# Numerical solution of the time-independent 1-D Schrodinger equation.

Name: Kevin Grainger
Student Number: 20333346

INDEX:

1) Introduction
2) Theory
2.1) Analytical Solution
2.2)Numerical Methods
3) Required Equations
4) Source Code
4.1)Analytical equation
4.2)Trial Energy Wavefunction in Well
4.3)Eigenstates
4.4)Normalisation
4.5)Uncertainty Relation
4.6)Harmonic Potential
4.7)Eigenstate Energies
5)Results

## 1)Introduction

This lab aims to investigate the numerical methods involved in solving and representing the 1D wave function. This is achieved using an iterative function to alter the wavefunction in order to normalize it. The Numerov method and finite difference scheme are used to perform any calculus involved (detailed in Section 3). The wave function was defined and normalized successfully, and from here it was compared to the numerical solution. The eigenfunctions were calculated and normalized. This allowed us to verify Heisenburg's Principal. Finally, the wave function in a harmonic potential was calculated and graphed.

## 2)Theory

### Analytical Method:

$$E\psi(x) = -\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x),$$

$$\frac{d^2\psi(\tilde{x})}{d\tilde{x}^2} + \gamma^2\left(\epsilon - \nu(\tilde{x})\right)\psi(\tilde{x}) = 0,$$

Analytical calculations are all done via the Schrodinger Equation. This can be solved for the initial potential of v=-1 . Giving the
standard solution:
$\psi(x) = Ae^{ikx} + Be^{-ikx}$

### Numerical Method:

The main numerical method involved was the Numerov method.
The Numerov algorithm is used instead of the Runge-Kutta as it takes advantage of the fact there is no first-order derivative in the equation and that the function $\varphi$ is linear. The algorithm is constructed via combining the Taylor expansion of $\varphi$(x+l) and $\varphi$(x-l). This leaves us with the implemented expression:

$$\psi_{n+1} = \frac{2\left(1 - \frac{5}{12}l^2k_n^2\right)\psi_n - \left(1 + \frac{1}{12}l^2k_{n-1}^2\right)\psi_{n-1}}{1 + \frac{1}{12}l^2k_{n+1}^2}$$

This can be combined with numerical discretization to solve the ODE with the following:

$$d\psi/dx \approx (\psi_{n+1} - \psi_n)/l.$$

Using the approximation below we can calculate the first derivative. All that is needed is an original point $\varphi_n$ and the proceeding point $\varphi_{n+1}$ . This process can be iterated over all particles.

## 3)Equations

$$E\psi(x) = -\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x), \qquad [1]$$

$$d\psi/dx \approx (\psi_{n+1} - \psi_n)/l. \qquad [2]$$

$$\psi_{n+1} = \frac{2\left(1 - \frac{5}{12}l^2k_n^2\right)\psi_n - \left(1 + \frac{1}{12}l^2k_{n-1}^2\right)\psi_{n-1}}{1 + \frac{1}{12}l^2k_{n+1}^2} \qquad [3]$$

## 1)Code & Method

```
N=1000
psi=np.zeros(N)
#def function(parameter1,parameter2):
#result=(parameter1+parameter2)**2
#return result


# Solve the non-dimensional Schr¨odinger equation (2) analytically for the
#infinite square well where v(˜x) = −1 for 0 < x <˜ 1 and find an expressio
#for the non-dimensional energy eigenvalues in terms of γ2
# Also, find
#the normalised wave functions (in non-dimensional units).
N=1000
l=1/(N-1)

#psi=np.array([N])
Gamsq=1000

x=np.linspace(0,1,N)
#v=np.zeros(N)-1
v=8*(x-0.5)**2-1
```

To begin we define N=1000 particles. To define the potential I started with an array of zeros of length N, then the array minus one gives us the required potential.

```
def numerov(E, v):

    k = Gamsq*(E - v)

    psi = np.zeros(N)
    psi[0]=0
    psi[1]=1e-4

    for i in range(0,N-2):

        a = 2*( 1 - ((5/12)*(l**2)*k[i+1]))*psi[i+1]
        b = ( 1+(0.08333333)*l**2*k[i])*psi[i]
        c = (1+(0.0833333)*l**2*k[i+2])

        psi[i+2] = (a-b)/c

    return psi
print(numerov(e,v))
#i changed potiential to v so change back if it doesn't work
```

