Hall1Dinterference

Bill Page

8/19/2015

Contents

1 From Poiriers Bohmian Mechanics without Wavefunctions To Halls Many Interacting Worlds

Ref:

1. Quantum Phenomena Modeled by Interactions between Many Classical Worlds Michael J. W. Hall Dirk-André Deckert and Howard M. Wiseman, PHYSICAL REVIEW X 4, 041013 (23 October 2014) http://arxiv.org/abs/1402.6144

2. Quantum Mechanics Without Wavefunctions Jeremy Schiff and Bill Poirier J. Chem. Phys. 136, 031102 (2012) http://arxiv.org/abs/1201.2382v1

3. `Verletintegration(Wikipedia)`

```
%typeset_mode True
from numpy import array, concatenate
from mpmath import erfinv
```

1.1 1-D

Comparison of Poiriers many-D expression to Halls 1-D toy expression.

```
vars= ['x']; d = 1
X = map(function,vars)
# world index
%var n
ind=[n]
# position of particle in world n
P=map(lambda x:x(*ind),X);P
```

$$[x(n)]$$

1.1.1 Difference operators

For Halls finite number of interacting worlds we need the following difference operators instead of derivatives.

```
def Dminus(x,i):return(x-x.subs(ind[i]==ind[i]-1))
def Dplus(x,i):return(x.subs(ind[i]==ind[i]+1)-x)
```

```
Dminus(P[0],0)
Dminus(Dminus(P[0],0),0)
Dplus(Dminus(Dminus(P[0],0),0),0)
Dplus(Dplus(Dminus(Dminus(P[0],0),0),0),0)
```

$$-x(n-1)+x(n)$$

$$-2\,x(n-1)+x(n-2)+x(n)$$

$$x(n+1)+3\,x(n-1)-x(n-2)-3\,x(n)$$

$$x(n+2)-4\,x(n+1)-4\,x(n-1)+x(n-2)+6\,x(n)$$

etc.

1.1.2 Halls toy expression

eqs. (24,25)

```
%var hbar,mm
```

```
def sif(n): return(1/((X[0](n))-(X[0](n-1))))
def sigma(n): return((sif(n)^2)*(sif(n+1)-2*sif(n)+sif(n-1)))
def r(n): return(hbar^2/4/mm*(sigma(n+1)-sigma(n)))
def U(n): return(hbar^2/8/mm*(sif(n+1)-sif(n))^2)
```

1.1.3 Poiriers expression

```
# Ensemble of Bohmian trajectories
En = map(lambda x:x(*ind),X);En
# Jacobian: How trajectories change with trajectory index (i.e. as a \
  function of the initial position).
```

```
Jn = matrix(map(lambda e:map(lambda c:Dminus(e,c),[i for i in range(d)]),\
    En));Jn
Kn = Jn^(-1);Kn
bool(sif(n)==Kn[0,0])
```

$$[x(n)]$$

$$\left(-x(n-1) + x(n) \right)$$

$$\left(-\frac{1}{x(n-1)-x(n)} \right)$$

True

1.1.4 Quantum Force

Schiff and Poirier eq. (18)

```
eq18n = [ sum([ sum([ sum([ sum([
 (hbar^2/4/mm)*Dplus( Kn[k,i]*Kn[p,j]*Dplus( Dminus( Kn[l,j],k ),l ),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)]
eq18n[0]
```

$$\frac{hbar^2\left(\frac{\frac{1}{x(n+2)-x(n+1)}-\frac{2}{x(n+1)-x(n)}-\frac{1}{x(n-1)-x(n)}}{(x(n+1)-x(n))^2}-\frac{\frac{1}{x(n+1)-x(n)}+\frac{1}{x(n-1)-x(n-2)}+\frac{2}{x(n-1)-x(n)}}{(x(n-1)-x(n))^2}\right)}{4\,mm}$$

Compare to Halls MIW quantum force

```
bool(eq18n[0]==r(n))
```

True

1.1.5 Fast Numeric Functions

Create a numeric function that depends on the nearest neighbors: x(n-2) x(n-1) x(n) x(n+1) x(n+2)

```
xsk=[X[i](n+j) for i in range(d) for j in [-2,-1,0,1,2]];xsk
xsv=[var(vars[i]+str(2+j)) for i in range(d) for j in [-2,-1,0,1,2]]; xsv
```

$$[x(n-2), x(n-1), x(n), x(n+1), x(n+2)]$$

$$[x_0, x_1, x_2, x_3, x_4]$$

```
eq18n3=map(lambda ex: ex.subs(dict(zip(xsk,xsv))),eq18n); eq18n3
```

$$\left[\frac{hbar^2\left(\frac{\frac{1}{x_0-x_1}-\frac{2}{x_1-x_2}+\frac{1}{x_2-x_3}}{(x_1-x_2)^2}-\frac{\frac{1}{x_1-x_2}-\frac{2}{x_2-x_3}+\frac{1}{x_3-x_4}}{(x_2-x_3)^2}\right)}{4\,mm}\right]$$

```
rf=fast_callable(eq18n3[0].subs({hbar:2/10,mm:1}), vars=xsv, domain=float\
   )
```

```
p1=[random() for i in xsv];p1
```

$$[0.600471360389, 0.755886986663, 0.605153493182, 0.0762442866092, 0.934690481751]$$

```
rf(*p1)
```

$$-7.89998173363$$

```
%timeit [rf(*p1)]
625 loops, best of 3: 373 ns per loop
```

