Using Numerical Methods to Explore Quantum Mechanics

Marc John Bordier Dam

May 4, 2017

Abstract

This report uses numerics to study a Gaussian wave package in an infinite square well and in a finite square well inside an infinite square well. We find that the wave package loses its shape over time although it returns to its initial state eventually and shows periodic behavior.

Contents

1 Introduction

In this report we will study two different potentials: The infinite square well and the finite square well inside an infinite square well. The infinite square well corresponds to a particle trapped in a box and is an idealized model of an electron trapped in a long organic molecule e.g. β-carotene. The finite square well is interesting to study since it is possible to find a bound particle outside the well. This differs from the classical behavior where a bound particle can only be found inside the well. The finite square well potential is shown in figure 2. For a figure of the infinite square well just imagine figure 2 with inifinitely tall walls.

2 Schrödinger's Equation and its Solutions

The most fundamental equation in quantum mechanics is the Schrödinger equation. It governs which functions are worthy of study, that is, which functions are allowed wave functions in a given potential. The time-dependent Schrödinger equation in one dimension is

$$i\hbar\frac{\partial\Psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} + V\Psi. \tag{1}$$

To make solving the time-dependent Schrödinger equation easier we use separation of variables to obtain the time-independent Schrödinger equation

$$-\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V\psi = E\psi, \tag{2}$$

where $\Psi = \psi e^{-\frac{iEt}{\hbar}}$.

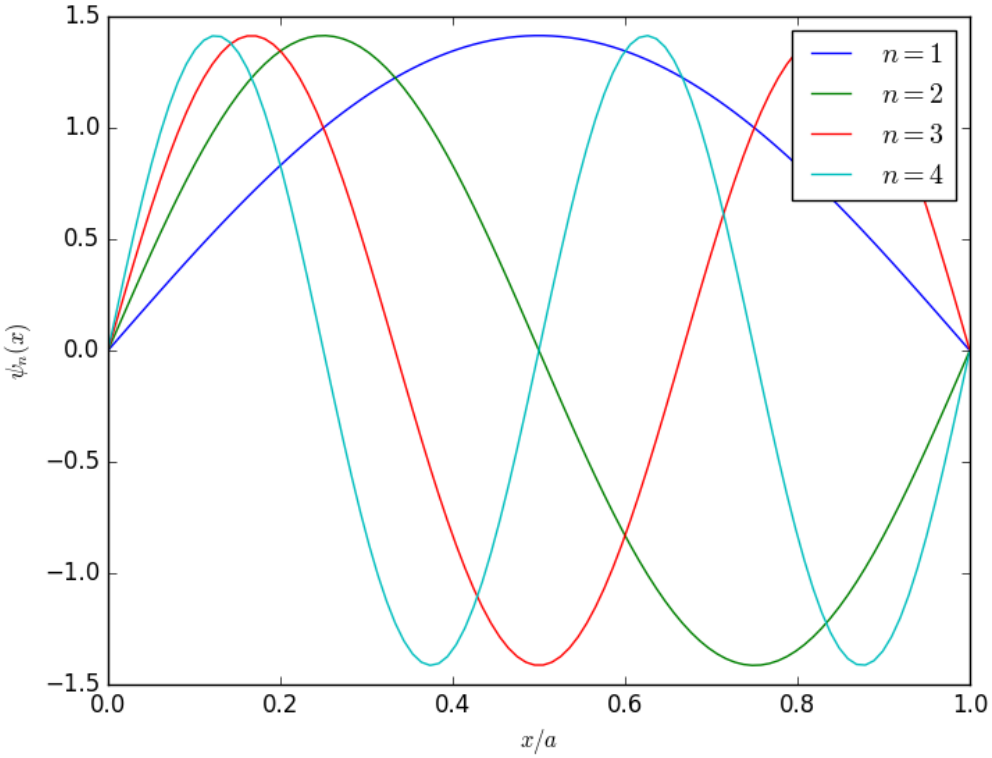

Figure 1: Wave Functions in the Infinite Square well

The wave functions which are solutions to the time-independent Schrödinger euqation represent stationary states. These stationary states have the nice property that any function in Hilbert space can be written as a linear combination of them. Thus it is in many cases sufficient to work with the stationary states and linear combinations of them. Below we will present the stationary states along with their allowed energies for the infinite and finite square well.

