 57(19), pp.1877-1881.

[3] Marsiglio, F. (2009). The harmonic oscillator in quantum mechanics: A third way. American Journal of Physics, 77(3), pp.253-258.

[4] Marston, C. and Balint-Kurti, G. (1989). The Fourier grid Hamiltonian method for bound state eigenvalues and eigenfunctions. The Journal of Chemical Physics, 91(6), pp.3571-3576.

Core Method in 1D Fourier Transform

```
#Fast Fourier transform method#
def fourier(y):
return numpy.fft.fft(y)


#The function takes in discrete points and return frequencies of the base functions#
def frequency(N):
frequency=numpy.fft.fftfreq(2*N+1,a/(2*N+1))
frequency=ishift(frequency)
frequency=frequency*2*pi
return frequency


#The kinetic matrix is constructed as follow#
def KineticMatrix(N):
k=frequency(N)
Kinetic=(k*hb)**2/(2*M)
Matrix=diag(Kinetic)
return Matrix


#The potential matrix is constructed as follow#
def PotentialMatrix(N,V):
sample=linspace(0,a,4*N+1)
V=vectorize(V)
sample=V(sample)
#Because k ranges from -2*N to 2*N#
f=fourier(sample)
#f is not the result of integral#
dotted=1./(4.*N+1.)*f
#Dotted is the result of integral, by times 1/sqrt(a) and sqrt(a)/(4*N+1) on f#
Matrix=zeros((2*N+1,2*N+1),dtype="complex")
for i in range(2*N+1):
for j in range(2*N+1):
Matrix[i,j]=dotted[i-j]
return Matrix


#The Hamiltonian matrix is constructed by adding the kinetic matrix and
the potential matrix#
def HamilMatrix(N,V):
Matrix=KineticMatrix(N)*(1.+0.*1j)
```

```
Matrix+=PotentialMatrix(N,V)
return Matrix


#The Hamiltonian matrix is diagonalised here#
def EigenSolve(Matrix):
Values, Vectors=scipy.linalg.eig(Matrix)
return Values, Vectors


#Restore the solution by using inverse Fourier transform#
def EigenPlot(vector,n):
graph=1/sqrt(a)*(2*n+1)*numpy.fft.ifft(vector)
x=linspace(0,a,2*n+1)
return x,graph



Core Method in 2D Fourier Transform

#2D fast Fourier transform method#
def fourier2(mat):
mat=numpy.fft.fft2(mat)
return mat


#Truncate a (2N+1)*(2N+1) matrix by taking the elements within the circle of
radius N and appending each row one after another#
def TruncateFlatArray(matrix):
    length=len(matrix)
    N=(length-1)/2
    rebuild=[]
    number=[]
    for i in range(length):
     x=int((N**2-(N-i)**2)**0.5)
     rebuild+=((matrix[i,N-x:N+1+x]).tolist())
     number.append(2*x+1)
    rebuild=array(rebuild,dtype="complex")
    number=array(number)
    return rebuild, number


#Truncate a (4N+1)*(4N+1) matrix by setting the elements outside the circle of
radius N and centred at (2N+1)*(2N+1) to be 0. This function is one of the
process of building the potential matrix#
```

```
def TruncateMatrix(mat):
a=len(mat)
N=(a-1)/4
Matrix=zeros((a,a),dtype="complex")
arr,num=TruncateArray(mat[N:-N,N:-N])
for i,n in enumerate(num):
Matrix[N+i,(a-n)/2:(a+n)/2]=arr[i]
return Matrix,num


#The kinetic matrix is constructed as follow#
def KineticMatrix(N):
kx=frequency(N)
ky=kx
kx,ky=meshgrid(kx,ky)
UntruncMat=(kx**2+ky**2)*(hb**2/(2*M))
KineticMatrix=diag((TruncateFlatArray(UntruncMat)[0]))
return KineticMatrix


#The potential matrix is constructed as follow#
def PotentialMatrix(N,V):
x=y=linspace(-a/2,a/2,4*N+1)
x,y=meshgrid(x,y)
V=vectorize(V)
sample=V(x,y)
f=fourier2(sample)/(4*N+1)**2
f=ishift2(f)
f,rowsize=TruncateMatrix(f)
L=sum(rowsize)
Matrix=zeros((L,L),dtype="complex")
rowPotential=0
for row, size in enumerate(rowsize):
for column in range(size):
leftcoor=N+(size-1)/2-column
arr=TruncateFlatArray(f[2*N-row:4*N+1-row,leftcoor:leftcoor+2*N+1])[0]
Matrix[rowPotential,:]=arr
rowPotential+=1
return Matrix, rowsize


#The Hamiltonian matrix is constructed by adding the kinetic matrix and
the potential matrix#
def HamilMatrix(N,V):
Matrix=KineticMatrix(N)
```

```
mat,rowsize=PotentialMatrix(N,V)
Matrix+=mat
return Matrix,rowsize


#The Hamiltonian matrix is diagonalised here#
def EigenSolve(Matrix):
Values, Vectors=scipy.linalg.eig(Matrix)
return Values, Vectors


#This function takes in an eigenvector and transforms the vector into  a matrix.
The matrix is used as the parameter of the inverse FFT method#
def Detruncate(vector,rowsize):
N=(len(rowsize)-1)/2
matrix=zeros((2*N+1,2*N+1),"complex")
index=0
for i,row in enumerate(rowsize):
matrix[i,N-(row-1)/2:N+(row+1)/2]=vector[index:index+row]
index+=row
return matrix


#The solution is restored by the inverse 2D Fourier transform#
def EigenPlot(vector,rowsize):
N=(len(rowsize)-1)/2
matrix=freq2d(vector,rowsize)
graph=1/a*(2*N+1)**2*fourier2(shift2(matrix))
x=linspace(-a/2,a/2,2*N+1)
y=linspace(-a/2,a/2,2*N+1)
x,y=meshgrid(x,y)
Plot3D(x,y,abs(graph),5e14)
return graph
```