1.1.6 Quantum Potential

Why does Poirier eq. (20) include a second term?

```
eq20n = sum([ sum([ sum([
 (hbar^2/4/mm)*(Kn[k,j]*Dplus( Dplus( Kn[l,j],l ),k )+1/2*Dplus(Kn[l,j],l\
    )*Dplus(Kn[k,j],k))
   for k in range(d)]) for j in range(d)]) for l in range(d)])
eq20n
```

$$\frac{\left(\left(\frac{1}{x(n+1)-x(n)}+\frac{1}{x(n-1)-x(n)}\right)^2-\frac{2\left(\frac{1}{x(n+2)-x(n+1)}-\frac{2}{x(n+1)-x(n)}-\frac{1}{x(n-1)-x(n)}\right)}{x(n-1)-x(n)}\right)hbar^2}{8\,mm}$$

Only the first term corresponds to Halls expression.

```
U(n)
```

$$\frac{hbar^2\left(\frac{1}{x(n+1)-x(n)}+\frac{1}{x(n-1)-x(n)}\right)^2}{8\,mm}$$

```
bool(eq20n==U(n))
```

$$\text{False}$$

```
eq20n3=eq20n.subs(dict(zip(xsk,xsv))); eq20n3
```

$$\frac{\left(\left(\frac{1}{x_1-x_2}-\frac{1}{x_2-x_3}\right)^2+\frac{2\left(\frac{1}{x_1-x_2}-\frac{2}{x_2-x_3}+\frac{1}{x_3-x_4}\right)}{x_1-x_2}\right)hbar^2}{8\,mm}$$

```
ru=fast_callable(eq20n3.subs({hbar:2/10,mm:1}), vars=xsv, domain=float)
```

```
ru(*p1)
```

$$0.224489215087$$

Computation of total energy below shows that Halls expression is correct.

```
eq20n4=U(n).subs(dict(zip(xsk,xsv))); eq20n4
```

$$\frac{hbar^2\left(\frac{1}{x_1-x_2}-\frac{1}{x_2-x_3}\right)^2}{8\,mm}$$

```
rU=fast_callable(eq20n4.subs({hbar:2/10,mm:1}), vars=xsv, domain=float)
```

```
rU(*p1)
```

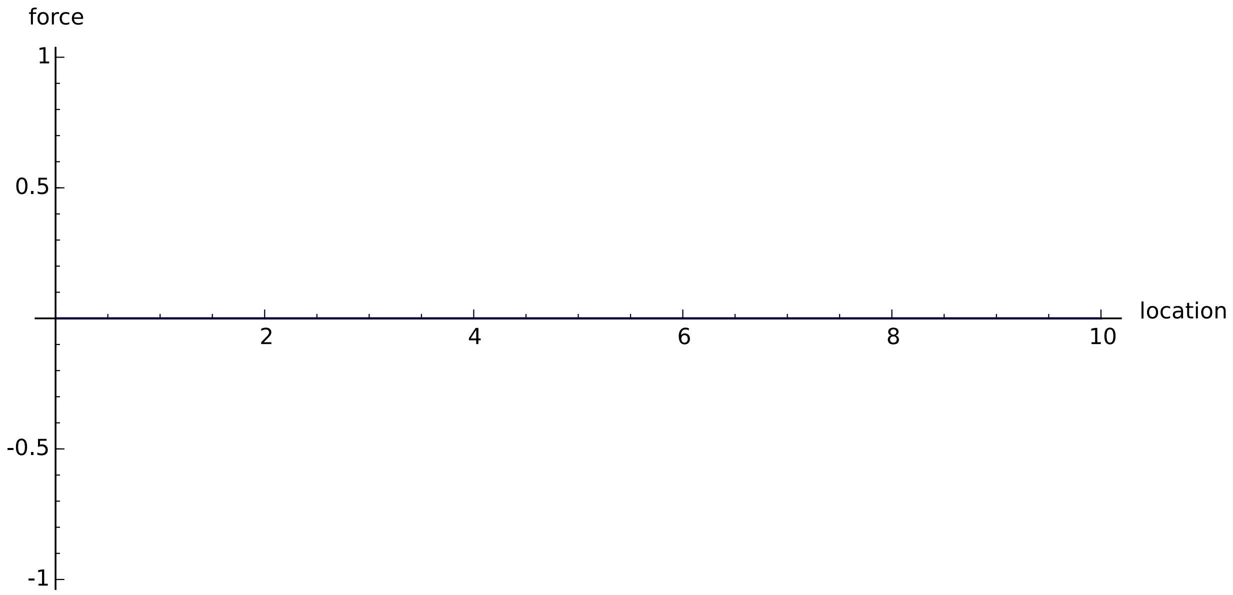
0.112505949801

1.1.7 Classical force

```
a = 0.25
c = 5
#F(x)=((x-c)/(a^3*(3.14)))*exp(-(x-c)^2/a^2)*0.7
#F(x)= 9*sech(5*(x-c))^2*tanh(5*(x-c))
F(x)=0
```

```
FF=fast_callable(F(x), vars=[x], domain=float)
```