2.1 The Infinite Square Well

In the infinite square well the potential is given by

$$V = \begin{cases} 0, & 0 \leq x \leq a \\ \infty, & \text{otherwise} \end{cases}. \tag{3}$$

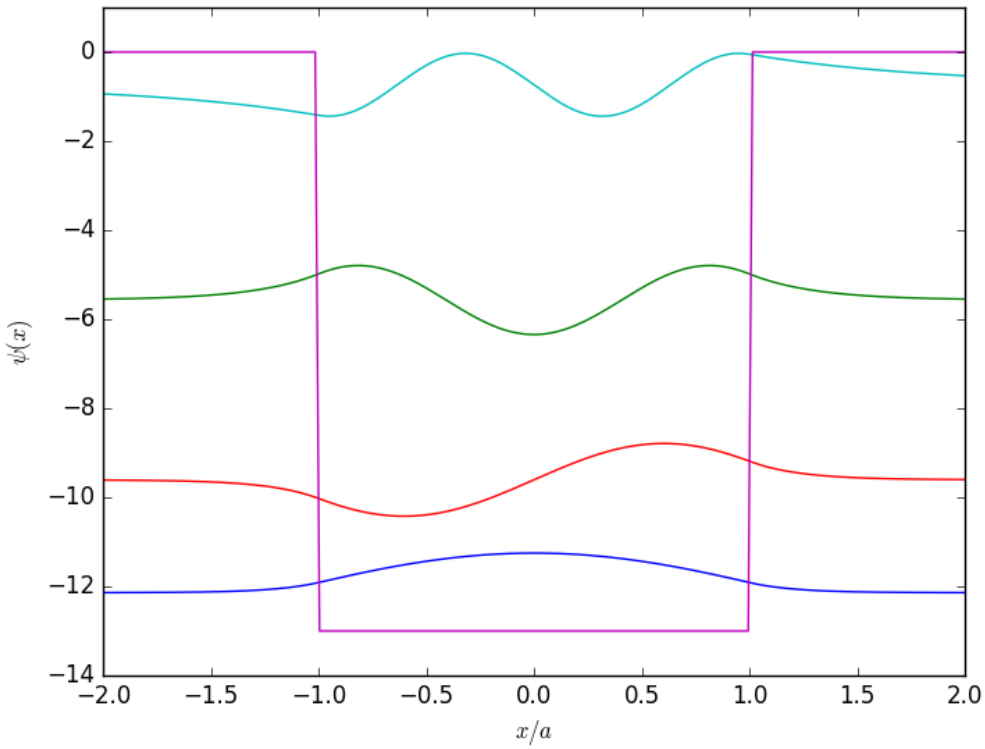

Figure 2: Wave Functions in the Finite Square well

Outside the well the wave function is 0 and inside the well it is given by

$$\psi_n(x) = \sqrt{\frac{2}{a}} \sin\left(\frac{n\pi}{a}x\right), \tag{4}$$

where n is any positive integer. In the infinite square well the energy spectrum is discrete with the nth stationary state having the energy

$$E_n = \frac{n^2\pi^2\hbar^2}{2ma^2}. \tag{5}$$

Some of the wave functions are shown in figure 1.

2.2 The Finite Square Well

In the finite square well the potential is given by

$$V = \begin{cases} -V_0, & -a \leq x \leq a \\ 0, & \text{otherwise} \end{cases}, \tag{6}$$

where V_0 is a positive number.

In the finite square well the wave function can be found both inside and outside the well. We also have both bound ($E < 0$) and scattering states ($E > 0$) in the finite square well.

The bound states come in two flavors: Even and odd. The even wave functions are given by

$$\psi_{even}(x) = \begin{cases} Ae^{-\kappa x}, & a < x \\ A\frac{e^{-\kappa a}}{\cos la}\cos lx, & -a \leq x \leq a \\ Ae^{\kappa x}, & x < -a \end{cases}, \tag{7}$$

where A is a normalization constant, $\kappa = \frac{\sqrt{-2mE}}{\hbar}$ and $l = \frac{\sqrt{2m(E+V_0)}}{\hbar}$.

