

# Discretization of the Quantum Dynamical Equations of Motion - One Dimension

Bill Page ([bill.page@newsynthesis.org](mailto:bill.page@newsynthesis.org))

Draft August 6, 2016

## Abstract

### **From Poirier's Bohmian Mechanics without Wavefunctions, to Hall's Many Interacting Worlds in One Dimension**

In this note we show that Hall's "toy" model of quantum mechanics as many interacting interacting classical worlds in one dimension follows directly from the approximation of Poirier's quantum mechanical equations of motion as difference equations with a judicious choice of difference operators. These calculations are a preliminary exercise for the case of more than one dimension. The main emphasis is methodological with the main applications to follow in later articles.

We provide a numerical simulation that reproduces the results given previously by Poirier [?].

## 1 Introduction

In a general sense the quantum mechanical equations of motion originates with the work of Erwin Madelung [1] on quantum hydrodynamics. David Bohm [2] rediscovered and extended an interpretation of quantum mechanics originating with Louis de Broglie [3], aka. Bohmian mechanics, which the notion of particles "guided" by the wavefunction. Peter Holland [4] and co-workers have continued to extend the de Broglie-Bohm theory in this direction, ultimately showing that it is possible to dispense with the wavefunction all together in favor of a continuous ensemble of quantum trajectories. Besides the conceptual metaphysical issues, this has lead to some new more powerful calculation methods that have become important for practical application of quantum mechanics to physical chemistry (e.g. Wyatt [5]). Poirier continues in this tradition while Hall, Deckert, and Wiseman [6] emphasize the interpretation of the quantum trajectories as interaction between parallel worlds reminiscent but differing from Hugh Everett's many-worlds interpretation of quantum mechanics.

In this article the Hamiltonian formulaion of these equations of motion is important. Conservation of energy is an important indicator of the quality of the numerical integration. The Störmer–Verlet method implemented in this article is well known and preserves the symplectic structure on the system phase space.

From the point of view of Poirier [7] this simulation is viewed as an approximation to the quantum dynamical field equations. One should think of this in hydrodynamic terms. But from the point of view of Hall, et al. these are discrete trajectories of the **same** particle in many different "parallel" worlds - but just one particle!

In the expressions which follow  $\mathbf{C}$  is the co-ordinate which indexes these worlds. One should think of this as the same particle in many different worlds interacting with the copy of itself in other worlds. It is a an important aspect of the equations of motion that trajectories of these particles never cross. The co-ordinate function  $\mathbf{C}$  is assumed to be "uniformized"<sup>1</sup> so that discretization of  $\mathbf{C}$  assigns an equal "density" to the particle in each world.

---

<sup>1</sup> Schiff and Poirier [7]

In the example simulation included in this article the starting distribution of the particle in these different worlds is a Gaussian distribution. But the distribution evolves as it interacts with the classical potential. Some trajectories are scattered from the potential while other tunnel through. From the point of view of Poirier, an important goal of this simulation is to provide an efficient calculation of the scattering and tunneling amplitudes. But from the point of view of Hall this is a just purely "classical" mechanical problem with a rather unusual form of interaction. It is the form of this interaction between worlds that leads to statistics that approximate quantum mechanics in the limit of a very large number of interacting worlds.

In this document we have used the computer algebra system Sage<sup>2</sup> with components of SciPy<sup>3</sup> and mpmath<sup>4</sup> for both symbolic and numeric calculations. This document and the source code for this work can be found online<sup>5</sup>. The inset bold text below and many of the formulas in this paper are input and output produced directly by the Sage system. For more details please refer to Appendix A.

For example we define some special symbols for Planck's constant and mass:

```
hbar = var('hbar', latex_name=r'\hbar')
m = var('m')
```

## 2 One Dimensional System

We show below that in one dimension Schiff and Poirier's quantum dynamical equation of motion, [7] eqs. (18-20),

$$m\ddot{x}^i + \frac{\partial V(x)}{\partial x^i} - \frac{\hbar^2}{4m} \frac{\partial}{\partial C^m} \left( K_i^k K_j^m \frac{\partial^2 K_j^l}{\partial C^k \partial C^l} \right) = 0 \quad (1)$$

that follows from a quantum potential of the form

$$Q = -\frac{\hbar^2}{4m} \left( K_j^k \frac{\partial^2 K_j^l}{\partial C^l \partial C^k} + \frac{1}{2} \frac{\partial K_j^l}{\partial C^l} \frac{\partial K_j^k}{\partial C^k} \right) \quad (2)$$

when discretized by replacing derivatives with difference operators, is equivalent to Hall's [6] "toy" expression eq. (24) for a finite number of classically interacting "parallel" worlds.

$$r_N(x_n; X) = \frac{\hbar^2}{4m} [\sigma_{n+1}(X) - \sigma_n(X)] \quad (3)$$

$$\sigma_n(X) = \frac{1}{(x_n - x_{n-1})^2} \left[ \frac{1}{x_{n+1} - x_n} - \frac{2}{x_n - x_{n-1}} + \frac{1}{x_{n-1} - x_{n-2}} \right]$$

By discretized we mean that the "uniformizing co-ordinate"  $\mathbf{C}$  will be replaced with discrete values representing a finite number of "worlds". Derivatives with respect to  $\mathbf{C}$  will be replaced with difference operators in the expressions for the quantum potential and quantum force.  $\mathbf{C}$  becomes a discrete index  $n$ .