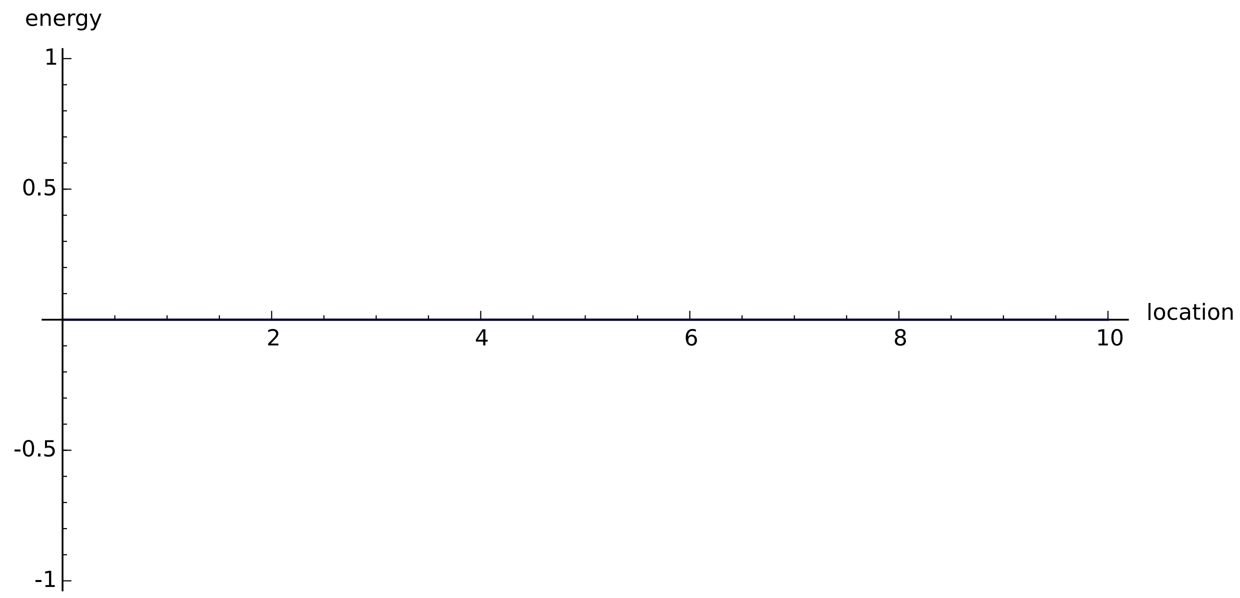
```
plot(FF,(0,10),axes_labels=['location','force'],axes_labels_size=1)
```



1.1.8 Classical potential

```
#PO=sage_eval("lambda x:"+fricas(-F(x)).integrate(x).simplify().\
   unparsed_input_form());PO(x)
#PO(x)=9/(10*cosh(5*x+-5*c)^2)
PO(x)=0
```

```
plot(PO,(0,10),axes_labels=['location','energy'],axes_labels_size=1)
```



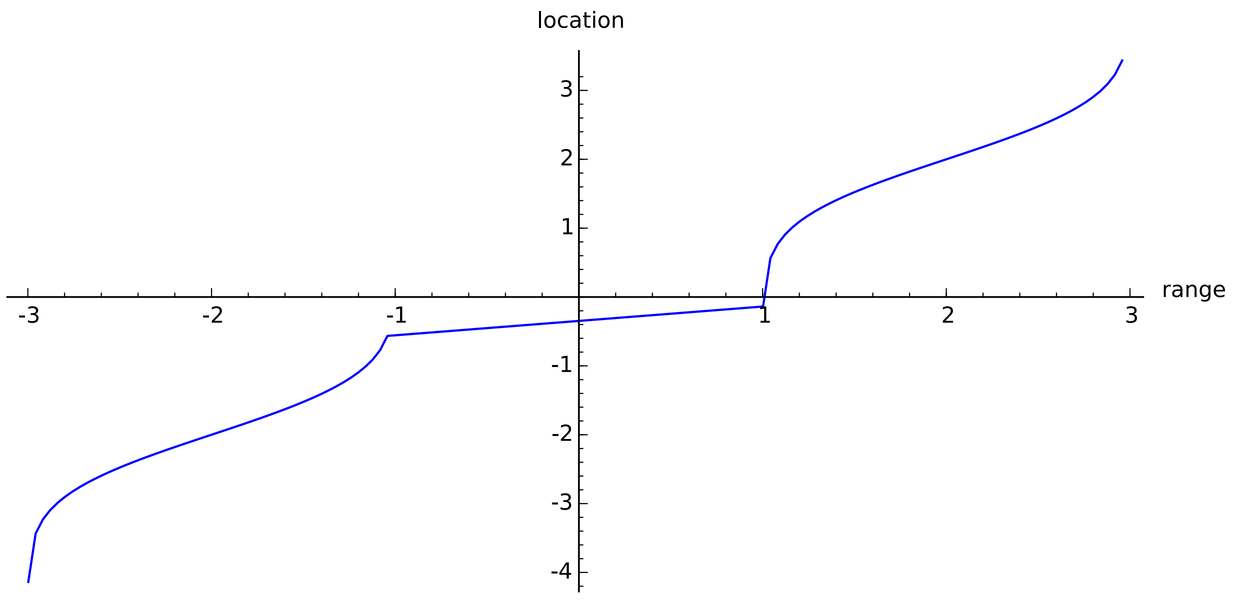
1.2 Simulation

Simulation of a single particle in one dimension over 100 worlds. The initial spacial distribution is a Gaussian wave packet. The uniformizing function is the cummulative distribution, this case the error function. We can use the inverse of the uniformizing function to create a gaussian packet from a uniform range.