Similarly the odd wave functions are given by

$$\psi_{odd}(x) = \begin{cases} Ae^{-\kappa x}, & a < x \\ A\frac{e^{-\kappa a}}{\sin la}\sin lx, & -a \leq x \leq a \\ -Ae^{\kappa x}, & x < -a \end{cases}, \tag{8}$$

where again A is a normalization constant, $\kappa = \frac{\sqrt{-2mE}}{\hbar}$ and $l = \frac{\sqrt{2m(E+V_0)}}{\hbar}$.

The energy spectrum of the bound states is discrete but can not be determined analytically. Defining $z = la$ and $z_0 = \frac{a}{\hbar}\sqrt{2mV_0}$ the allowed energies for the even wave functions can be found by solving the equation

$$\tan z = \sqrt{\left(\frac{z_0}{z}\right)^2 - 1}. \tag{9}$$

Likewise the allowed energies for the odd wave functions can be found by solving the equation

$$-\cot z = \sqrt{\left(\frac{z_0}{z}\right)^2 - 1}. \tag{10}$$

Some of the wave functions for the bound states are shown in figure 2.

The wave functions for the scattering states are given by

$$\psi(x) = \begin{cases} Fe^{ikx} & a < x \\ C\sin lx + D\cos lx & -a \leq x \leq a \\ Ae^{ikx} + Be^{-ikx} & x < -a \end{cases}, \tag{11}$$

where A is a normalization constant and

$$\begin{aligned} k &= \frac{\sqrt{2mE}}{\hbar}, \\ l &= \frac{\sqrt{2m(E+V_0)}}{\hbar}, \\ B &= i\frac{\sin 2la}{2kl}\left(l^2 - k^2\right)F, \\ C &= \left(\sin la + i\frac{k}{l}\cos la\right)e^{ika}F, \\ D &= \left(\cos la - i\frac{k}{l}\sin la\right)e^{ika}F, \\ F &= \frac{e^{-2ika}}{\cos 2la - i\frac{k^2+l^2}{2kl}\sin 2la}A. \end{aligned}$$

For the scattering states we observe reflection and transmission through the well wall.

The energy spectrum of the scattering states is continuous and thus any positive energy is allowed.

3 Quantum Mechanics in Reduced Units

When doing simulations we want to work with the physical quantities in reduced units. We do this for two reasons: We don't have to keep track of the units on everything and quantities in reduced units tend to be of a reasonable order of magnitude.

In our simulations we have three constants: The particle mass m_0, the well width a and Planck's constant $\hbar$. In our system of reduced units these will all

Table 1: Reduced Units (subscript r denotes the quantity in real units)

Quantity (Q.)	Q. in R. U.	Characteristic Q.	Electron in π-bond
Length, x	$\frac{x_r}{a}$	a	$1.34\,\text{Å}$
Mass, m	$\frac{m_r}{m_0}$	m_0	$m_e = 9.109 \times 10^{-31}\,\text{kg}$
Time, t	$\frac{t_r}{t_0}$	$t_0 = \frac{m_0 a^2}{\hbar}$	$1.55 \times 10^{-16}\,\text{s}$
Energy, E	$\frac{E_r}{E_0}$	$E_0 = \frac{\hbar^2}{m_0 a^2}$	$6.80 \times 10^{-19}\,\text{J}$
Velocity, v	$\frac{v_r}{v_0}$	$v_0 = \frac{\hbar}{m_0 a}$	$8.64 \times 10^{5}\,\text{m s}^{-1}$
Momentum, p	$\frac{p_r}{p_0}$	$p_0 = \frac{\hbar}{a}$	$7.87 \times 10^{-25}\,\text{m kg s}^{-1}$

be equal to 1. With these three constants we can derive the reduced units for the remaining relevant quantities in our simulation. The relevant quantities are shown in table 1, where the real values for an electron in a π-bond in an ethylene molecule is also shown.

