Python calculations of Bohmian trajectories

Dane Odekirk

December 12th, 2012

Abstract.

The Python programming language is used to calculate Bohmian trajectories for two interfering wave-packets. A fourth-order Runge-Kutta is implemented to numerically solve the differential equations that describe the trajectories. Trajectories are computed on a 1.7Ghz Core i5 processor using Python version 2.7.2 with the NumPy, SymPy and Matplotlib packages. All results agree with current de Broglie-Bohm theory. The Python scripts are referenced in the appendix.

1 Introduction

Quantum mechanics is famed for introducing uncertainty to a once certain world. The de Broglie-Bohm theory rectifies this with a causal interpretation of quantum mechanics by prescribing definite trajectories to particles without sacrificing predictive accuracy. This is a stark contrast to the Copenhagen interpretation where a system's behavior is nondeterministic. The ramifications of the de Broglie-Bohm interpretation remedy the concept of wave-particle duality since quantum systems can be predicted without losing the particles. This gives rise to a narrative of the particles that can be discussed, analyzed and is not hidden nor ignored as it with the Copenhagen interpretation.

The de Broglie-Bohm theory proposes that a system of n particles has trajectories described by

$$\dot{r} = \frac{\hbar}{m_i}\mathrm{Im}[\frac{\nabla_i\Psi(r_1,\dots,r_n,t)}{\Psi(r_1,\dots,r_n,t)}] \tag{1}$$

for each particle i; where Ψ is the total wave function [3]. This is recognized as the guiding equation. First presented by de Broglie in 1927, the de Broglie interpretation of quantum mechanics was quickly abandoned after a confident rebuttal by Pauli during a conference. Although de Broglie's defense was accurate, Pauli and his objections garnered support and momentum away from the de Broglie interpretation and it was eventually abandoned as the Copenhagen interpretation became more popular. In 1952, however, Bohm reintroduced the theory with proper rebuttals of Pauli's objections [1].

Bohm's theory, also known as Bohmian mechanics or de Broglie-Bohm theory, is formulated via the Schrödinger equation [1]. To illustrate, consider the one-particle Schrödinger equation given by

$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) = \left[\frac{-\hbar}{2m}\nabla^2 + V(\mathbf{r},t)\right]\Psi(\mathbf{r},t) \tag{2}$$

where V is the potential and Ψ is the wave function. When the wave function is written in Euler form

$$\Psi(\mathbf{r},t) = \sqrt{P}\exp(\frac{iS}{\hbar}) \tag{3}$$

where P and S are real, dimensionless functions, the Schrödinger equation can be evaluated. Separating the result into real and imaginary parts yields the following equations:

$$\frac{\partial P}{\partial t} = -\nabla\cdot(P\frac{\nabla S}{m}) \tag{4}$$

$$\frac{\partial S}{\partial t} = -\frac{1}{2}\left[\frac{(\nabla S)^2}{m} - 2V(\mathbf{r}) + Q\right] \tag{5}$$

where Q is given by

$$Q = -\frac{\hbar^2}{2m}\frac{\nabla^2\sqrt{P}}{\sqrt{P}} \tag{6}$$

For the remainder of this paper, the constants m and $\hbar$ are set equal to one.

Bohm recognized when evaluating equation (4) at the classical limit, $\hbar \to 0$, that S is the solution to the classical Hamilton-Jacobi equations of motion [1]. The Hamilton-Jacobi equations are accurate for classical systems, however, are incomplete for the quantum realm. Specifically, the function Q approaches zero classically and only exists at the quantum limit. This function Q is known as the quantum potential and is the difference between the quantum and classical Hamilton-Jacobi equations of motion. It also plays a significant role in the behavior of de Broglie-Bohm particle trajectories. Bohm quotes the following theorem [1]

> If we consider an ensemble of particle trajectories which are solutions of the equations of motion, then it is a well-known theorem of mechanics that if all these trajectories are normal to any given surface of constant S, then they are normal to all surfaces of constant S, and $\nabla S(x)/m$ will be equal to the velocity vector, $\mathbf{v}(\mathbf{x})$, for any particle passing the point $\mathbf{x}$.

Applying this theorem yields two results. First, equation (4) can be rewritten in terms of a particle's velocity since $\nabla S(x)$ is equal to the velocity $\mathbf{v}$. P is therefore consistent with the probability density for these particles, where $P\mathbf{v}$ represents the mean current of particles. Equation (4) then represents the conservation of probability. Second, equation (5) is consistent with the Hamilton-Jacobi for a system of such particles. Unlike its classical counterpart, however, the particles are affected by two types of potentials: the classical potential V and the quantum potential Q.

To summarize, solving the Schrödinger equation with a wave function in Euler form results in a set of equations. These set of equations are equal to the Hamilton-Jacobi equations of motion at the classical limit and introduce the new quantum potential Q at the quantum limit. Furthermore, they imply a set of calculable trajectories for each particle in the system; defining a narrative that is nonexistent in the Copenhagen interpretation.

In order to calculate Bohmian trajectories of a particle, its x and y position must be determined at any given time t. The x component of the particle's position can be found via

$$x(t) = x(0) + k_x t \tag{7}$$

where k_x is the velocity in the x direction [3]. This simplifies the calculation by allowing us to focus specifically on the y coordinate realizing that the x coordinate is simply proportional to the time t. The y coordinate of a particle at time t is found by feeding the wave function Ψ that describes the system into equation (1). With initial random values of t_0 and y_0, equation (1) is evaluated over iterations of times t between t_i and t_f to calculate the particle's y coordinate at each specific time t. Compiling this solution with the x coordinate yields the trajectory of a particle with initial position x_0 and y_0.