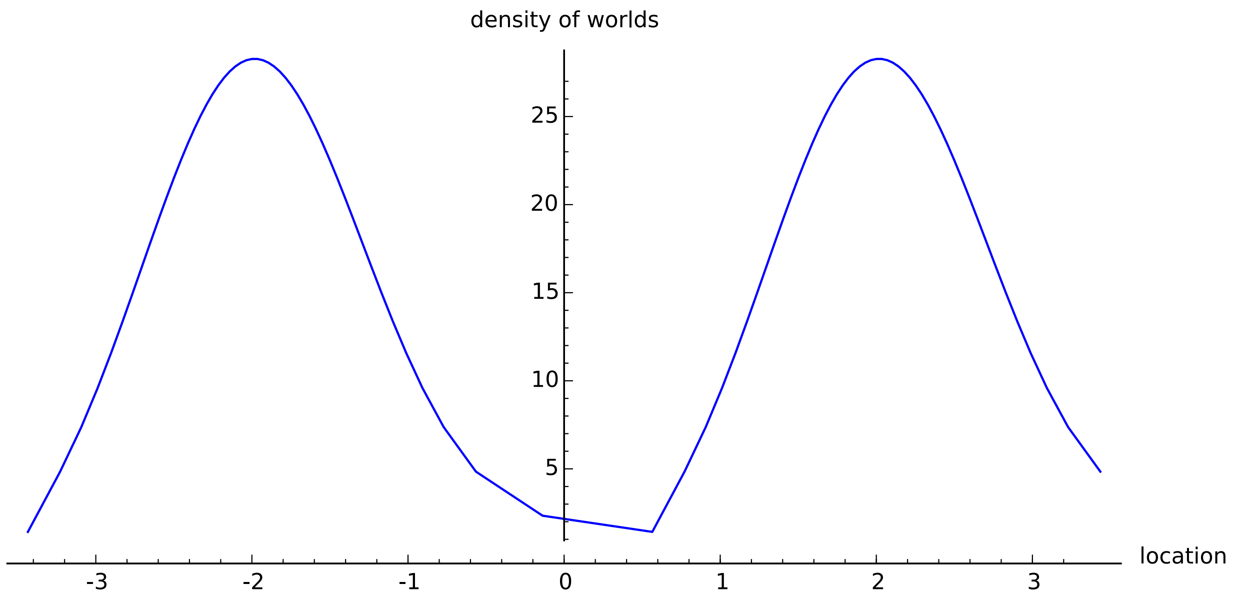
1.2.1 Initial conditions

```
N=50
r=[1.995*i/N for i in range(-N/2,(N+1)/2)]
r0=map(lambda x:x-2,r)+map(lambda x:x+2,r)
p=map(lambda x:float(1.0*erfinv(x)),r)
p0=map(lambda x:x-2,p)+map(lambda x:x+2,p);
N=len(p0)
```

```
line2d(zip(r0,p0),axes_labels=['range','location'],axes_labels_size=1)
```



```
# density
import numpy
def den(p): return map(lambda x:1/x,p[1:]-p[0:N-1])
line2d(zip(p0[1:N],den(array(p0))),axes_labels=['location','density of \
   worlds'],axes_labels_size=1)
```



ensembles are approaching each other

```
# initial velocity of particle in each world
v0=[0.0 for i in p]+[0.0 for i in p]
```

1.2.2 Acceleration

```
# (quantum force + classical force)/mass
def rf1(X,i):
    global N
    lm2=X[i-2] if i>1 else -(10.0^10)
    lm1=X[i-1] if i>0 else -(10.0^6)
    rp1=X[i+1] if i+1<N else 10.0^6
    rp2=X[i+2] if i+2<N else 10.0^10
    return rf(lm2,lm1,X[i],rp1,rp2)+FF(X[i])
```

Total classical potential

```
def PU(x): return sum([PO(x[i]) for i in range(N)])
```

Total quantum Potential

```
def ru1(X,i):
    global N
    lm2=X[i-2] if i>1 else -(10.0^10)
    lm1=X[i-1] if i>0 else -(10.0^6)
    rp1=X[i+1] if i+1<N else 10.0^6
    rp2=X[i+2] if i+2<N else 10.0^10
    # (quantum potential
    return rU(lm2,lm1,X[i],rp1,rp2)
def QU(x): return sum([ru1(x,i) for i in range(N)])
```