3.1 Schrödinger's Equation in Reduced Units

Since m and $\hbar$ both are equal to 1 in our system of reduced units the time-dependent Schrödinger equation takes the form

$$i\frac{\partial \Psi}{\partial t} = -\frac{1}{2}\frac{\partial^2 \Psi}{\partial x^2} + V\Psi, \tag{12}$$

where t, x and V are all in reduced units. Likewise the time-indepedent Schrödinger equation takes the form

$$-\frac{1}{2}\frac{\partial^2 \psi}{\partial x^2} + V\psi = E\psi, \tag{13}$$

where again x, V and E are all in reduced units and $\Psi = \psi e^{iEt}$. The stationary wave functions in the infinite square well along with the allowed energies take the form

$$\psi_n(x) = \sqrt{2}\sin(n\pi x), \quad E_n = \frac{n^2\pi^2}{2}. \tag{14}$$

The wave functions representing bound stationary states take the form

$$\psi_{even}(x) = \begin{cases} Ae^{-\sqrt{-2E}x}, & 1 < x \\ A\frac{e^{-\sqrt{-2E}}}{\cos\sqrt{2(E+V_0)}}\cos\left(\sqrt{2(E+V_0)}x\right), & -1 \leq x \leq 1 \\ Ae^{\sqrt{-2E}x}, & x < -1 \end{cases} \tag{15}$$

and

$$\psi_{odd}(x) = \begin{cases} Ae^{-\sqrt{-2E}x}, & 1 < x \\ A\frac{e^{-\sqrt{-2E}}}{\sin\sqrt{2(E+V_0)}}\sin\left(\sqrt{2(E+V_0)}x\right), & -1 \leq x \leq 1\,. \\ -Ae^{\sqrt{-2E}x}, & x < -1 \end{cases} \tag{16}$$

These equations along with equation (9) and equation (10) in reduced units is what will form the basis of our numerical investigations of the infinite and finite square well.

An idealized model of the π-bond in an ethylene molecule is an infinite square well where the width of the well is the length of a bond. The particle confined in the well then corresponds to an electron confined in a π-bond. The bond length between the two carbon atoms in ethylene is about 1.34 Å and the mass of the electron is about $m_e = 9.109 \times 10^{-31}$ kg. Using these numbers we can make a conversion table (table 1) which allows us to convert from reduced units in our simulation to the real world example of an electron in a π-bond in an ethylene molecule.

4 Numerics

In this section we will introduce the software as well as some of the numerical methods we have used in the process of simulation the potentials and wave packages.

4.1 The Tools

In this project we have used Python to numerically simulate a guassian wave package in an infinite square well and in a finite square well inside an infinite

square well.

Since Python does not have native support for working with matrices we have used the NumPy package. NumPy includes tools for working with matrices and multidimensional arrays as well as tools for doing numerical integration and differentiation.

The function `numpy.trapz` does numerical integration with the trapezoidal method which approximates the area under a curve using trapezoids. The function `numpy.gradient` does numerical differentiation using a finite difference method.

For plotting we have used the Matplotlib package. We have also used the Matplotlib package for producing the animated videos of the plots.

4.2 Root Finding

When working with the finite square well we need to do numerical root finding to determine the allowed energies. We use Newton's method to determine the solutions to equation (9) and equation (10).

Newton's method works by repeatedly estimating the root using a linear approximation of the function we wish to find the root of. As always we use a first order Taylor expansion to find a linear approximation. Starting from an initial guess of x_0 this gives us the equation

$$0 = f(x_0) + f'(x_0)(x - x_0) \tag{17}$$

If we solve for x we get that

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{18}$$

Since we arrived at this "root" by using an approximation we would expect that this "root" is also only an an approximation to the real root which we are interested in. Since linear approximations of differentiable functions are more accurate the closer we are to the point around which we approximate, we can get a better estimate of our root by repeating the process. Thus we rewrite equation (18) as a recurrence relation

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{19}$$

For the implementation used in our script see the appendix A.1.

4.3 Code Structure

The code for each potential is divided into a number of modules. The lowest level module `psi_n` (named `psi_n.py`, `FSW_psi_n_bound.py` and `WiW_psi_n.py` respectively) contains the wave functions of the time-independent stationary states as well as the energies corresponding to the stationary states. It takes the wave function index n as well as a position array as arguments.

The `psi_n` module is then used in the `Psi_n` module (named `Psi_n.py`, `FSW_Psi_n_bound.py` and `WiW_Psi_n.py` respectively) which contains the time-depedent wave functions. This module takes a time value as argument in addition to the arguments of the `psi` module. The `Psi_n` module is then used to construct the general wave function. This is done in the `Psi` module (named `Psi.py`, `FSW_Psi_bound.py` and `WiW_Psi.py` respectively) which takes a list of coefficients c_n as an argument in addition to the arguments of the `Psi_n` module.