In this paper a fourth-order Runge-Kutta is implemented to compute the Bohmian trajectories defined by equation (1) [6]. Interfering wave packets are investigated for two related systems. Python 2.7.2 is the programming language of choice because of its syntax, built in debugger, and open-sourced scientific packages. The language itself is freely available and open-sourced as well. The specific packages used are NumPy, SymPy and Matplotlib which provide numerical functions, symbolic math and plotting functions respectively. Full understanding of these packages is not necessary to understand or run the Python code; although some background knowledge of their capabilities, purposes and implementations could prove helpful.

2 Double-slit experiment

Bohmian trajectories are calculated for a double-slit setup. The classical double slit experiment consists of the following parts: a source that emits particles, a barrier with two slits A and B cut into it, and a screen that captures and records the particles. This is illustrated in Figure 1. The system implemented in this investigation shares these characteristics and makes no special alterations to the traditional experimental setup.

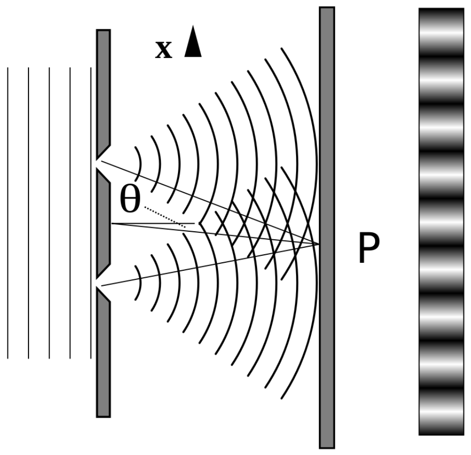

Figure 1
Traditional double slit setup [7]. Bohmian trajectories are calculated for such a system.

When a particle goes through either slit its wave function is described by $\psi_A(\mathbf{r},t)$ and $\psi_B(\mathbf{r},t)$ for slits A and B respectively. The wave that leaves the slit has an exact solution described by the following time-dependent equation [4]

$$\psi_n(\mathbf{r},t) = (2\pi\sigma^2)^{-1/4} \exp\left[\frac{(y_i - Y)^2}{4\sigma_0\sigma_t} + i\left[k_x x_i - \frac{k_x^2 t}{2}\right]\right] \tag{8}$$

where

$$\sigma_t = \sigma_0(1 + \frac{it}{2\sigma_0^2}) \tag{9}$$

This definition assumes the particles are traveling in free space and the classical voltage V is zero. The total wave function needed to calculate trajectories with equation (1) is described by the following equation

$$\Psi(\mathbf{r},t) = N[\psi_A + \psi_B] \tag{10}$$

where ψ_A and ψ_B are defined by equation (8) and N is a normalization constant.

Figures 2 and 3 illustrate the time-dependence of the wave function Ψ. Both the real and imaginary parts of of the wave function, described by equation (10), are plotted at times $t = 0$ and $t = 1$. Both Figures 2 and 3, span y values from -13 to 13 with a spacing of 0.01, accounting for 2600 points to be plotted. Good consistency is found between these figures and figures in reference [4] on which these graphs are based. The code to produce these figures is shown in appendix A.

According to equation (1) the imaginary part of the wave function plays an important role in guiding particles along their Bohmian trajectories. For a more revealing perspective, the imaginary part of the wave function is plotted three-dimensionally over time in Figure 4 with a color grading representing the gradient along its surface. Note that the x axis has been replaced by the time t as they are proportional to one another. Values for y once again span between -13 and 13 with a spacing of 0.01 and values for t span between 0 and 1 with a spacing of 0.01. These values are chosen on a rather subjective basis of time - as decreasing the spacing increases the computation time. Generating Figure 4 takes approximately 1.35 seconds with the given values. The code to generate this graph is shown in appendix B.

Figure 4 clarifies the underlying mechanisms that govern the Bohmian trajectories. Notice the imaginary parts of Figures 2 and 3 agree with the cross-section parallel to x of Figure 4 at times $t = 0$ and $t = 1$. The crest and troughs of the wave function govern the direction of the particle. Higher

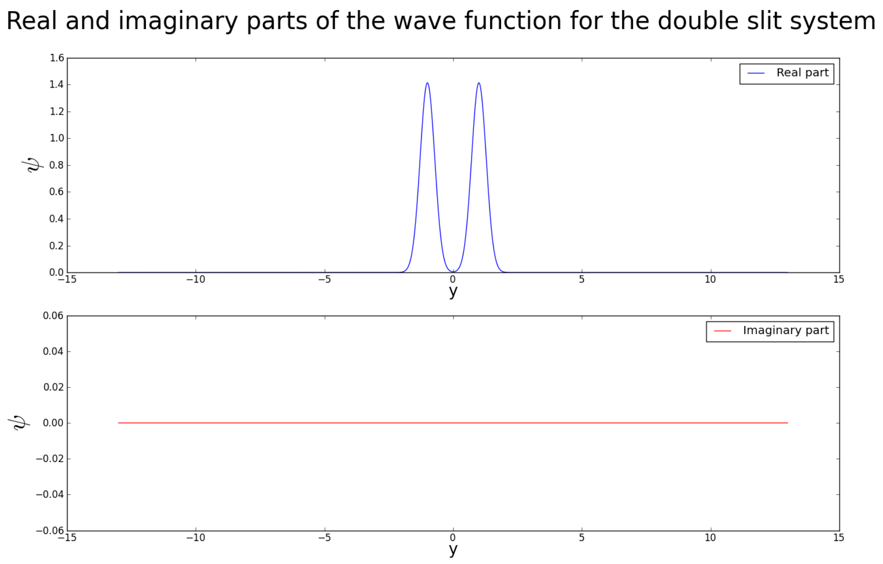

Figure 2
Two-dimensional breakdown of the real (blue) and imaginary (red) parts of equation (10) evaluated at $t = 0$. Values of y span from -13 to 13 with spacing Δy of 0.01.