In the following we write most expressions in their general vector form even though in one dimension considerable simplification is possible to connect with related articles where we use these expressions in more than one dimension.

Let  $E = [x_n]$  represent the configuration space of one particle in 1-D space with world co-ordinate  $n$ . In Sage this can be represented as:

<sup>2</sup>SageMath Version 6.10, Release Date: 2015-12-18 [8] on SageMathCloud [9]

<sup>3</sup>numpy Version 1.10.1 [10]

<sup>4</sup>mpmath Version 0.18 [11]

<sup>5</sup>L<sup>A</sup>T<sub>E</sub>X source pdf SageWorksheet. The online versions may be more up to date.

```

X = functions('x')
d = len(X)
C = (var('n'),)
E = tuple([x(*C) for x in X])

```

$$(x_n) \quad (4)$$

### 3 Forward/Backward Difference Approximation

In this approximation to the derivative each world interacts only locally but with the *two* nearest neighbors to the left and right.

The forward and backward difference operators

```

def DM(x,n):return x-x.subs(n==n-1)
def DP(x,n):return x.subs(n==n+1)-x

```

approximate derivatives.

To derive Hall's equations for a finite number of interacting worlds we replace derivatives with difference operators so the Jacobian becomes a  $1 \times 1$  matrix

```

J = fun(n) (matrix([[DM(E[j]),C[i]] for i in range(d)] for j in range(d))))

```

$$\begin{pmatrix} -x_{n-1} + x_n \end{pmatrix} \quad (5)$$

It will be convenient for our calculations to represent the inverse of the Jacobian in terms of its adjugate and delay evaluation of the determinant until needed. The adjugate “matrix” is just

```

A = fun(n) (J(n).adjoint())

```

$$\begin{pmatrix} 1 \end{pmatrix} \quad (6)$$

so the inverse “matrix” is

```

K = fun(n) (A(n)/J_(n))

```

$$\begin{pmatrix} \frac{1}{|J|_n} \end{pmatrix} \quad (7)$$

where the symbol  $|J|_n$  represents the determinant. We will eventually need to replace the symbolic determinants with then explicit expressions

```

_J = fun(n) (factor(J(n).determinant()))
dets = {J_(n+i):_J(n+i) for i in range(-2,3)}

```

$$-x_{n-1} + x_n \quad (8)$$

It is True that  $J(n)^{-1} = K(n)$  where  $|J|_n = J(n)$ .

### 3.1 Quantum Potential

The Schiff and Poirier expression for the quantum potential (2) can be approximated in terms of difference operations. Let  $Q(n) = Q_2(n) + Q_3(n)$  be the 2nd and 3rd order terms, respectively.

$$Q = \text{fun}(n) \left( -(\hbar^2/4/m) * \sum \left( \sum \left( \sum \left( K(n)[k,j] * DM(DP(K(n)[1,j], C[1]), C[k]), 1/2 * DP(K(n)[1,j], C[1]) * DP(K(n)[k,j], C[k])) \right) \right) \right) \right)$$

$$-\frac{\hbar^2}{8m|J|_{n+1}^2} + \frac{3\hbar^2}{8m|J|_n^2} - \frac{\hbar^2}{4m|J|_{n-1}|J|_n} \quad (9)$$

$$Q_2 = \text{fun}(n) \left( -(\hbar^2/4/m) * \sum \left( \sum \left( \sum \left( 1/2 * DP(K(n)[1,j], C[1]) * DP(K(n)[k,j], C[k]) \right) \right) \right) \right)$$

$$-\frac{\hbar^2}{8m|J|_{n+1}^2} - \frac{\hbar^2}{8m|J|_n^2} + \frac{\hbar^2}{4m|J|_{n+1}|J|_n} \quad (10)$$

$$Q_3 = \text{fun}(n) \left( -(\hbar^2/4/m) * \sum \left( \sum \left( \sum \left( K(n)[k,j] * DM(DP(K(n)[1,j], C[1]), C[k]), \text{for } k \text{ in range}(d) \text{ for } j \text{ in range}(d) \text{ for } l \text{ in range}(d) \right) \right) \right) \right)$$

$$\frac{\hbar^2}{2m|J|_n^2} - \frac{\hbar^2}{4m|J|_{n+1}|J|_n} - \frac{\hbar^2}{4m|J|_{n-1}|J|_n} \quad (11)$$

In the discretized expressions it will be convenient to replace the  $X$  co-ordinates with scalar quantities. The forward and backward difference approximations will then require  $2 \times 5 \times 5 = 50$  scalar quantities.