To get the c_n's for an arbitrary wave function we use Fourier's trick to get the projection of the wave function on the stationary states. This is done in the module `getcn` (named `getcn.py`, `FSW_getcn_bound.py` and `WiW_getcn.py` respectively) which determines the c_n's by numerical integration. The `getcn` module has two different options. It can either determine a fixed number of c_n's or determine enough c_n's to make the sum of the norm squares within an ε of 1. In the case of the bound states in the finite square well it is not always possible to ensure that the c_n's have the required norm since there is a finite number of bound states in the finite square well.

We gather all these in the main script (named `wavePackageSimultaneousPlot.py` and `WiW_wavePackageSimultaneousPlot.py` respectively) where we simulate the wave package in the potential. The overall structure of the script is to first load in the modules `Psi` and `getcn`. Then we define and normalize the wave package. After this we use the `getcn` module to get a list of c_n's which we then use to calculate the linear combination of stationary states using the `Psi` module. With the setup done we then calculates $\langle x \rangle$, $\langle x^2 \rangle$, $\langle p \rangle$ and $\langle p^2 \rangle$ using `numpy.trapz` and

`numpy.gradient`. These values can be used to calculate σ_x and σ_p which we use to check that the uncertainty principle holds. Finally we calculate the momentum distribution by projecting the wave function Ψ onto the eigenfunctions of the momentum operator. The remainder of the script is used to animate the wave package along with its momentum spectrum.

5 Numerical Results

In this section we present the numerical results obtained from simulating a wave package in the infinite square well and in the finite square well inside an infinite square well.

The infinite square well extends from 0 to 1 in reduced units. The finite square well inside the infinite square well is made such that the finite square well extends from -1 to 1 in reduced units and the surrounding infinite square well extends from -3 to 3 in reduced units.

We have simulated a Gaussian wave package given by the equation

$$\Psi(x,0) = Ae^{-\frac{1}{2}\frac{(x-x_0)^2}{\sigma_0^2}}e^{-ik_0x}, \tag{20}$$

where A is a normalization constant. In the infinite square well we set $x_0 = 0.5$, $\sigma_0 = 0.1$ and $k_0 = 100$. The simulated wave package is a linear combination of the first 48 staionary states. In the finite square well inside an infinite square well we set $x_0 = 2.5$, $\sigma_0 = 0.1$ and $k_0 = 25$. This wave package is a linear combination of the first 110 stationary states. The time evolution as well as the momentum spectrum of these wave packages are shown in the accompanying videos.

Two snapshots of the wave package in the infinite square well at different times are shown in figure 3 and figure 4. From these figures we see that the wave package has a well defined shape, position and momentum early on. At time zero $\sigma_x\sigma_p$ for the wave package is close to the lower limit of the uncertainty principle. At later times the wave package has a less well defined shape, position and momentum. This is also consistent with $\sigma_x\sigma_p$ being large. The wave package in the finite square well inside the infinite square well shows similar behavior.

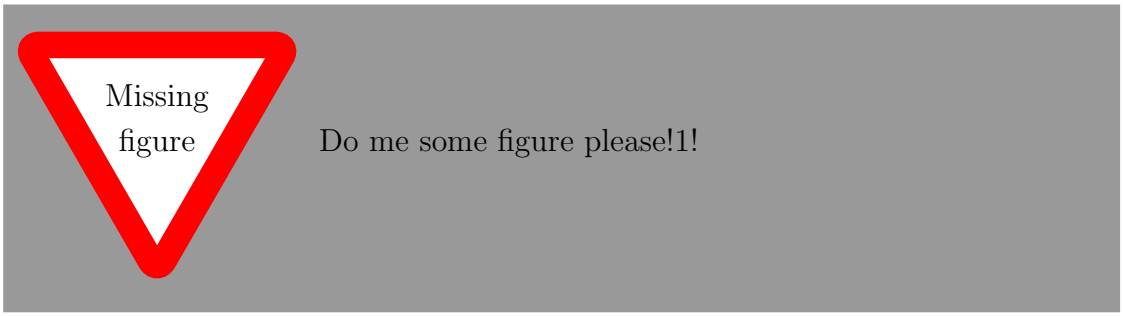

Figure 3: The wave package in the infinite square well along with its momentum spectrum at t = ...