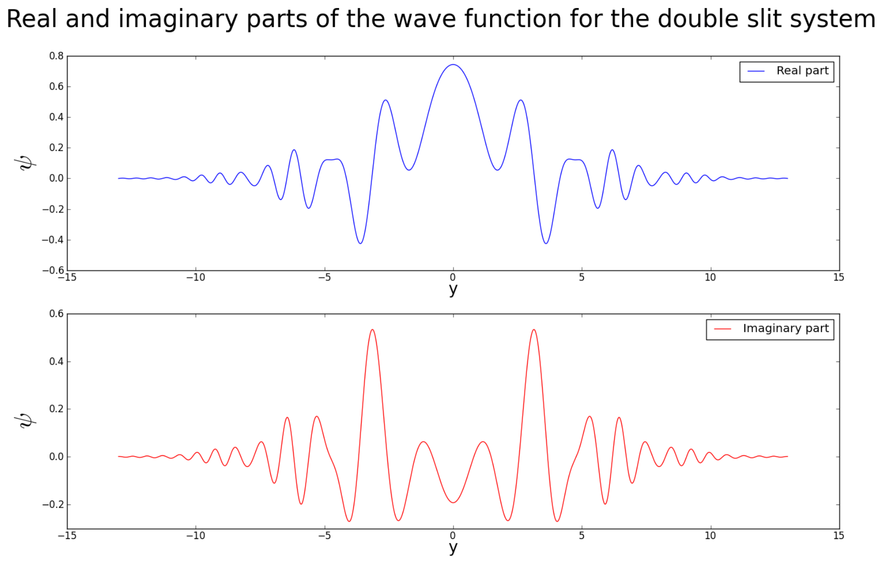

Figure 3
Two-dimensional breakdown of the real (blue) and imaginary (red) parts of equation (10) evaluated at $t = 1$. Values of y span from -13 to 13 with spacing Δy of 0.01.

crests require higher energy while the troughs require lower energy. The trajectories follow the path of lowest energy. Not depicted in Figure 4 is the quantum potential Q that also affects the trajectories - it is plotted in reference [1]. When Q is plotted three-dimensionally, it also illustrates the crests and troughs that govern the particle's trajectory [1].

Of more interest, however, is the concept of the trajectory. The narrative behind the Copenhagen interpretation is one that focuses on predictions and observations while ignoring how the observation came to be. With the de Broglie-Bohm interpretation, the prediction is still valid however the particle's trajectory, or the history of how it arrived at the observation, can still be calculated and discussed. For example, the de Broglie-Bohm interpretation shows that a particle does in fact go through a specific slit; either one or the other. It does not have a predictive measure to calculate which slit the particle will go through prior to reaching the slits, however, does define a historical trajectory that leads to a single, specific slit. This is not the case in the Copenhagen interpretation where the particle's probability wave passes through both slits and the concept of a physical particle passing through a single slit is irrelevant.

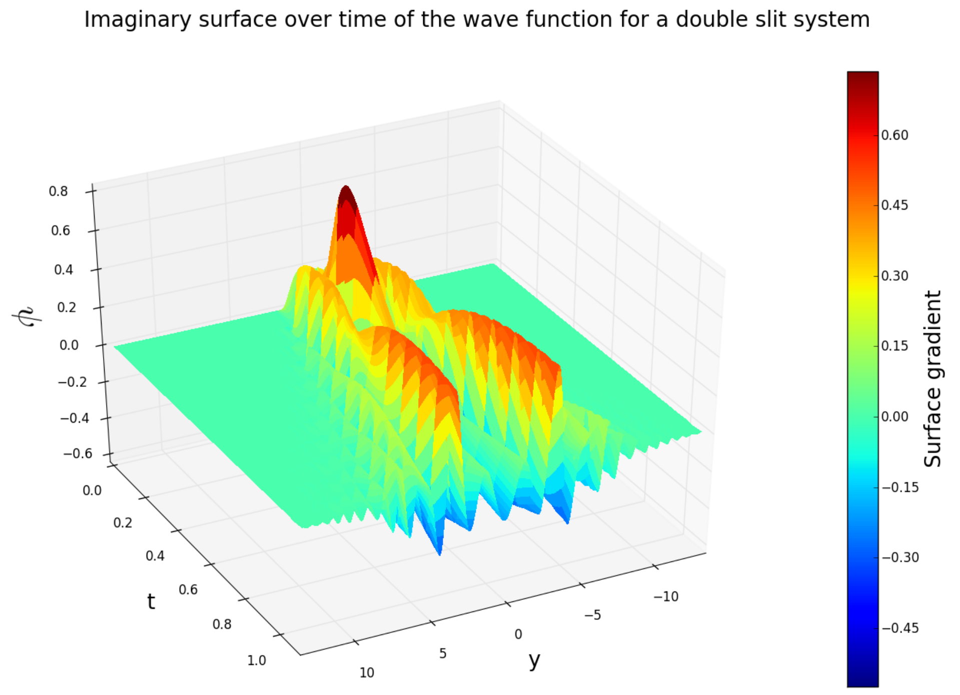

Figure 4
Three-dimensional plot of the imaginary surface of equation (10) as it propagates through time. This figure exposes the physical importance of the wave function, which inevitably governs the path of particles at they traverse from $t = 0$ to $t = 1$.

3 Double-slit trajectories

Figures 5, 6 and 7 plot the Bohmian trajectories for particles in a double slit setup; numerically solved via Python. There are two main commonalities between these three figures. First, an interference pattern emerges in all three as expected. This is promising since the double slit experiment does produce physical interference patterns that cannot be contradicted by the calculation. Second, all three figures exhibit trajectories that do not cross paths, which is an intrinsic characteristic of Bohmian trajectories.