$$\begin{aligned} \text{xyk} &= [X[i](n+j) \\ &\quad \text{for } j \text{ in range}(-2,3) \text{ for } i \text{ in range}(d)] \\ \text{xyv} &= [\text{var}(X[i].\text{name}()+\text{str}(2+j)) \\ &\quad \text{for } j \text{ in range}(-2,3) \text{ for } i \text{ in range}(d)] \\ \text{xykv} &= \text{dict}(\text{zip}(\text{xyk}, \text{xyv})) \\ \text{xyvk} &= \text{dict}(\text{zip}(\text{xyv}, \text{xyk})) \\ \text{xn} &= X[0](n).\text{subs}(\text{xykv}) \end{aligned}$$

$$(n = x_2) \quad (12)$$

### 3.2 Quantum Force

The Schiff and Poirier expression for the quantum force is the last term on the left of (1). It can be rewritten as terms of difference operations as follows

```

R = fun(n) (vector([ (hbar^2/4/m) * sum( sum( sum( sum(
    DP( K(n)[k,i] * K(n)[p,j] * DP( DM( K(n)[l,j],C[l] ),C[k] ),C[p] )
    for p in range(d)) for k in range(d)) for j in range(d)) for l in range(d))
    for i in range(d))]) )

```

$$-\frac{\hbar^2}{2m|J|_{n+1}^3} + \frac{\hbar^2}{4m|J|_{n+2}|J|_{n+1}^2} + \frac{\hbar^2}{2m|J|_n^3} - \frac{\hbar^2}{4m|J|_{n+1}|J|_n^2} - \frac{\hbar^2}{4m|J|_{n-1}|J|_n^2} + \frac{\hbar^2}{4m|J|_{n+1}^2|J|_n} \quad (13)$$

We can compute the discretized quantum force as the negative gradient of the discretized quantum potential symbolically. We would like to compare this with the discretization of Poirier's expression for the force.

To get the corresponding expression for the force we sum the potential symbolically over a sufficiently large area and then differentiate by  $x(n)$ . Since the potential is local, involving only nearby worlds, the result is independent of the size of the area so long as the area is large enough to include all the neighbors of a given world at  $n$ .

Gradients:

```

r = fun(n) (vector([ -sum(Q(n+i) for i in range(-2,3)
    ).subs(dets).subs(xylv).diff(xn).expand()
    for k in range(d))])

```

$$\left( -\frac{\hbar^2}{2m(x_{n+1}-x_n)^3} + \frac{\hbar^2}{4m(x_{n+2}-x_{n+1})(x_{n+1}-x_n)^2} \right. \\ \left. - \frac{\hbar^2}{2m(x_{n-1}-x_n)^3} - \frac{\hbar^2}{4m(x_{n+1}-x_n)(x_{n-1}-x_n)^2} \right. \\ \left. - \frac{\hbar^2}{4m(x_{n-1}-x_{n-2})(x_{n-1}-x_n)^2} - \frac{\hbar^2}{4m(x_{n+1}-x_n)^2(x_{n-1}-x_n)} \right) \quad (14)$$

with  $r_2$  and  $r_3$  the gradients of the 2nd and 3rd order terms of the potential, separately.

```

r2 = fun(n) (vector([ -sum(Q2(n+i) for i in range(-2,3)
    ).subs(dets).subs(xylv).diff(xn).expand()
    for k in range(d))])

```

$$\left( \frac{\hbar^2}{2m(x_{n+1}-x_n)^3} - \frac{\hbar^2}{4m(x_{n+2}-x_{n+1})(x_{n+1}-x_n)^2} \right. \\ \left. + \frac{\hbar^2}{2m(x_{n-1}-x_n)^3} + \frac{\hbar^2}{4m(x_{n+1}-x_n)(x_{n-1}-x_n)^2} \right. \\ \left. + \frac{\hbar^2}{4m(x_{n-1}-x_{n-2})(x_{n-1}-x_n)^2} + \frac{\hbar^2}{4m(x_{n+1}-x_n)^2(x_{n-1}-x_n)} \right) \quad (15)$$

```

r3 = fun(n) (vector([ -sum([Q3(n+i) for i in range(-2,3)]
    ).subs(dets).subs(xylv).diff(xn).expand()
    for k in range(d))])

```

$$\left( \begin{aligned} & -\frac{\hbar^2}{m(x_{n+1}-x_n)^3} + \frac{\hbar^2}{2m(x_{n+2}-x_{n+1})(x_{n+1}-x_n)^2} \\ & -\frac{\hbar^2}{m(x_{n-1}-x_n)^3} - \frac{\hbar^2}{2m(x_{n+1}-x_n)(x_{n-1}-x_n)^2} \\ & -\frac{\hbar^2}{2m(x_{n-1}-x_{n-2})(x_{n-1}-x_n)^2} - \frac{\hbar^2}{2m(x_{n+1}-x_n)^2(x_{n-1}-x_n)} \end{aligned} \right) \quad (16)$$

Of course  $r2(n) + r3(n) = r(n)$  is True.