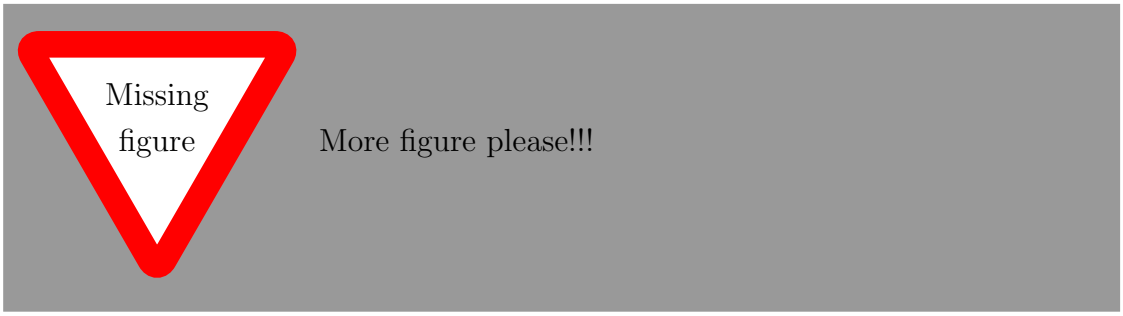

Figure 4: The wave package in the infinite square well along with its momentum spectrum at t = ...

Our numerical simulations highlight an interesting feature of the infinite square well: If we wait long enough the wave package will eventually return to its initial shape. This happens in our simulations because we are working with the periodic functions sine and cosine. In the real world we would not be able to have an isolated wave package in an infinite square well. Perturbations and interactions from outside the well would break the periodicity and it would be very unlikely for the wave package to assume its initial shape at later times. We would also expect that a real world wave package would not return to a state of minimal uncertainty.

It is worth noting that the wave package in the finite square well inside the infinite square well loses its shape faster than the wave package in the infinite square well. This is consistent with the fact that the wave package in the finite square well inside an infinite square well has a higher energy than the wave package in the infinite square well. The finite square well appearently also does some work in spreading out the wave package and we can see from the video of the wave package in the finite square well inside the infinite square well that the standard deviation of the position of the wave package increases as the wave package passes over the finite square well.

The energy spectra shown in figure 5 and figure 6 show that our wave packages are in a superposition of stationary states which are eigenfunctions of the Hamiltonian. Thus if we were to measure the energy of one of our wave package we would only be able to measure the energies shown in the figures. It is important to note that we can only measure the discrete energies plotted and not any energy in between. When we measure the energy of one of our wave package $|c_n|^2$ represents the probability of measuring a certain energy. Hence if we were to measure the energy of one of our wave package in the finite square well inside the infinite square well we would be more likely to observe energies around 10 000 than energies around 40 000 and we would never measure energies above 60 000.