All the trajectories plotted in Figures 5, 6 and 7 were calculated with the following constants: a distance between slits of 2 units, a slit width of 1 unit and an initial x velocity of 0.1 units per second. The time spacing Δt between calculated points is 1/100 seconds. Calculating the graphs takes approximately 26.4, 28.3 and 66.3 seconds respectively. The initial distribution is a random uniform distribution across each slits. The Python class that generates these plots is shown in appendix C.

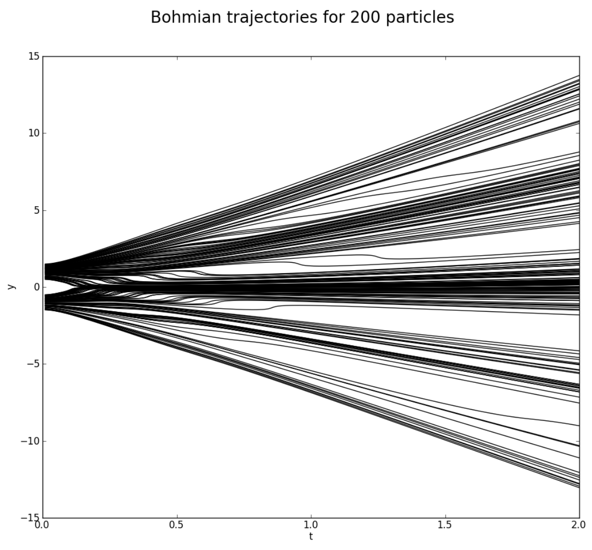

Figure 5
200 particle trajectories with a random uniform distribution over each slit.

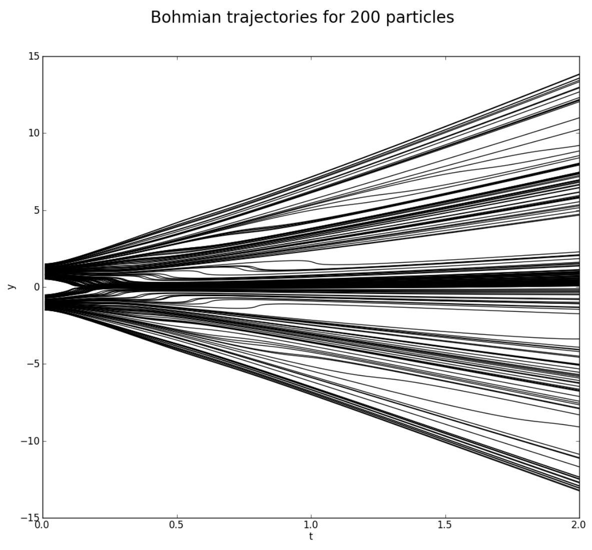

Figure 6
200 particle trajectories with an random uniform distribution over each slit. There is no difference between this and Figure 5 except the initial position of the particles.

The basic criteria for Bohmian trajectories appear correct. There is a visual departure from classical trajectories in Figures 5, 6 and 7 as well. Classically, if a particle starts at the slit with a distinct position and velocity it will continue along the direction of the particle's velocity in a straight line unless acted upon by an external force. The trajectories depicted do not follow straight paths and appear to veer at specific points. The cause of these detours is the quantum potential given by equation (6). It provides a force, in the quantum realm, that governs the particles into trajectories foreign to classical behavior.

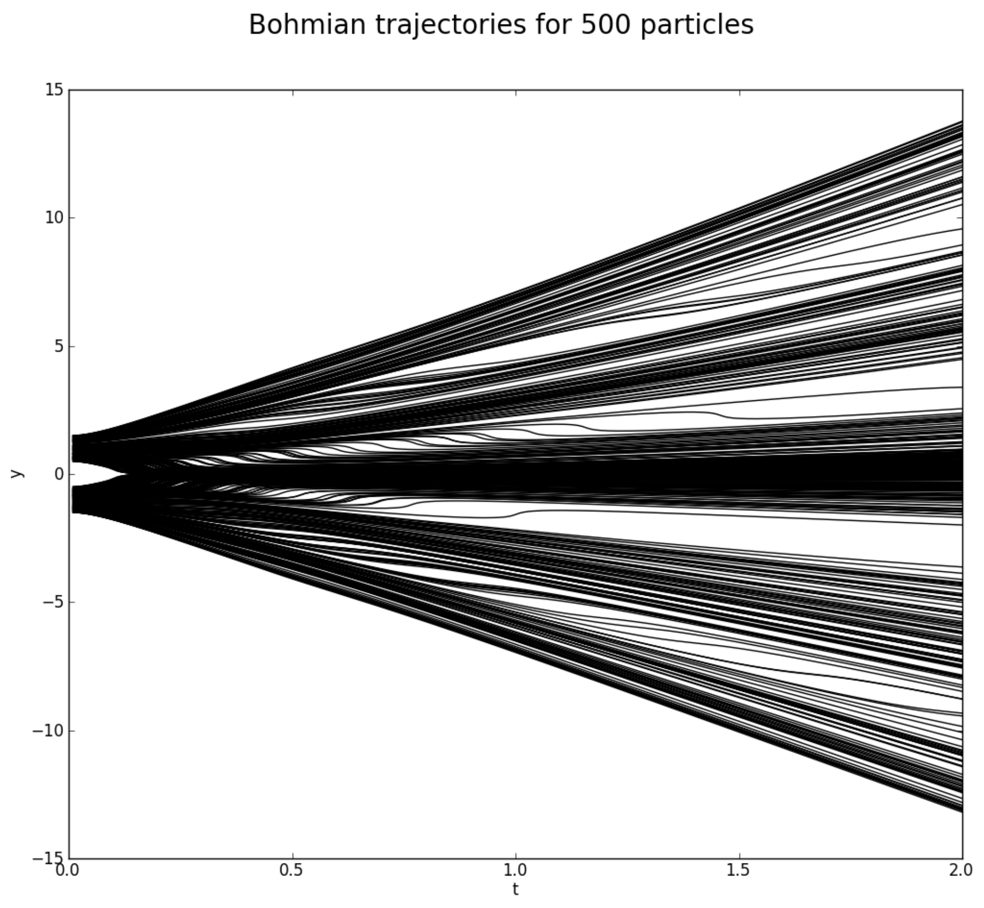

Figure 7
500 particle trajectories with a random, uniform distribution over each slit.