Surprisingly, the force term from the 3rd order term is exactly -2 times the force term obtained from the 2nd order term so the difference between the force from just the 2nd order term so the force from the full expression for the quantum potential differs from the 2nd term by only a sign.

It is True that  $r3(n) = -2 r2(n)$ .

This result is the same as the expression we obtained directly by substituting difference operators for derivatives in Poirier's expression eq. (18) for the quantum force. We will see later that this is not the case in more than one dimension.

It is True that  $r(n) == R(n)$

Hall's equation (3) can be written

$$\begin{aligned} \text{sigma} &= \text{fun}(n) \left( \left( \frac{1}{X[0](n)} - X[0](n-1) \right)^2 * \right. \\ & \quad \left. \left( \frac{1}{X[0](n+1)} - X[0](n) \right) - \frac{2}{X[0](n)} - X[0](n-1) + \frac{1}{X[0](n-1)} - X[0](n-2) \right) \right) \\ \text{r\_hall} &= \text{fun}(n) \left( \hbar^2/4/m * (\text{sigma}(n+1) - \text{sigma}(n)) \right) \end{aligned}$$

$$\begin{aligned} & -\frac{\hbar^2}{2m(x_{n+1}-x_n)^3} + \frac{\hbar^2}{4m(x_{n+2}-x_{n+1})(x_{n+1}-x_n)^2} \\ & -\frac{\hbar^2}{2m(x_{n-1}-x_n)^3} - \frac{\hbar^2}{4m(x_{n+1}-x_n)(x_{n-1}-x_n)^2} \\ & -\frac{\hbar^2}{4m(x_{n-1}-x_{n-2})(x_{n-1}-x_n)^2} - \frac{\hbar^2}{4m(x_{n+1}-x_n)^2(x_{n-1}-x_n)} \end{aligned} \quad (17)$$

It is True that  $rr = r_{hall}$ .

Thus Hall's model of quantum mechanics as many interacting interacting classical worlds in one dimension follows directly from the approximation of Poirier's quantum mechanical equations of motion as difference equations with a certain choice of operators. Obviously this is not the only way to approximate Poirier's equations as difference equations.

Save these calculations for later.

$$\begin{aligned} \text{Q\_hall} &= \text{Q}(n).\text{subs}(\text{dets}) \\ \text{R\_hall} &= \text{R}(n).\text{subs}(\text{dets}) \\ \text{xyv\_hall} &= \text{xyv} \\ \text{xykv\_hall} &= \text{xykv} \end{aligned}$$

## 4 Symmetric Difference Approximation

The central difference approximation to this derivative is

```
def D(x,n):return x.subs(n==n+1)/2-x.subs(n==n-1)/2
```

In this approximation we will see that each world interacts with its four neighbors to the left and right. In fact we may continue this process of approximation to higher orders with more neighbors but in all these Hall-like models this non-locality in phase space is always bounded. Perhaps most interesting from the point of view of efficient computation is the lowest order approximation given previously. But the surprise is that all of these approximations gives quantum mechanics in the limit of a very large number of worlds (i.e. a continuum) and are good approximations even for a very small number of worlds.

Repeating the computations above using a symmetric difference approximation to the derivatives we again find that the quantum force computed as the gradient of the discretized quantum potential is identical to the discretization of Poirier's expression for the force.

```
J = fun(n) (matrix([[D(E[j],C[i]) for i in range(d)] for j in range(d)]))
```

$$\left( \frac{1}{2} x_{n+1} - \frac{1}{2} x_{n-1} \right) \quad (18)$$

```
A = fun(n) (fix(J(n).adjoint()))
K = fun(n) (A(n)/J_(n))
```

$$\left( \frac{1}{|J|_n} \right) \quad (19)$$

```
_J = fun(n) (fix(J(n).determinant().expand()))
dets = {J_(n+i):_J(n+i) for i in range(-6,7)}
```

$$\frac{1}{2} x_{n+1} - \frac{1}{2} x_{n-1} \quad (20)$$

```
Q = fun(n) ( -(hbar^2/4/m) *
    sum( sum( sum(
        (K(n)[k,j] * D( D( K(n)[l,j],C[l] ),C[k] ) +
        1/2 * D(K(n)[l,j],C[l]) * D(K(n)[k,j],C[k]))
        for k in range(d)) for j in range(d)) for l in range(d)) )
Q2 = fun(n) ( -(hbar^2/4/m) *
    sum( sum( sum( (1/2 * D(K(n)[l,j],C[l]) * D(K(n)[k,j],C[k]))
        for k in range(d)) for j in range(d)) for l in range(d)) )
Q3 = fun(n) ( -(hbar^2/4/m) *
    sum( sum( sum( (K(n)[k,j] * D( D( K(n)[l,j],C[l] ),C[k] ))
        for k in range(d)) for j in range(d)) for l in range(d)) )
```