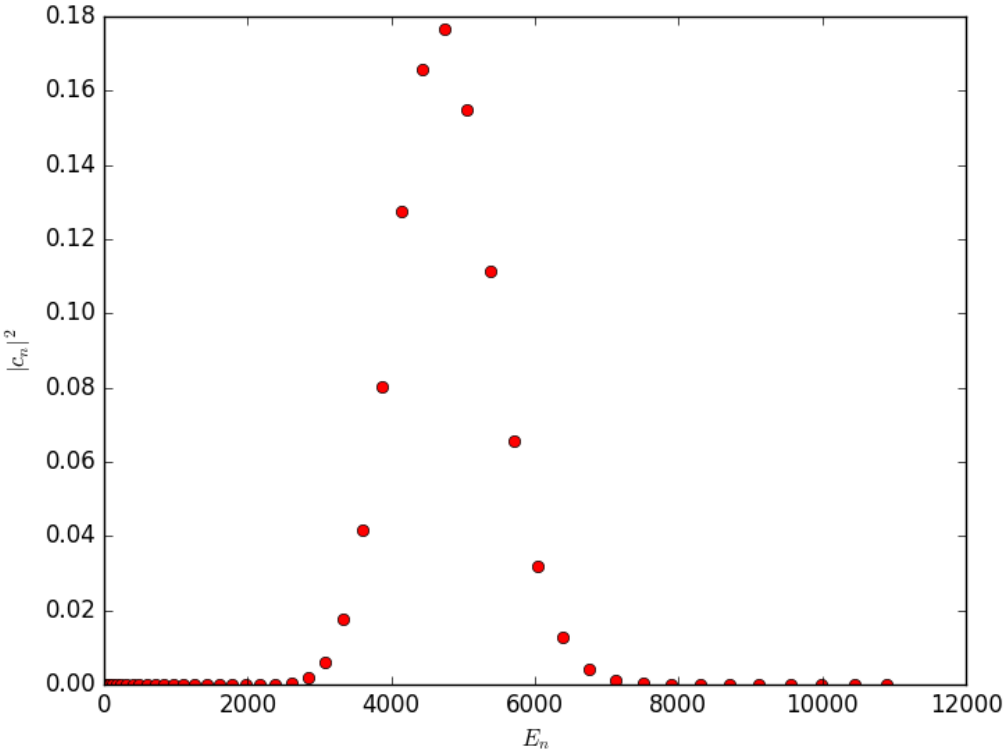

Figure 5: Energy spectrum of the wave package in the infinite square well

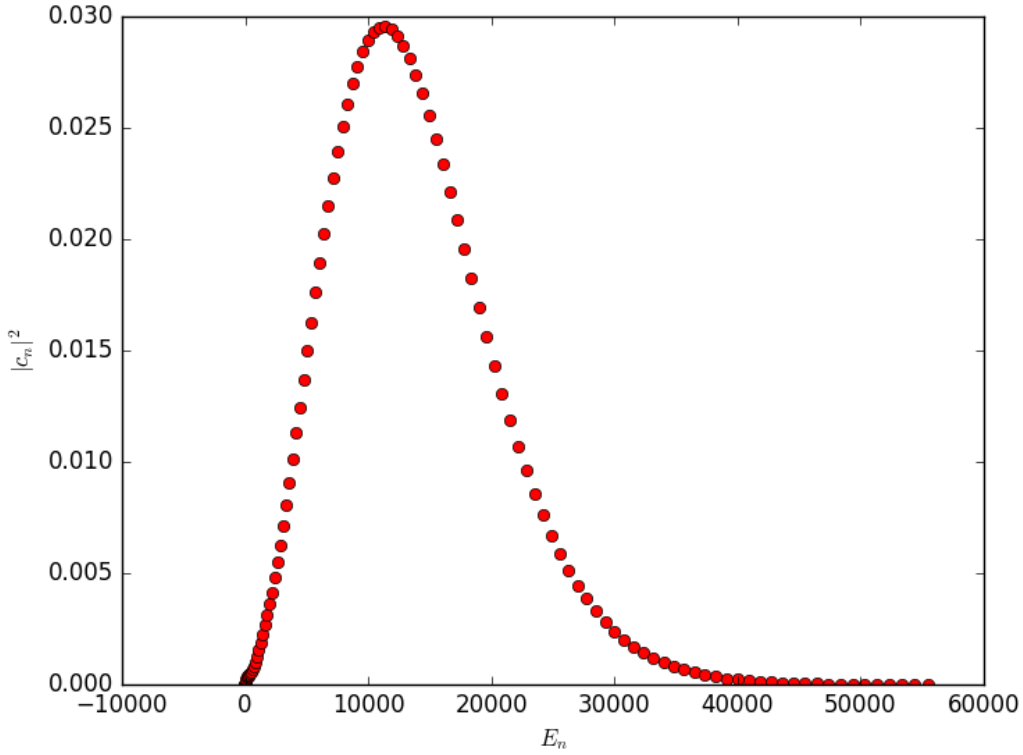

Figure 6: Energy spectrum of the wave package in the finite square well

6 Conclusion

We have in this report seen how many of the peculiarities of quantum mechanics can be visualized through the use of numerics in the case of the relatively simple potentials of the infinite and finite square well. We have also seen how the numerics highlights some of the stranger features of the potentials which result from the potentials being idealizations of real world potentials.

A Appendix: Code Snippets

This appendix includes some of the interesting excerpts from our code.

