Variational Monte Carlo to find Ground State Energy for Helium

Chris Dopilka

December 2, 2011

1 Introduction[1][2]

The variational principle from quantum mechanics gives us a way to estimate the ground state energy of complicated potentials since we don't have to actually know the exact form the of wave function for the ground state. The variational principle states that for an arbitrary normalized wavefunction, ψ ,

$$E_{\text{gs}} \leq \langle \psi | H | \psi \rangle$$

that is the expectation value of H will overestimate the ground state energy. So by guessing the form of the wavefunction and varying the wavefunction, you can look at the minimum of the expectation value of H of the varying wavefunction, and conclude that it must be closest to the ground state energy.

Computationally, this reduces the problem to finding the expectation value of H for some wavefunction with an adjustable parameter. In other words, we need to evaluate the integral:

$$\langle \psi | H | \psi \rangle = \int \psi^* H \psi \, dr$$

The easiest way to evaluate integrals computationally, over any number of dimensions is by using Monte Carlo techniques. In this paper, points to be used in evaluating the integral are found using the Metropolis algorithm, which works as follows. First, assuming ψ is real, we can rewrite the integral above as:

$$\int \psi H \psi \, dr = \int dr \, (\psi\psi) \frac{1}{\psi} H\psi = \int dr \, p(r) E_L(r)$$

where $p(r) = \psi\psi$ and $E_L(r) = \frac{1}{\psi} H\psi$. E_L is known as the local energy.

Then using the Monte Carlo method,

$$\int dr \, p(r) E_L(r) \approx \frac{1}{N} \sum_{i=1}^{N} E_L(r_i)$$

where $\{r_1, \ldots, r_N\}$ are points chosen according to the probability distribution $p(r)$, and N is the total number of points used.

In order to come up with the points used, start with an initial point r, then $r_{\text{new}} = r + dr$ where dr is a random point in the hypercube, whose length is determined by the simulator. Define,

$$p = \frac{p(r_{\text{new}})}{p(r)}$$

then if $p \geq 1$ we always accept the new point, but if $p < 1$ we accept the new point with probability p. This algorithm is repeated many times until it has thermalized to the distribution. Once this has happened, points are taken so many steps after a point is accepted so as to avoid the points from being too correlated. The amount of steps it takes to thermalize to the distribution and how many steps to run in between each point taken needs to be determined by the simulator. By doing this process several times, you should end up with enough points with enough randomness to be able to evaluate the integral.

Note that normalizing the wavefunction is unnecessary, since any normalization constant divides out in evaluating p and E_L.

2 One Dimension Harmonic Oscillator

In order to test this method, choosing a system with known analytic solutions is good. For this case, the one dimension harmonic oscillator is used. The hamiltonian for this system is:

$$H = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{1}{2}m\omega^2 x^2$$

and the ground state wave function is:

$$\psi_0 = \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-\frac{m\omega}{2\hbar}x^2}$$

with energy,

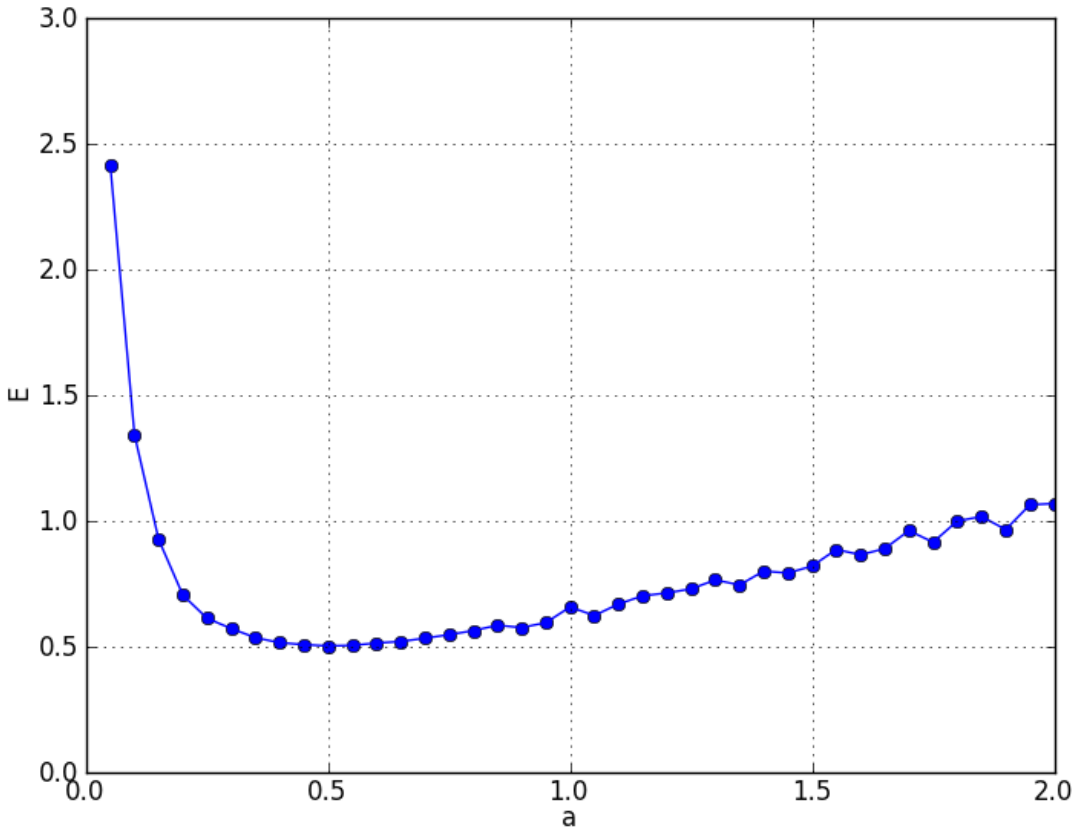
$$E_0 = \frac{1}{2}\hbar\omega$$

In this case, take $\hbar = m = \omega = 1$, and consider trial wavefunctions of the from $\psi_a(x) = e^{-ax^2}$. Then, $p(x) = e^{-2ax^2}$, and