Total kinetic energy

```
def K(x): return sum(x^2)/2
```

1.3 Solution

Run the integration

1.3.1 Velocity Verlet

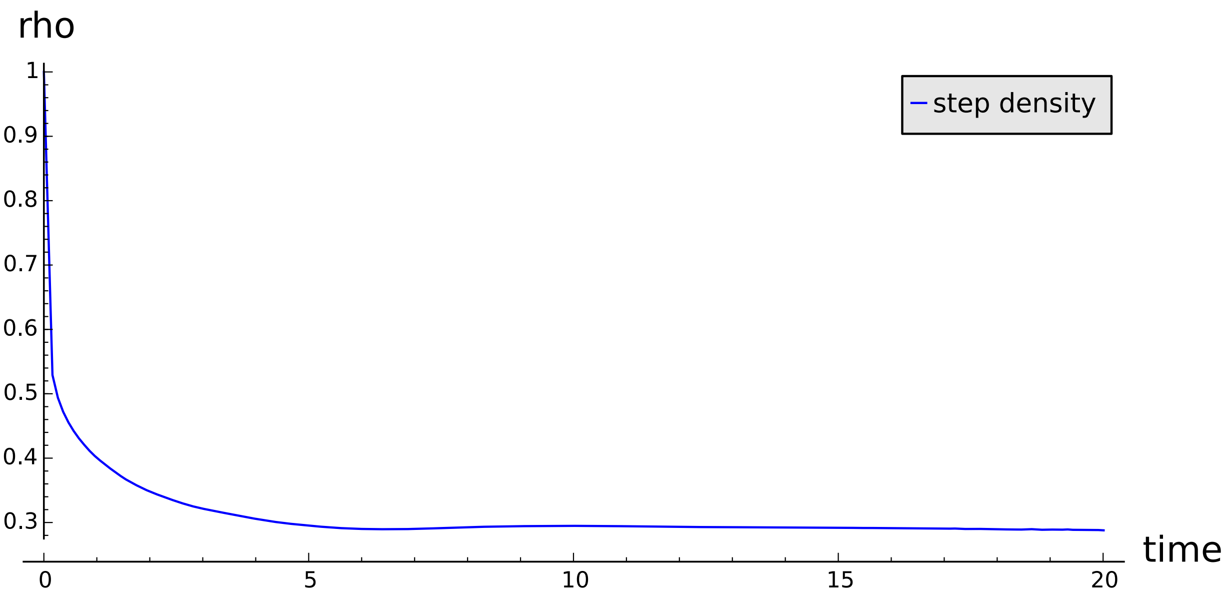
Störmer–Verlet discretization with continuous integrating stepsize controller
Ref: `Explicit,TimeReversible,AdaptiveStepSizeControl` by Ernst Hairer and Gustaf Söderlind

```
%time
t1 = 20; step = 0.1;
def A(x): return array([rf1(x,i) for i in range(N)])
XX = array(p0)
VV = array(v0)
AX = A(XX)
# minimum step size
ds = 0.002
# controller
def G(p,q): return -alpha*sum(map(prod,zip(p,q)))/(eps+sum(map(prod,zip(q\
   ,q))))+beta*(1-ds/dt)
alpha = 0.1
```