A.1 Newton's Method

We have implemented Newton's method in our script used for finding the allowed energy levels of the stationary states in the finite square well as follows:

```
niter = 10**4
guesses = [(np.pi/2 - 0.1, np.pi), (np.pi - 0.1, 2)]
newtonRoots = []

for i, hfun in enumerate(hfuns):
  print(str(hfun))

  guess = guesses[i][0]
  newtonRoots.append([])
  dhdz = dhdzfuns[i]

  while guess < z0:
    zi = guess

    for j in range(niter):
      zi = zi - hfun(zi)/dhdz(zi)

    newtonRoots[i].append(zi)
    guess = zi + guesses[i][1]

    print(zi)
```

Here `hfuns` is a previously defined list of functions which correspond to equations (9) and (10) with the RHS subtracted from the LHS. The derivatives `dhdzfuns` are calculated symbolically since the numerical derivative was erroneous near the singularities of $\tan z$ and $\cot z$. As initial guesses we use values slightly below the

singularities of $\tan z$ and $\cot z$ respectively.

A.2 The Main Script

This section will give a bit more detail on the main script in which we simulate the wave package.

```
x = np.linspace(0, 1, 2**12)
t = np.linspace(0, 4/(np.pi), 2**13)
```

Through experimenting with the code we have found that it is important to have a good resolution in the position array. Likewise we need a good resolution in the time array since the wave functions with higher energies have a small period. We simulate up until $t = \frac{4}{\pi}$ since this is the largest period of any of the wave functions and all the other wave functions have periods which divides this.

```
# Wave package constants
x0 = 0.5;
sigma = 0.1;
k0 = 100;

# Normalize the custom wave function and get the cn's
fun = np.exp(-1/2 * (x-x0)**2/sigma**2) * np.exp(-1j*k0*x)

A = 1/np.sqrt(np.trapz(np.abs(fun)**2, x = x))

fun = A * fun

cn = getcn(x, fun)

# Check normalization

f = np.abs(Psi(cn, x, t[0]))**2
integral = np.trapz(f, x = x)
```

Before we compute the c_n's we normalize the wave package using `numpy.trapz` to determine the normalization constant. The default option for the `getcn` module is to calculate c_n's until the square norm of the c_n's is below 10^{-12}. Because the c_n's have a square norm close to 1 the integral of the wave function is also close to 1. In the case of the infinite square well `getcn` returns 48 c_n's and the integral has a value of 0.99999999999907341.

```
41 # Calculate the function for all times
   for i in range(0, len(t)):
     f.append(Psi(cn, x, t[i]))
```

We calculate the wave function for all times to avoid extra function calls whenever we need the wave function. This saves us calculations when we calculate expected values, calculate the momentum spectrum and animate the wave function.

```
45 # Compute <x> and <x^2>

   Ex = np.zeros(t.shape)
   Ex2 = np.zeros(t.shape)

50 for i in range(0, len(t)):
     Ex[i] = np.trapz(x*(np.abs(f[i])**2), x = x)
     Ex2[i] = np.trapz(x**2*np.abs(f[i])**2, x = x)

   STDx = np.sqrt(Ex2 - Ex**2)
55
   # Compute <p> and <p^2>

   Ep = np.zeros(t.shape, dtype = "complex128")
   Ep2 = np.zeros(t.shape, dtype = "complex128")
60
   dx = x[1] - x[0]
   for i in range(0, len(t)):
     Ep[i] = np.trapz(np.conjugate(f[i])*(-1j)*np.gradient(f[i],
```

```
      dx), x = x)
  Ep2[i] = -np.trapz(np.conjugate(f[i])*np.gradient(np.gradient
      (f[i], dx), dx), x = x)

STDp = np.sqrt(Ep2 - Ep**2)
```

We calculate the expected values using the standard integrals. To determine $\frac{\partial \Psi}{\partial x}$ we use numerical differentiation with `numpy.gradient`.

```
# Calculate the momentum spectrum
p = np.linspace(-150, 150, 2**9)
pdist = []

for i in range(0, len(t)):
  if i % 2**8 == 0:
    print("%.2f" %(i/len(t)))

  tmp = np.zeros(p.shape, dtype=np.complex128)
  for j in range(0, len(p)):
    tmp[j] = np.trapz(np.conj(1/np.sqrt(2*np.pi)*np.exp(1j*p[j
        ]*x))*f[i], x = x)

  pdist.append(tmp)
```

To calculate the momentum distribution we loop over the momentum array because we have to calculate an integral over position space for each point in the momentum spectrum.

The remainder of the code is about 60 to 70 lines of setting up, running and saving the animation. Please note that it can take a long time to encode a four to five minute video at near HD.