Here we have my Numerov algorithm. We know the value of psi[1] and psi[2] means we can implement this method. Using equation [3] we can fill in our values. This function returns a value for each N.

```
def bisection(e):
    de=1e-2

    while (abs(de) > 1e-8):
        psi1= numerov(e,v)
        e = e + de
        psi2 = numerov(e, v)

        if(psi1[N-1]*psi2[N-1] < 0):
            de=-de/2

    return psi1, e

trial = -0.99
```

Next using the ‘shooting method’ or bisection method we can attempt to normalize our wavefunction. This means making sure it zeros out at the appropriate length. We achieve this by setting the if condition to ensure psi1[N-1]*psi2[N-1]>0

Thus ending the iteration process at zero.
We then add define our energy and wave arrays to store this data.

```
l=(1.0/(N-1))
def normal(psi):
    N = simps(psi**2,dx=l)
    return psi/np.sqrt(N)

print(simps(normal(psi)**2,dx=l))     #verify that it is normalized

x_n=np.linspace(0,1,1000)

x_val1=simps((x_n*normal(psi)**2),dx=l)**2
x_val2=simps((x_n**2*normal(psi)**2),dx=l)

delta_x=np.sqrt(x_val2-x_val1)

print(delta_x)



for i in range(20):
    psi, E = bisection(trial)
    print(f'Found energy: {E}')
    E_array.append(E)
    wave_array.append(psi)
    plt.plot(x,psi,label='eigenvalue {i}')
    plt.title(' Wavefunctions in v=-1')

    trial = E + 0.01

#plt.show()
```

Here we have *def Normal(psi)* to normalize the wavefunction. Using the simps function we can perform the integration easily. Thus getting delta x Using a for loop we start adding values to our arrays.

Final steps, finding the derivative of psi via simps function, and filling the uncertainty arrays. The last calculations are made and the plots are coded.

```
def deriv_psi(psi,l):
    psi_2=np.zeros(1000)
    for i in range(1,len(psi_2)-1):
        psi_2[i] = (psi[i-1]-2*psi[i]+psi[i+1])/l**2
    return psi_2



delta_p_array=[]
delta_x_array=[]

for i in range(len(wave_array)):
    psi=wave_array[i]
    second_deriv = deriv_psi(normal(psi), l)
    delta_p=np.sqrt(simps(-1*normal(psi)*second_deriv,dx=l))

    x_val1=simps((x_n*normal(psi)**2),dx=l)**2
    x_val2=simps((x_n**2*normal(psi)**2),dx=l)

    delta_x=np.sqrt(x_val2-x_val1)

    delta_p_array.append(delta_p)
    delta_x_array.append(delta_x)

product=[]
```

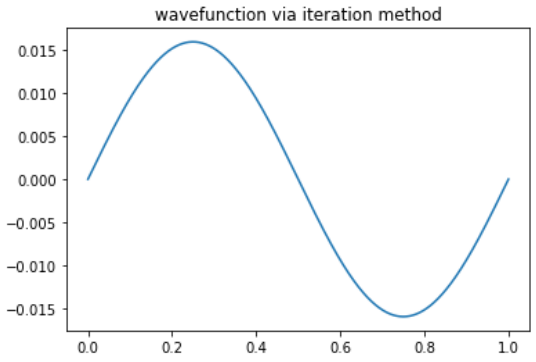
```
for i in range(len(delta_p_array)):
    product.append(delta_p_array[i]*delta_x_array[i])

#plt.plot(delta_p_array,label='delta_p')
#plt.plot(delta_x_array, label='delta_x')
#plt.plot(product,label='(delta_p)*(delta_x)')
#plt.title('Uncertainty (delta x, delta p & (delta_p * delta_x)')

#plt.show()

print(delta_p_array)
print(delta_x_array)
plt.show()
#enrgy difffernce between the different eigenstates
E_diff=[]
for i in range(len(E_array)-1):
    k=E_array[i+1]-E_array[i]
    E_diff.append(k)
print(E_diff)
plt.plot(E_diff)
plt.title('Energy Change Between Eigenstates')
plt.xscale('log')
plt.yscale('log')
plt.show()
```