When this experiment is performed, Figure 8 illustrates what a screen registers at different times. Both the Copenhagen and de Broglie-Bohm interpretations predict this interference pattern. The Copenhagen interpretation, however, ignores an explanation of the particle itself until it is measured by the screen, while the de Broglie-Bohm interpretation offers a narrative of the particle's trajectory. As stated previously, a fundamental distinction is the de Broglie-Bohm notion that a single particle goes through a specific slit. The probability wave used in the Copenhagen interpretation does not quite offer this distinction and suggests that the particle goes through both slits and interferes with itself. The trajectories in Figures 5 and 6 would be representative of a screen at time c in Figure 8, while Figure 7 would be representative of a screen at time d.

4 Secondary setup and trajectories

The Python class used to generate the previous trajectories is extended to fit a new wave function. Similar to the double-slit experiment, this secondary setup consists of a source that emits particles and a screen that captures the particle. Instead of the particle going through two slits, however, it hits a beam-splitter with 50% chance of the particle going in one direction or another. After passing through the beam-splitter the particle hits a mirror and reflects towards the screen as illustrated in

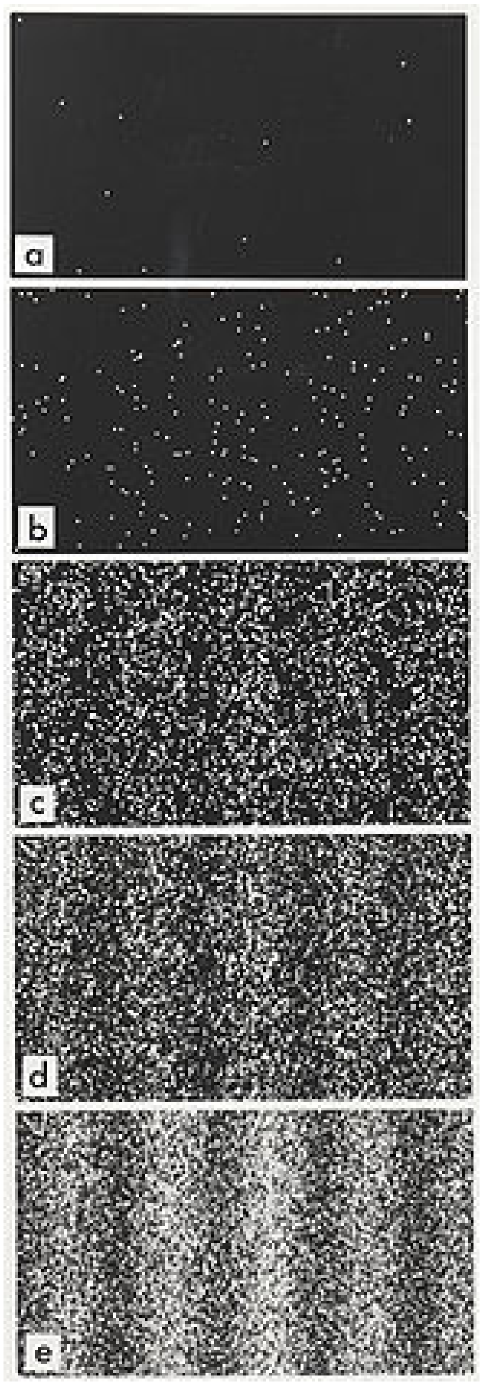

Figure 8
Screen results of the double slit experiment over time [7].

Figure 9.

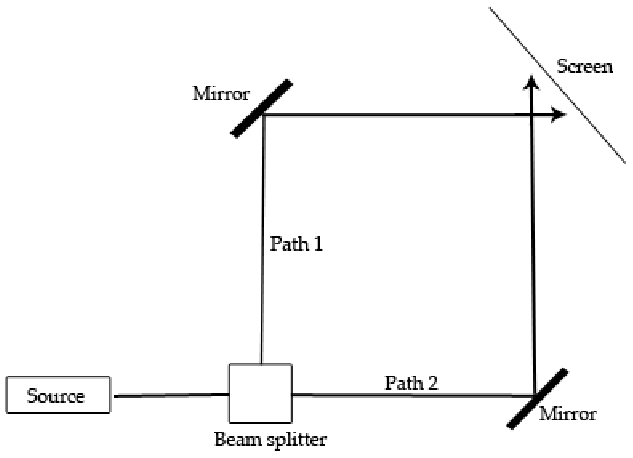

Figure 9
The configuration setup for the interfering wave packet experiment.

The wave packets are assumed to be Gaussian in nature and do not spread with time [2]. Equation (1) is used again to calculate the Bohmian trajectories for the wave function Ψ that describes this system. The contour plot shown in Figure 10 illustrates the wave function Ψ at times $t = -15$, $t = 0$ and $t = 15$ seconds as the wave packets ψ_A and ψ_B propagate towards each other on intersecting paths. Initially, at $t = -15s$, the two wave packets are separate entities described by a single wave function traversing orthogonal paths. At $t = 0$, the wave packets are completely superimposed upon each other forming an interference pattern with nodes and anti-nodes as the two Gaussian wave packets interfere. As the wave packets continue to propagate the interference pattern diminishes until eventually the wave packets are separate entities again; propagating along the same path as they initially were. The code to generate Figure 10 is shown in appendix D. Code to generate a continuous animation of this contour plot is shown in appendix E.