$$-\frac{\hbar^2}{32m|J|_{n+1}^2} - \frac{\hbar^2}{32m|J|_{n-1}^2} + \frac{\hbar^2}{16m|J|_{n+1}|J|_{n-1}} + \frac{\hbar^2}{8m|J|_n^2} - \frac{\hbar^2}{16m|J|_{n+2}|J|_n} - \frac{\hbar^2}{16m|J|_{n-2}|J|_n} \quad (21)$$

```
R = fun(n) (vector([ (hbar^2/4/m)*sum( sum( sum( sum(
    D( K(n)[k,i] * K(n)[p,j] * D( D( K(n)[l,j],C[l] ),C[k] ),C[p] )
    for p in range(d)) for k in range(d)) for j in range(d)) for l in range(d))
    for i in range(d)])) )
```

$$\begin{aligned}
& -\frac{\hbar^2}{16m|J|_{n+1}^3} + \frac{\hbar^2}{16m|J|_{n-1}^3} - \frac{\hbar^2}{32m|J|_{n+1}|J|_{n-1}^2} \\
& -\frac{\hbar^2}{32mJ(n-3)|J|_{n-1}^2} + \frac{\hbar^2}{32m|J|_{n+1}^2J(n+3)} + \frac{\hbar^2}{32m|J|_{n+1}^2|J|_{n-1}}
\end{aligned} \tag{22}$$

```

xyk = [X[i](n+j) for j in range(-4,5) for i in range(d)]
xyv = [var(X[i].name()+str(5+j)) for j in range(-4,5) for i in range(d)]
xykv = dict(zip(xyk,xyv))
xn = X[0](n).subs(xylv)

```

$$n = x_5 \tag{23}$$

The neighborhood for the symmetric difference operators is larger than previous case.

```

r = fun(n) (vector([-sum(Q(n+i) for i in range(-3,4)
).subs(dets).subs(xylv).diff(xn).expand()
for k in range(d))]))
r2 = fun(n) (vector([-sum(Q2(n+i) for i in range(-3,4)
).subs(dets).subs(xylv).diff(xn).expand()
for k in range(d))]))
r3 = fun(n) (vector([-sum([Q3(n+i) for i in range(-3,4)]
).subs(dets).subs(xylv).diff(xn).expand()
for k in range(d))]))

```

It is True that  $r3(n) = -2 r2(n)$ .

It is True that  $r(n) = R(n)$

This expression is the same as the expression we obtained directly by substituting difference operators for derivatives in Poirier's expression (1) for the quantum force.

Save these calculations for later.

```

Q_poirier = Q(n).subs(dets)
R_poirier = R(n).subs(dets)
xylv_poirier = xylv
xykv_poirier = xykv

```

## 5 Simulation

Using the expressions derived above we can generate efficient numerical functions which we use later in determining the motion of a particle in many worlds simultaneously.

We will use Hall's difference approximation to Poirier's equations above for a numerical simulation of quantum motion in a small number of "parallel" worlds.

For efficiency we expression the following operations in "vectorized" form. First we need the difference operator

```

def dM(x,n):return shift(x,n) - shift(x,n-1)

```



## 5.1 Classical Potential

Poirier's problem involves scattering a proton-mass particle from an Eckart potential slightly less than the particles incident energy.

```
alf = 2.5 ; x0=0 ; B = 0.0024
V_(x) = B /cosh(alf *(x-x0))^2
#V_(x) = 2/cosh(2*x)^2
#V_(x) = 0
V = vectorize(fast_callable( V_(x), vars=[x], domain=RDF))
_ = plot(V,(-3,3),axes_labels=['distance','energy'],axes_labels_size=1,
        legend_label='potential')
```

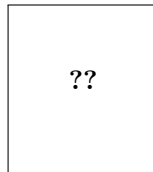


Figure 1: Classical Potential

## 5.2 Classical Force

```
F = vectorize(fast_callable( -diff(V_(x),x), vars=[x], domain=RDF))
_ =plot(F,(-3,3),axes_labels=['distance','force'],axes_labels_size=1,
        legend_label='force')
```

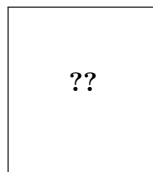


Figure 2: Classical Force

## 5.3 Quantum Potential

```
def qq(hbar,m,x):
    dx0  = dM(x,0)
    dxp1 = dM(x,+1)
    dxm1 = dM(x,-1)
    return hbar^2/4/m*(-1/2/dxp1^2 + 3/2/dx0^2 - 1/dxm1/dx0)
```

1

2

3

4

5

## 5.4 Quantum Force

In general the force is a vector but in the following we treat it as a single scalar quantity.