## 4) Results



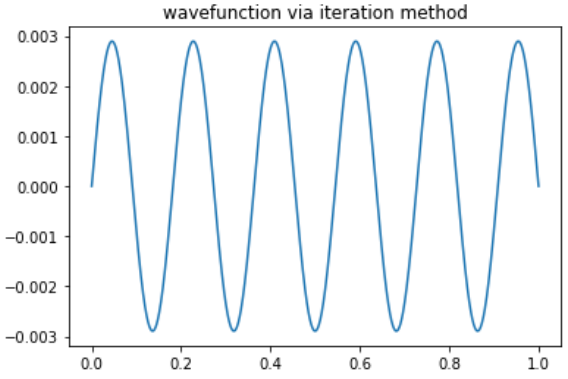
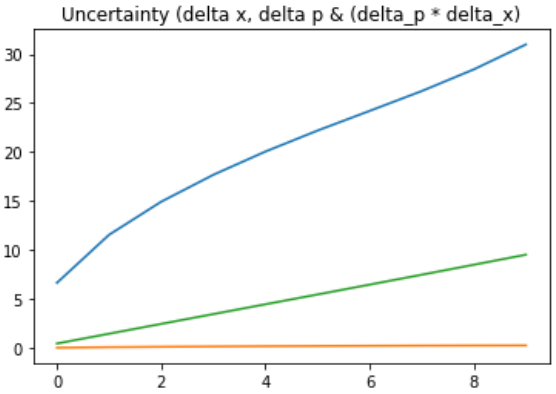
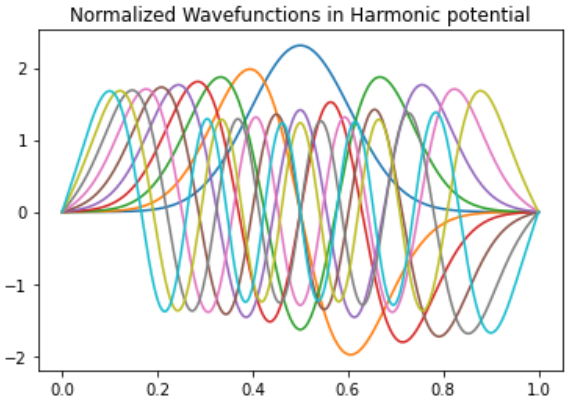



Figure [1]
The normalized wavefunction.

Figure [2]
An example of an eigenfunction of our system.




Respectively:

*Figure [3]*

The generalized uncertainty over the period.

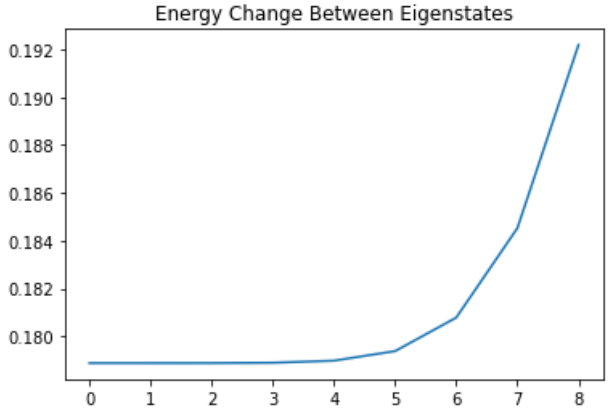
-Uncertainty of momentum (Δp) - Blue

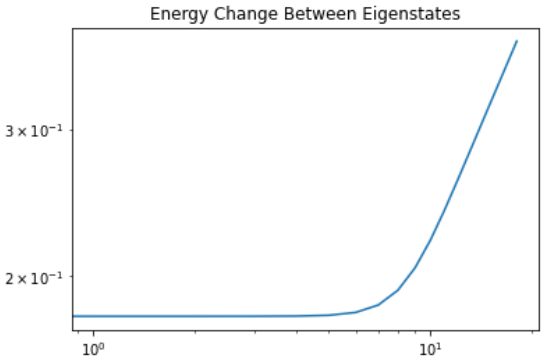
-Uncertainty in position (Δx) - Orange

-Uncertainty (Δpx)- Green

*Figure [4]*

Here we have the first state wavefunction and its eigenstates in a harmonic potential. The ‘ground state’ takes the form of a central peak (blue). The eigenfunctions add a peak and a trough towards the center. Which differs from our previous v(x)= -1.




Respectively:

Figure [5] Energy change between the different eigenstates. We see an exponential increase. Harmonic potential only exists between x, 0 & 1. Due to our first simplification of X=x/l.Thus after this, we see an infinite square well forming.

Figure [6] Log plot of the energy change, as in Figure[5]

## References:

[1] TCD, school of physics, lab manual ‘Numerical solution of the time-independent 1-D Schr¨odinger equation Paul Eastham, Mark Mitchison, Matthias E. M¨obius

[2] Quantum Physics for Dummies-Steven Holzner