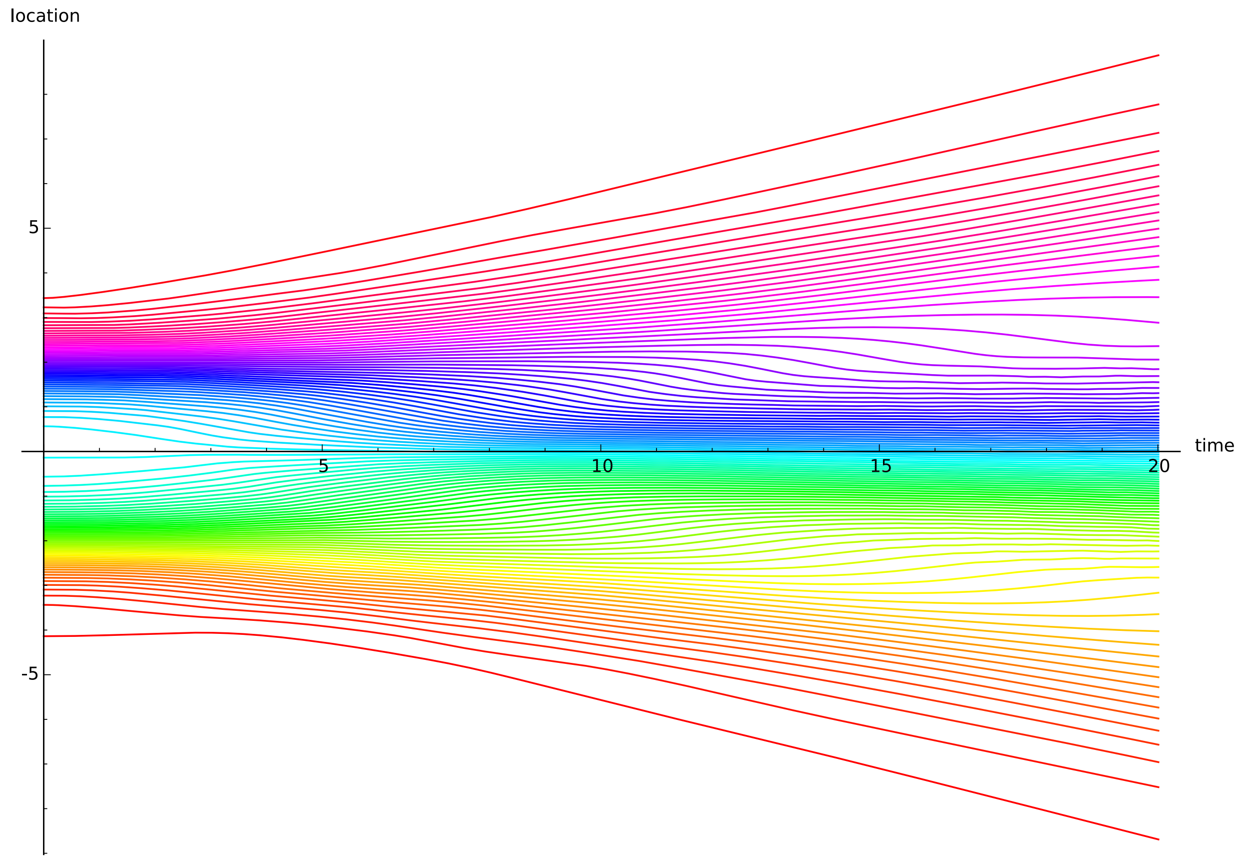
```
beta = 0.0
eps = 10^-10
# step density
rho = 1; dt=ds/rho; rho = rho - G(AX,VV)*dt/2
t = 0
XV = [concatenate((XX,VV))]; tt = [t]
rhos = [rho]
while t<t1:
    # Check total energy conservation
    T1=K(VV)+PU(XX)+QU(XX)
    for i in range(int(step/dt)):
        rho = rho + G(AX,VV)*dt
        dt = ds/rho; t += dt
        XX += VV*dt + AX/2*dt^2
        VV += AX/2*dt; AX = A(XX); VV += AX/2*dt
    T2=K(VV)+PU(XX)+QU(XX)
    if abs(T1-T2)>0.05:
        print "Integration failed at %s."%t,"Try a shorter time step: %s.\
            "%ds/2
        break
    XV.append(concatenate((XX,VV))); tt += [t]; rhos.append(rho)
CPU time: 4.97 s, Wall time: 4.98 s
```

```
len(rhos)
line2d(zip(tt,rhos),axes_labels=['time','rho'],legend_label='step density\
    ')
```

208



```
show(sum([line([[tt[i],XV[i][j]] for i in range(0,len(tt))],rgbcolor = \
    hue(float(j)/N))
  for j in range(N)]),figsize=[10,7],axes_labels=['time','location'],\
      axes_labels_size=1)
```



```
animate([plot(P0,(-10,10),axes_labels=['location','classical potential'],\
    axes_labels_size=1,
  figsize=[10,4]) + text("time:"+("%4.1f"%X[0]).rjust(5), (10,1),color='\
      black') +
  list_plot(zip(X[1][0:N],map(P0,X[1][0:N])), size=15, xmin=-10, xmax=10,\
      ymax=1)
    for X in zip(tt,XV)]).show(gif=true,delay=20)
```

https://cloud.sagemath.com/blobs/tmp_4yUqrb.gif?uuid=09a7ccca-0902-482b-8b1a-9c969e0a69c0

```
animate([line2d(zip(X[1][0:N],den(X[1][0:N])),xmin=-10, xmax=10,ymin=0,\
    ymax=35,
  axes_labels=['location','density'],axes_labels_size=1)+
    text("time:"+("%4.1f"%X[0]).rjust(5), (10,35),color='black')
      for X in zip(tt,XV)]).show(gif=true,delay=20)
```