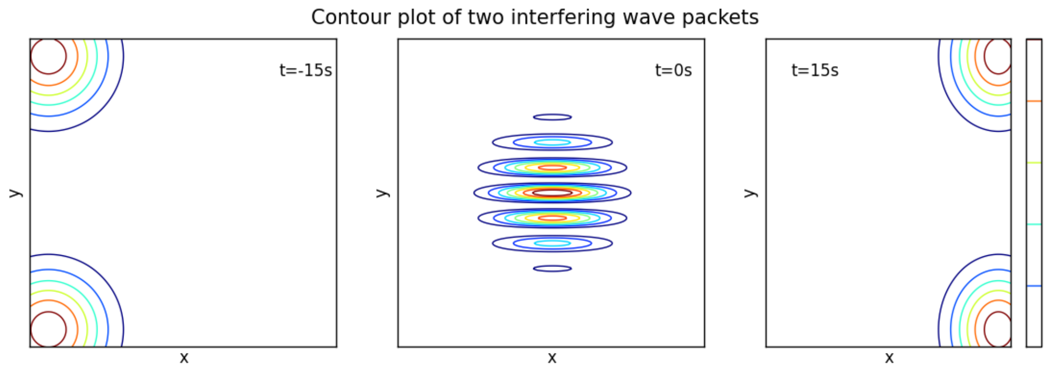

Figure 10
Contour drawing of two interfering wave packets at times $t = -15s$, $t = 0s$ and $t = 15s$.

Calculating the Bohmian trajectories for the system yields the trajectories plotted in Figure 11. The time t spans from -15 seconds to 15 seconds with iterations of $1/100s$. The particles are initially distributed with equidistant spacing with the bottom and top packets offset by a 1/2 step to avoid symmetry over the $y = 0$ axis. Plotting 100 trajectories takes approximately 5 minutes and 55 seconds. Again, the calculations only find the y coordinate of each particle since the x coordinate is proportional to the time t. The Python code to generate Figure 11 is shown in appendix F and is a subclass of the Python class in appendix C that accommodates the new initial conditions and wave function Ψ.

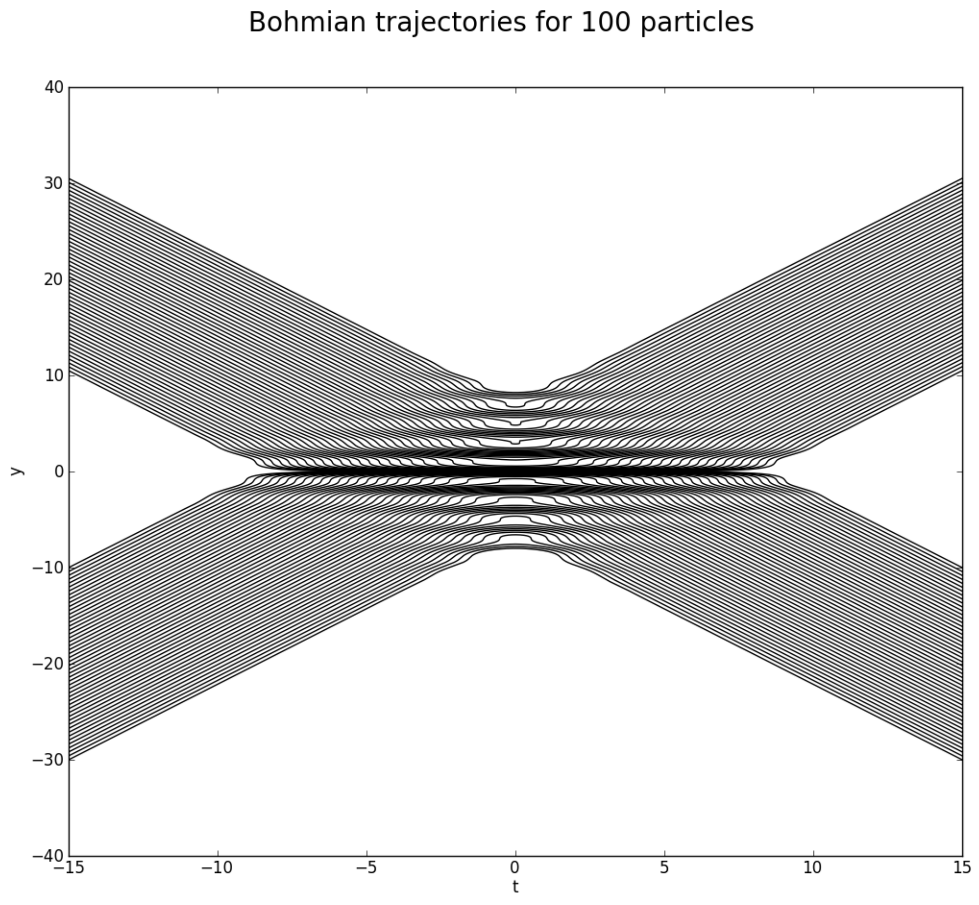

Figure 11
Bohmian trajectories plotted for wave function Ψ describing two interfering Gaussian wave packets. 100 particles are initially distributed with equidistant spacing on the $t = -15s$ axis. The bottom and top packets are offset by a 1/2 step to prevent symmetry over the $y = 0$ axis.

An instinctive interpretation of the interfering wave packets depicted in Figure 10 can lead one to assume that the two wave packets head towards each other, interfere with each other, and then continue along their original path. The Bohmian trajectories show that as the wave packets ψ_A and ψ_B begin to interfere with one another the Bohmian trajectories begin detouring. When the wave packets are fully superimposed at $t = 0s$, the trajectories are almost parallel with the t axis, never crossing the $y = 0$ axis. As ψ_A and ψ_B continue propagating forward, the trajectories eventually turn around and head back towards the upper or lower region they originated from; creating another symmetry along the $t = 0$ axis. This may indicate that the wave packets behave similarly and simply appear to continue along their original path. The familiar characteristics of Bohmian trajectories are still observed: the trajectories do not cross and veer from classical trajectories. Furthermore, the particle trajectories follow the pattern illustrated in Figure 11 regardless of the initial distribution.