Vectorized Quantum Force

```

def ff(hbar,m,x):
    dx0 = dM(x,0)
    dxp1 = dM(x,+1)
    dxp2 = dM(x,+2)
    dxm1 = dM(x,-1)
    return hbar^2/2/m*(-1/dxp1^3 + 1/2/dxp2/dxp1^2 + 1/dx0^3 - 1/2/dxp1
        /dx0^2 - 1/2/dxm1/dx0^2 + 1/2/dxp1^2/dx0)

```

## 5.5 Initial Data

### 5.5.1 Spatial Distribution

In order to compute the quantum force on the particles in world  $n$  we introduce four widely separated fictitious particles in worlds to the left and right to represent open boundary conditions.

```

def boundary(x): # returns x as a view over a base with boundary values
    return concatenate((
        array([-1e18,-1e14,-1e10,-1e6]),
        x,
        array([1e6,1e10,1e14,1e18]))) [4:-4]

```

For example consider the spatial distribution of a Gaussian ensemble of one particle in  $N$  “parallel” worlds. Everything is in atomic units below, so  $\hbar = 1$ .

```

N = 2000
var('alf,x0,m,v,x1')
model1={m:RDF(2000),alf:RDF(0.70),v:RDF(0.00164317),x0:RDF(-7.0)}
W(x)=abs((alf/pi)^(1/4)*exp(-(alf/2)*(x-x0)^2)*exp(I*m*v*x))^2

```

(24)

We may compute a uniform mapping into this distribution by inverting the cummulative distribution.

```

assume(alf>0)
erf0=integrate(W(x),x,-oo,x1).subs(model1)
y1=var('y1');inverse_erf=function('inverse_erf')
erfinv0=solve(
    integrate(W(x),x,-oo,x1).subs(model1)==y1,x1)[0].rhs()
def erfinv1(x): return erfinv0.subs(y1==x).substitute_function(
    inverse_erf,erfinv)

```

After adding the open boundary conditions, we can check the accuracy by comparing the discrete representation to the original distribution.

```

p0 = array([erfinv1((i+0.001)/(N-1+0.002)) for i in range(N)], dtype=rdf)
b0 = boundary(p0)
_ = (list_plot([[b0[i],2/N/(b0[i+1]-b0[i-1]]) for i in range(4,len(b0)-4)],
    axes_labels=['location','density'], color='red',
    legend_label='discrete')+
    plot(W(x).subs(model1),(x,-15,5),
    legend_label='continuous'))

```

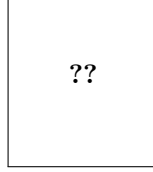


Figure 3: Initial Spatial Density

### 5.5.2 Distribution of Velocity

The particles have an identical initial velocity in each world of ??.

```
v0=array([v.subs(model1) for i in range(N)],dtype=rdf)
```

## 5.6 Energy of the Ensemble

Total classical potential

```
def PU(m,x): return sum(V(i) for i in x)
```

Total kinetic energy

```
def KU(m,v): return RDF(0.5)*m*(v.dot(v))
```

Total quantum potential

```
def QU(m,hbar,x): return qq(m,hbar,x).sum()
```

## 5.7 Acceleration

Acceleration is the combined effect of both classical and quantum forces.

```
def A(hbar,m,x): return (F(x) + ff(hbar,m,x))/m
```

## 5.8 Step Size Controller

Continuous integrating step size controller [12]

```
def G(a,v):
    alpha = RDF(-100.0) # sensitivity
    eps = RDF(10.0^-10)
    return -alpha*(a.dot(v))/max(v.dot(v),eps)
```

1  
2  
3  
4

## 5.9 Integration

Integrate using the Störmer–Verlet algorithm with adaptive step size.

```

t_end = 11600; t_samples = 100; x_samples = 100
t_sample = t_end/t_samples
x_sample = max(1,int(N)/int(x_samples))
x_start = max(0,int(N-x_samples*x_sample)/int(2))
XS = range(x_start,N-x_start-1,x_sample)
ds0 = RDF(0.007) # initial step size
dsn = 1 # initial step size divider
xb = boundary(p0)
while True:
    try: # step size
        ds = ds0/dsn
        rho = RDF(1.0) # step density
        dt = ds/rho
        m0 = RDF(2000.0)
        hbar0 = RDF(1.0)
        # initial values
        x = boundary(p0); v = v0.copy(); a = A(hbar0,m0,x)
        rho = rho - RDF(0.5)*G(a,v)*dt
        t = 0; T = [t]
        XX = {t:x[XS].copy()}; XV = {t:v[XS].copy()}
        KK = KU(m0,v); PP = PU(m0,x); QQ = QU(hbar0,m0,x)
        E1 = KK+PP+QQ # total energy
        Rho = {t:rho}; TK = {t:KK}; TP = {t:PP}; TQ = {t:QQ}; E = {t:E1
    }
    try:
        while t<t_end:
            print "t_=", t, "rho_=", rho
            t1 = t + t_sample
            while t<t1:
                rho = rho + G(a,v)*dt
                dt = ds/rho; t += dt
                x += v*dt + 0.5*a*dt^2
                v += 0.5*a*dt; a = A(hbar0,m0,x); v += 0.5*a*dt
                if isnan(rho) or rho>100 or rho<0.01:
                    raise ValueError(
                        "Step_control_failed_at_%s_rho=%s"%(t,rho)
                    )
                # Check no-crossing
                if (x[1:]-x[:N-1]).min()<0:
                    raise ValueError("crossing")
                # Check total energy conservation
                KK = KU(m0,v); PP = PU(m0,x); QQ = QU(hbar0,m0,x)
                E2 = KK+PP+QQ
                if abs(E1-E2)>0.01:
                    raise ValueError(
                        "Energy_conservation_bound_failed_at_%s_Delta_
                        E:\n
                        "|%s|>0.01"%(t,E1-E2))
                E1 = E2
                T += [t]; XX[t] = x[XS].copy(); XV[t] = v[XS].copy()