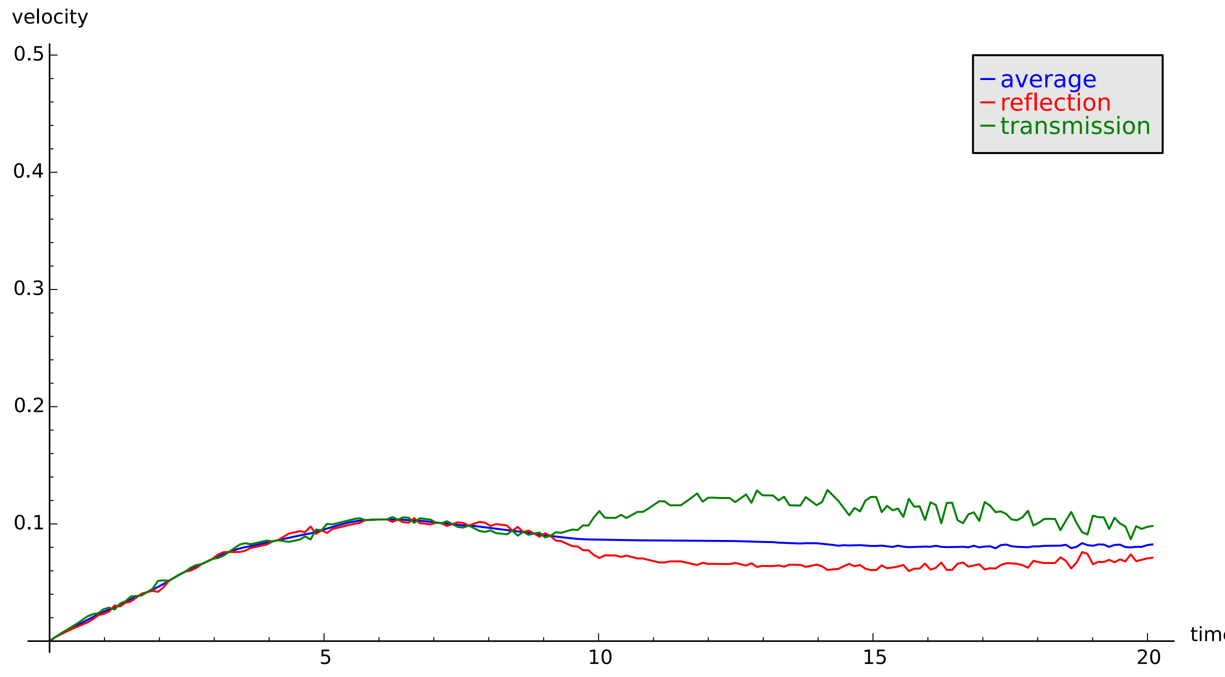
https://cloud.sagemath.com/blobs/tmp_VVYq2H.gif?uuid=eb6ee50b-f133-4e94-bcc7-39d21ccc9ba4

```
# momentum density
animate([line2d(zip(mu[1],den(mu[1])),xmin=-3,xmax=3,ymin=0,ymax=10^3,
  axes_labels=['velocity','density'],axes_labels_size=1)+
  text("time:"+("%4.1f"%mu[0]).rjust(5), (3,1000),color='black')
  for mu in zip(tt,map(lambda x: x[N:2*N].sort() or x[N:2*N],XV))]).show(\
      gif=true,delay=20)
```

https://cloud.sagemath.com/blobs/tmp_Qe3Lii.gif?uuid=5231dd94-eada-4344-84bc-57ee7b2d3436

1.3.2 Velocity

```
def average_velocity(t): return sum(map(abs,XV[t][N:2*N]))/N
def num_left(t): return sum(map(lambda x:1 if x<0 else 0,XV[t][N:2*N]))
def average_left(t): return sum(map(lambda x:-x if x<0 else 0,XV[t][N:2*N\
    ]))/(0.000001+num_left(t))
def num_right(t): return sum(map(lambda x:1 if x>=0 else 0,XV[t][N:2*N]))
def average_right(t): return sum(map(lambda x: x if x>0 else 0,XV[t][N:2*\
    N]))/(0.000001+num_right(t))
tot=line2d([[tt[i],average_velocity(i)] for i in range(len(tt))],color='\
    blue',legend_label='average',legend_color='blue',ymax=0.5)
left=line2d([[tt[i],average_left(i)] for i in range(len(tt))],color='red'\
    ,legend_label='reflection',legend_color='red',ymax=0.5)
right=line2d([[tt[i],average_right(i)] for i in range(len(tt))],color='\
    green',legend_label='transmission',legend_color='green',ymax=0.5)
show(tot+left+right,figsize=[9,5],axes_labels=['time','velocity'],\
    axes_labels_size=1)
```



```
sum(map(abs,XV[len(tt)-1][N:2*N]))-sum(map(abs,XV[0][N:2*N]))
```

8.1831853166

Final state

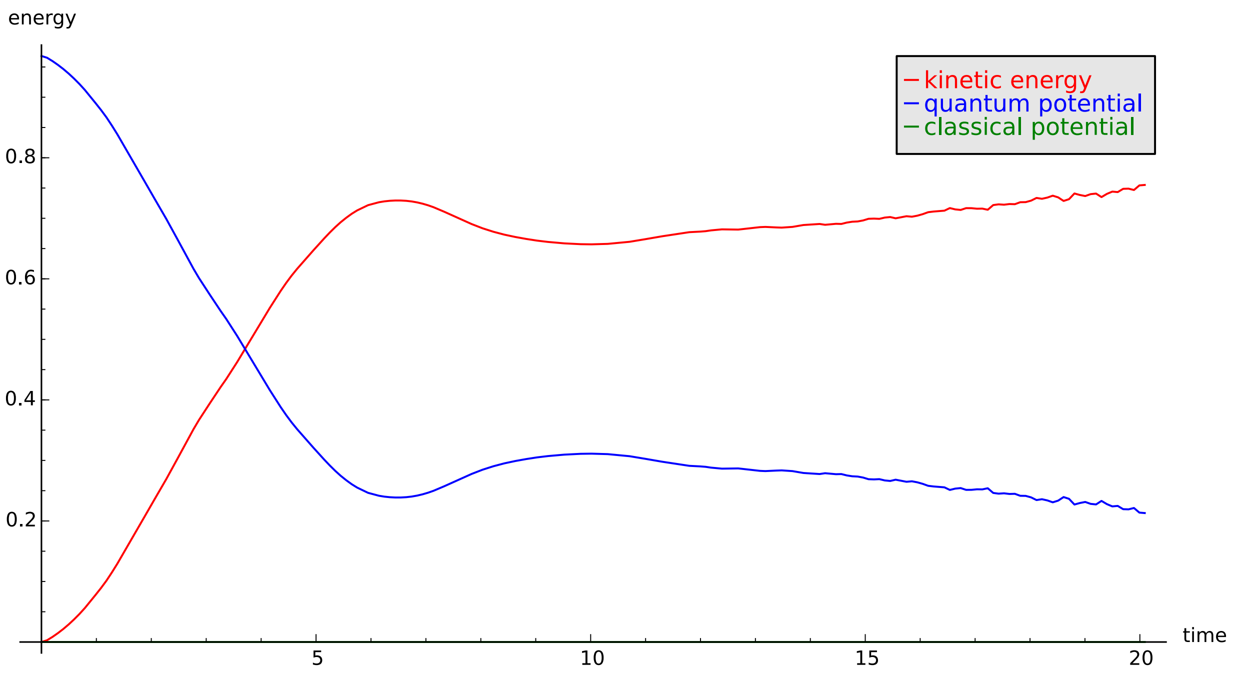
```
print "# of worlds with particles moving left: ", num_left(len(tt)-1), "\
    average velocity: ",average_left(len(tt)-1)
```

```
print "# of worlds with particles moving right: ", num_right(len(tt)-1), \
   "average velocity: ",average_right(len(tt)-1)
print "overall average: ",average_velocity(len(tt)-1)
```