5 Conclusion

In conclusion, Bohmian trajectories were numerically calculated with Python 2.7.2. A fourth-order Runge-Kutta was implemented to numerically solve the differential equation (1) that describes the trajectories. Basic corroboration with known results was found and the expected interference patterns were observed. The effects of the quantum potential were noticeable as well via the deviations in the trajectories from their classical counterparts. All the Python scripts are referenced in the appendix and have the NumPy, SymPy and Matplotlib packages as dependencies. Computation times were relatively quick and given the open-sourced foundation of Python and its scientific packages the results can be freely replicated.

References

[1] David Bohm. *A Suggested Interpretation of the Quantum Theory in Terms of "Hidden" Variables. I.* Physical Review, 1951.

[2] Vladimir Chaloupka, 2012. University of Washington.

[3] E. Guay and L. Marchildon. *Two-particle interference in standard and Bohmian quantum mechanics.* J. Phys. A: Math. Gen. 36, 5617, 2003.

[4] E. Guay and L. Marchildon. *Wave functions and Bohmian trajectories in interference phenomena.* Elsevier Science, 2004.

[5] C. Philippidis, C. Dewdney, and B. J. Hiley. *Quantum Interference and the Quantum Potential.* Il Nuovo Cimento 52 B, 15, 1978.

[6] David Stygstra, 2012. Improved rk4 implementation, dowsa.com.

[7] Wikipedia. Double-slit experiment, 2012. wikipedia.org/wiki/Double-slit_experiment.

A Real and imaginary parts of the wave function for the double slit system

```
from scipy import *
from numpy import *
from pylab import *

def psi_y(y,t=1.):
    Y  = 1.
    s0 = .2
    k  = .1
    x  = 0.
    st  = s0*(1+ (1j*t)/(2*s0**2))
    N  = (2*pi*st**2)**-.25
    return N*exp((-((y-Y)**2)/(4*s0*st)) + 1j*(k*x-((t*k**2)/2)))

y = arange(-13,13,.01)
Z = psi_y(y)+psi_y(-y)

figure(figsize=(12,10))

suptitle("Real and imaginary parts of the wave function for the double slit system",
            fontsize=20)

subplot(211)
xlabel("y",fontsize=20)
ylabel(r"$\psi$",fontsize=30)
plot(y,real(Z), "b", label="Real part")
legend()

subplot(212)
xlabel("y",fontsize=20)
ylabel(r"$\psi$",fontsize=30)
plot(y,imag(Z), "r", label="Imaginary part")
legend()

show()
```

B Imaginary surface over time of the wave function for a double slit system

```
from mpl_toolkits.mplot3d import Axes3D
from numpy import *
from pylab import *

def psi_y(y,t=0.):
    Y  = 1.
    s0 = .2
    k  = .1
    x  = 0.
    st  = s0*(1+ (1j*t)/(2*s0**2))
    N  = (2*pi*st**2)**-.25
    return N*exp((-((y-Y)**2)/(4*s0*st)) + 1j*(k*x-((t*k**2)/2)))

t = arange(-0,1.1,.01)
y = arange(-13,13,.01)
Y,T = meshgrid(y, t)
Z = psi_y(Y,T)+psi_y(-Y,T)

fig = figure(figsize=(17,10))
suptitle("Imaginary surface over time of the wave function for a double slit system",
            fontsize=20)

ax = fig.gca(projection="3d")
ax.set_zlim([-10,10])

surf = ax.plot_surface(Y, T, imag(Z), cmap=cm.jet, linewidth=0, antialiased=False)
ax.set_xlabel("y",fontsize=20)
ax.set_ylabel("t",fontsize=20)
ax.set_zlabel(r"$\psi$",fontsize=30)
ax.view_init(elev=36, azim=63)

cbar = colorbar(surf)
cbar.set_label("Surface gradient", fontsize=20)

show()
```

C Bohmian trajectories for the double-slit setup

```
import time
import os
import numpy as np
import sympy as sp

from pylab import *
from random import getrandbits

class Bohm:
    def __init__(self,particles=200,plot=False,save=False,**kwargs):
        self.start = time.time()

        self.slitdistance = kwargs.get('slitdistance',1.)
        self.slitwidth    = kwargs.get('slitwidth',.2)
        self.velx         = kwargs.get('velx',.1)
        self.dt           = kwargs.get('dt',1/100.)
        self.save         = save

        self.y = sp.var("y")
        self.t = sp.var("t")
        self.x  = y*t

        self.st = self.slitwidth*(1+ (1j*t)/(2*self.slitwidth**2))
        self.N  = (2*pi*self.st**2)**-.25

        inits = np.array([x for x in np.random.uniform(-1.5, 1.5, particles*2)
                             if x < -.5 or x > .5][:particles])

        self.inits     = kwargs.get('inits', inits)
        self.t0        = kwargs.get('t0',0)
        self.tf        = kwargs.get('tf',2)

        self.PSI = self.psi()
        # turn the symoblic functions into callable python functions
        self.wave         = sp.lambdify([y,t],self.PSI,"numpy")
        self.del_wave     = sp.lambdify([y,t],self.PSI.diff(y),"numpy")

        if plot: self.plot()

    def rk4(self,x, h, y, f):
        k1 = h * f(x, y)
        k2 = h * f(x + 0.5*h, y + 0.5*k1)
        k3 = h * f(x + 0.5*h, y + 0.5*k2)
        k4 = h * f(x + h, y + k3)
        return x + h, y + (k1 + 2*(k2 + k3) + k4)/6.0

    def psi(self):
        return (self.wavefunction(self.x,self.y,self.t)
                    + self.wavefunction(self.x,-self.y,self.t))

    def wavefunction(self,x,y,t):
        return self.N*sp.exp((-((y-self.slitdistance)**2.)/(4.*self.slitwidth*self.st))
                          + 1.j*(self.velx*x-((t*self.velx**2.)/2.)))

    def trajectories(self,t,state):
        y,vel = state
        return imag(self.del_wave(y,t)/self.wave(y,t))

    def plot(self):
        figure(figsize=(12,10))
        xlim(self.t0,self.tf)
        suptitle("Bohmian trajectories for %d particles" % len(self.inits), fontsize=20)
        xlabel("t")
        ylabel("y")
        for index,y0 in enumerate(self.inits):
            x = []
            y = []
            t = self.t0
            state = np.array([y0,0])
            while t < self.tf:
                t, state = self.rk4(t,self.dt,state,self.trajectories)
                x.append(t)
                y.append(state[0])
            plot(x,y,color="black")
            print "Completed trajectory %d completed" % index

        print "Completed in %s seconds" % str(time.time() - self.start)
        if self.save:
            filename = ("%s/Desktop/%d-bohmian-trajectories-%s.png" %
                        (os.path.expanduser("~"), len(self.inits), os.urandom(8).encode('hex')))
            savefig(filename)
            print "Saved to %s" % filename
        show()

if __name__ == '__main__':
    Bohm(plot=True,save=True)
```