```

```

        Rho[t] = rho; TK[t] = KK; TP[t] = PP; TQ[t] = QQ; E[t] = E1 48
        = E1
    except KeyboardInterrupt: 49
        print "Interrupted at %s..."%(t) 50
        T += [t]; XX[t] = x[XS].copy(); XV[t] = v[XS].copy() 51
        Rho[t] = rho; TK[t] = KK; TP[t] = PP; TQ[t] = QQ; E[t] = E1 52
    tmax = t 53
    break 54
except ValueError as msg: 55
    print msg 56
    dsn = dsn + 1 57
    print "Trying a shorter initial step size: %s"%(ds0/dsn) 58
    continue 59
print "t=", tmax, "rho=", rho 60

```

The adaptation of the integration stepsize during the simulation is given by rho. A value greater than 1 indicates a smaller stepsize.

```
_ = line2d([[t,Rho[t]] for t in T],axes_labels=['time','rho'])
```

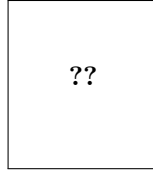


Figure 4: Step Density

Energy is exchanged between classical, quantum mechanical and kinetic forms due to the interaction of the particle with the potential barrier (in each world) as well as the interaction between worlds.

```

_ = (line2d([[t,TK[t]] for t in T], axes_labels=['time','energy'],
    legend_label='kinetic', color='green') +
    line2d([[t,TQ[t]] for t in T], axes_labels=['time','energy'],
    legend_label='quantum potential',color='red') +
    line2d([[t,TP[t]] for t in T], axes_labels=['time','energy'],
    legend_label='classical potential',color='blue'))

```

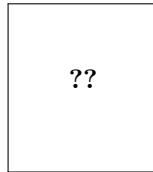


Figure 5: Quantum Potential, Classical Potential and Kinetic Energy

Conservation of energy is an important indicator of the quality of the numerical integration. Note in particular how the changes in total energy over time remain well-bounded.

```

_ = line2d([[t,E[t]] for t in T],axes_labels=['time','energy'],
    legend_label='total energy',color='black')

```

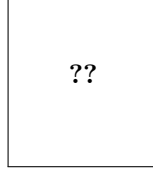


Figure 6: Energy Conservation

The trajectory of the particle in each world is shown below. Notice how the trajectories do not cross, yet in some worlds the particle is scattered from the barrier while in others the particle tunnels across the barrier “pushed” by its counterpart in other worlds.

```
_ = (line([[t,XX[t].mean()]] for t in T], color='black',
      thickness=2, legend_label="average",
      axes_labels=['time','location'], axes_labels_size=1)+
sum(line([[t,XX[t][i]] for t in T], color=hue((0.5*i)/len(XS)), alpha=0.5,
      axes_labels=['time','location'], axes_labels_size=1)
    for i in range(len(XS))))
```

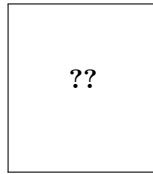


Figure 7: Trajectories

The distribution at the end of the simulation shows the reflected and transmitted “wave packets” separating from each other.

```
_ = list_plot([[x[i-1],2/N/(x[i+1]-x[i-1])]] for i in range(1,N-1) ],
              axes_labels=['location','density'])
```

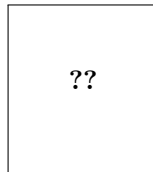


Figure 8: Final Spatial Density

Transmission probability

```
_ = line([[t,(XV[t]>0).mean()]] for t in T ],
#_ = line([[t,(XX[t]>0).mean()]] for t in T ],
          axes_labels=['time','transmission'])
```

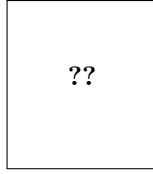


Figure 9: Evolution of Spatial Density

Final ??.