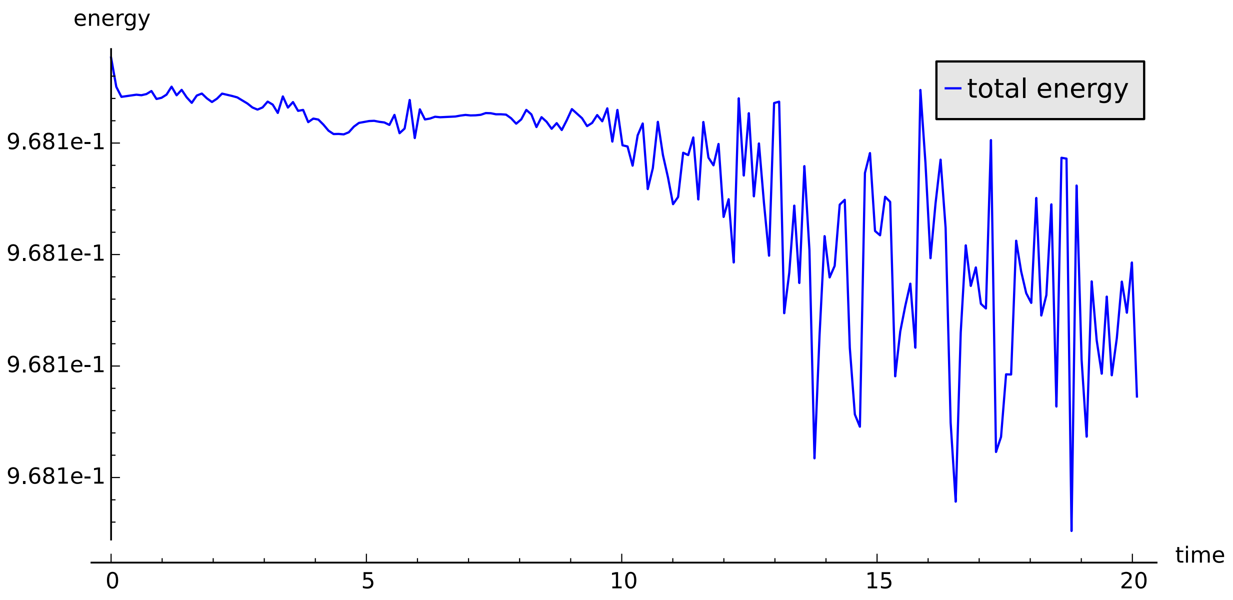
```
# of worlds with particles moving left:  56 average velocity:  0.0739570105298
# of worlds with particles moving right:  44 average velocity:  0.0941271036102
overall average:  0.082831853166
```

1.3.3 Conservation of energy

```
U1=line2d([[tt[i],QU(XV[i][0:N])] for i in range(len(tt))],
  color='blue',legend_color='blue',legend_label="quantum potential")
U2=line2d([[tt[i],PU(XV[i][0:N])] for i in range(len(tt))],
  color='green', legend_color='green', legend_label="classical potential"\
     )
K1=line2d([[tt[i],K(XV[i][N:2*N])] for i in range(len(tt))],
  legend_color='red',color='red',legend_label="kinetic energy")
show(K1+U1+U2,figsize=[9,5],axes_labels=['time','energy'],\
   axes_labels_size=1)
```



```
line([[tt[i],K(XV[i][N:2*N])+PU(XV[i][0:N])+QU(XV[i][0:N])] for i in \
   range(len(tt))],
  legend_label="total energy",axes_labels=['time','energy'],\
     axes_labels_size=1)
```



```
K(XV[len(tt)-1][N:2*N])+QU(XV[len(tt)-1][0:N])+PU(XV[len(tt)-1][0:N])-K(\
    XV[1][N:2*N])-QU(XV[1][0:N])-PU(XV[1][0:N])
```

$$-2.77788740755 \times 10^{-06}$$