D Contour plot

```
from numpy import *
from pylab import *

def wave(x,y,t=15.,a=36.,c=1.,theta=pi/4.):
    k0 = sqrt(2.)*pi/2.
    sint = sin(theta)
    cost = cos(theta)
    return ( exp(1.j*(k0*(x*cost-y*sint)-c*t)) *
             exp(-((x*cost-y*sint)-c*t)**2/a) *
             exp(-(x*sint+y*cost)**2/a) +
             exp(1.j*(k0*(x*cost+y*sint)-c*t)) *
             exp(-((x*cost+y*sint)-c*t)**2/a) *
             exp(-(x*sint-y*cost)**2/a) )

delta = 0.1
x = np.arange(-12.0, 12.0, delta)
y = np.arange(-12.0, 12.0, delta)
X, Y = np.meshgrid(x, y)

figure(figsize=(14,4))
suptitle('Contour plot of two interfering wave packets',
            fontsize=15)

ax = subplot(131)
#ax.text(200,220,"t=-15s")
text(0.9, 0.9,"t=-15s",
        ha="center", va="center", transform=ax.transAxes)
xlabel("x")
ylabel("y")
ax.xaxis.set_ticks([])
ax.yaxis.set_ticks([])
Z = wave(X,Y,t=-15.)
contour(Z*conj(Z))

ax = subplot(132)
text(0.9, 0.9,"t=0s",
        ha="center", va="center", transform=ax.transAxes)
xlabel("x")
ylabel("y")
ax.xaxis.set_ticks([])
ax.yaxis.set_ticks([])
Z = wave(X,Y,t=0)
contour(Z*conj(Z))
```

```
ax = subplot(133)
text(0.2, 0.9,"t=15s",
        ha="center", va="center", transform=ax.transAxes)
xlabel("x")
ylabel("y")
ax.xaxis.set_ticks([])
ax.yaxis.set_ticks([])
Z = wave(X,Y,t=15.)
c = contour(Z*conj(Z))

cbar = colorbar(c)
cbar.set_ticks([])

legend()

show()
```

E Animated contour plot

```
import os
import time

from numpy import *
from pylab import *

cwd = os.getcwd() + "/temp"

def wave(x,y,t=15.):
    k0 = sqrt(2.)*pi/2.
    a = 3.
    c = 1.
    sint = sin(pi/4.)
    cost = cos(pi/4.)
    return ( exp(1.j*(k0*(x*cost-y*sint)-c*t))*5.*sqrt(pi) *
             exp(-((x*cost-y*sint)-c*t)**2./4./a**2.)/a*5.*sqrt(pi) *
             exp(-(x*sint+y*cost)**2./4./a**2.)/a +
             exp(1.j*(k0*(x*cost+y*sint)-c*t))*5.*sqrt(pi) *
             exp(-((x*cost+y*sint)-c*t)**2./4./a**2.)/a*5.*sqrt(pi) *
             exp(-(x*sint-y*cost)**2./4./a**2.)/a )

delta = 0.1
x = np.arange(-12.0, 12.0, delta)
y = np.arange(-12.0, 12.0, delta)
X, Y = np.meshgrid(x, y)

t = -25.
while t<25.:
    clf()
    Z = wave(X,Y,t)
    contour(Z*conj(Z))
    t+=1.
    pause(.001)
    if t == 25:
        t = -25.
```

F Bohmian trajectories for the second setup

```
import sympy as sp

from numpy import *
from bohm import Bohm

class Second(Bohm):
    def psi(self):
        return self.wavefunction(self.y,self.t)

    def wavefunction(self,y,t):
        # initial conditions
        a = 3.
        c = 1.
        k0 = sqrt(2.)*pi/2.
        x = 1.
        sint = sin(pi/4.)
        cost = cos(pi/4.)
        return ( sp.exp(1.j*(k0*(x*cost-y*sint)-c*t)) *
                 sp.exp(-((x*cost-y*sint)-c*t)**2/4/a**2) *
                 sp.exp(-(x*sint+y*cost)**2/4/a**2) +
                 sp.exp(1.j*(k0*(x*cost+y*sint)-c*t)) *
                 sp.exp(-((x*cost+y*sint)-c*t)**2/4/a**2) *
                 sp.exp(-(x*sint-y*cost)**2/4/a**2) )

if __name__ == "__main__":
    inits = linspace(-10,-30,50)
    inits = append(inits, linspace(10.5,30.5,50))
    Second(save=True,plot=True, inits=inits,t0=-15.,tf=15.)
```