## Appendix A Sage Preliminaries

It is convenient to define a compact notation for functions:

```

import numpy 1
from numpy import array, concatenate, vectorize, isnan 2
rdf=numpy.array([RDF.pi()]).dtype # ensure compatible types 3
import mpmath 4
from mpmath import erfinv 5

6
def fun(*y): # fun(x,y)(f)(x,y)=f 7
    cache={} 8
    def upd(f,x,y): # memoized 9
        try: return cache[x] 10
        except KeyError: # not found 11
            try: cache[x] = f.subs(dict([[y[i],x[i]]
                                         for i in range(len(y))])) 12
            except AttributeError: # ask forgiveness 14
                def sub(f): return f.subs(dict([[y[i],x[i]]
                                                for i in range(len(y))])) 15
                cache[x] = map(sub,f) 16
            return cache[x] 17
    return lambda f: lambda *x: upd(f,x,y) 18

def argscript(self, *args): 19
    return "%s_{%s}"%(self.name(),', '.join(map(repr, args))) 20

def functions(*names): 21
    return map(lambda nam: function(nam, print_latex_func=argscript), 22
              names) 23

def detJscript(self, *args): 24
    return '\J_{%s}'% (' '.join(map(repr, args))) 25

J__= function('J_') # determinant of J 26
J_ = function('J_', print_latex_func=detJscript) 27

def squared(x): # x^2 28
    try: return x.dot(x) # numpy array 29
    except: return power(x,2) 30

```

The following is a work-round for a bug<sup>6</sup> in Sage.

```

def fix(it): 1
    ret = it.subs(dict([[function('x')(n+i), X[0](n+i)] for i in range 2
                        (-2,3)] +
                      [[J__(n+i), J_(n+i)] for i in range(-2,3)])) 3
    return ret 4
def factor(x): 5
    try: return fix(x.factor()) 6
    except: return vector(map(lambda y: fix(y.factor()),x)) 7

```

This routine shifts a numpy array view left or right by  $n$  cells.

```

def shift(x,n): 1
    try: # deduce slicing 2

```

---

<sup>6</sup>Some commands for manipulating and simplifying expressions do not respect the `print_latex_func` option. <https://trac.sagemath.org/ticket/19151>



```
start = (x.__array_interface__['data'][0] -  
         x.base.__array_interface__['data'][0])/x.itemsize  
stop = start + x.shape[0]  
if start+n<0 or stop+n>x.base.shape[0]:  
    raise RuntimeError('bounds')  
return x.base[start+n:stop+n]  
except AttributeError:  
    raise RuntimeError('expecting view')
```

## List of Figures

1	Classical Potential . . . . .	9
2	Classical Force . . . . .	10
3	Initial Spatial Density . . . . .	12
4	Step Density . . . . .	14
5	Quantum Potential, Classical Potential and Kinetic Energy . . . . .	15
6	Energy Conservation . . . . .	15
7	Trajectories . . . . .	15
8	Final Spatial Density . . . . .	16
9	Evolution of Spatial Density . . . . .	16

## References

- [1] E. Madelung, “Quantentheorie in hydrodynamischer Form,” *Zeitschrift fur Physik* **40** (Mar., 1927) 322–326.
- [2] D. Bohm and B. Hiley, *The Undivided Universe: An Ontological Interpretation of Quantum Theory*. Routledge, 1993. <https://books.google.com.au/books?id=QfiHAgAAQBAJ>.
- [3] L. de Broglie, *Non-linear Wave Mechanics: A Causal Interpretation*. Elsevier Pub. Co., 1960. <https://books.google.ca/books?id=RNoIAQAIAAJ>.
- [4] P. Holland, *The Quantum Theory of Motion: An Account of the de Broglie-Bohm Causal Interpretation of Quantum Mechanics*. Cambridge University Press, 1995. <https://books.google.ca/books?id=BsEfVBzToRMC>.
- [5] R. Wyatt, *Quantum Dynamics with Trajectories: Introduction to Quantum Hydrodynamics*. Interdisciplinary Applied Mathematics. Springer New York, 2005. <https://books.google.com.au/books?id=QDjq5SmIREsC>.
- [6] M. J. Hall, D.-A. Deckert, and H. M. Wiseman, “Quantum phenomena modeled by interactions between many classical worlds,” *Physical Review X* **4** no. 4, (2014) 041013. <http://link.aps.org/doi/10.1103/PhysRevX.4.041013>.
- [7] J. Schiff and B. Poirier, “Communication: Quantum mechanics without wavefunctions,” *The Journal of chemical physics* **136** no. 3, (Jan., 2012) 031102, arXiv:1201.2382 [quant-ph]. <http://scitation.aip.org/content/aip/journal/jcp/136/3/10.1063/1.3680558>.
- [8] Various, “SageMath, the mathematical software system,” 2016. <http://www.sagemath.org>. [Online; accessed 2016-07-02].
- [9] W. Stein, H. Schilly, *et al.*, “SageMathCloud, a web-based cloud computing and course management platform for computational mathematics,” 2016. <http://cloud.sagemath.com>. [Online; accessed 2016-07-02].
- [10] E. Jones, T. Oliphant, P. Peterson, *et al.*, “SciPy: Open source scientific tools for Python,” 2016. <http://www.scipy.org/>. [Online; accessed 2016-07-02].
- [11] F. Johansson *et al.*, *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.19)*, December, 2016. <http://mpmath.org/>. [Online; accessed 2016-07-02].

- [12] E. Hairer and G. Söderlind, “Explicit, time reversible, adaptive step size control,” *SIAM Journal on Scientific Computing* **26** no. 6, (2005) 1838–1851.