$$E_L = \frac{1}{\psi_a(x)}H\psi_a(x) = a + x^2(\frac{1}{2} - 2a^2)$$

Then solving for the expectation value of H for the trial wavefunction becomes the sum $\frac{1}{N}\sum_{i=1}^{N} E_L(r_i)$, where the r_i are picked by the algorithm above.

Doing this computationally, for different values of a we see that when $a = 1/2$ the expectation value of H reaches a minimum, and at this point $E = 1/2$, which is exactly the analytical result. A graph of E versus a is seen below.



3 Helium

Helium is a three body system consisting of two electrons orbiting a nucleus. If we assume that the nucleus remains fixed, we can write the hamiltonian, in atomic units, as

$$H\psi(r_1, r_2) = \left[-\frac{1}{2}(\nabla_{r_1}^2 + \nabla_{r_2}^2) - \frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}}\right]\psi(r_1, r_2)$$

Where r_1, r_2 are the distances of the two electrons from the nucleus and r_{12} is the distance between them.

The ground state energy has been experimentally determined to be 2.90372 hartrees [3].

3.1 First Trial Function

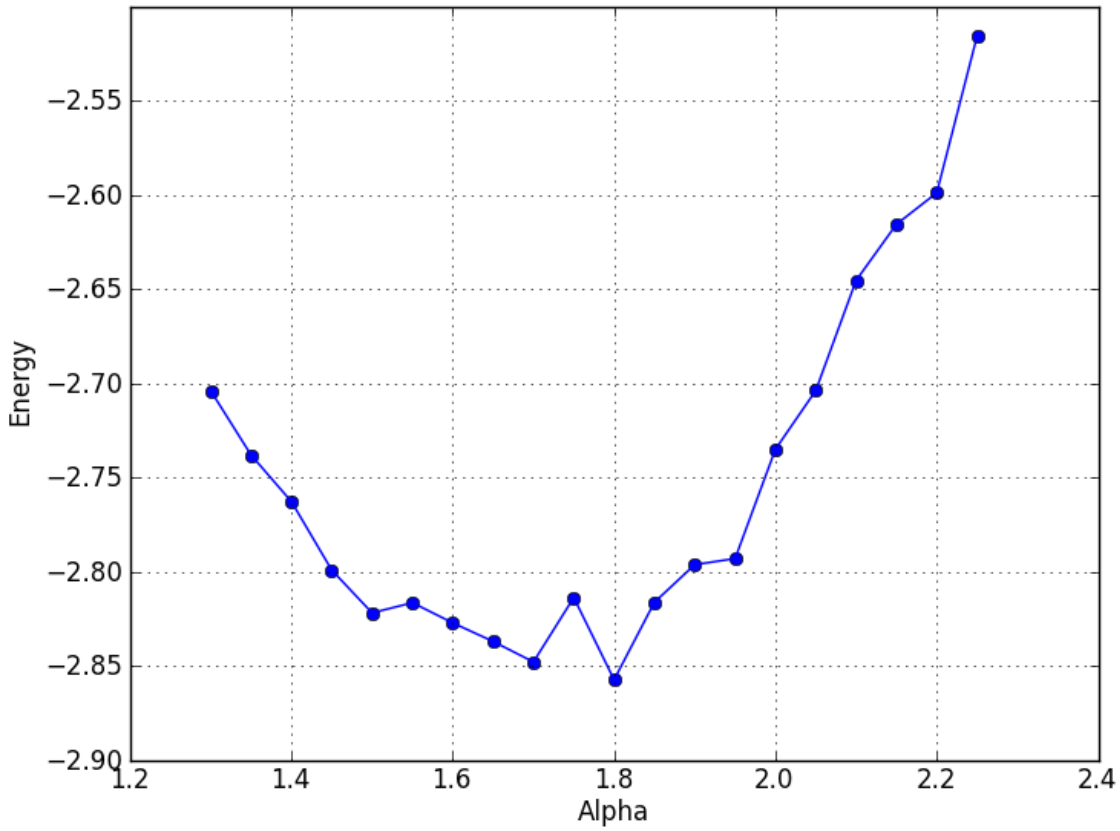
For the first trial wavefunction of Helium, the product of two electrons in the ground state of Hydrogen is taken, while varying the radius of the atom. The wavefunction looks like:

$$\psi(r_1, r_2) = e^{-\alpha(r_1 + r_2)}$$

Then the local energy of this trial wavefunction is:

$$E_L(r_1, r_2) = -\alpha^2 + \frac{\alpha}{r_1} + \frac{\alpha}{r_2} - \frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}}$$

The variational method was applied with 60 walkers doing 60000 steps, waiting 10000 steps to thermalize and taking points every 600 steps, for different values of α. This resulted in a minimum energy found at -2.8575 hartrees. This is within 1.6% of the experimental value for the ground state of Helium. A graph of the energies found with respect to the variational parameter α is seen below. The minimum was taken directly from the minimum that was determined as seen in the graph.



3.2 Second Trial Function

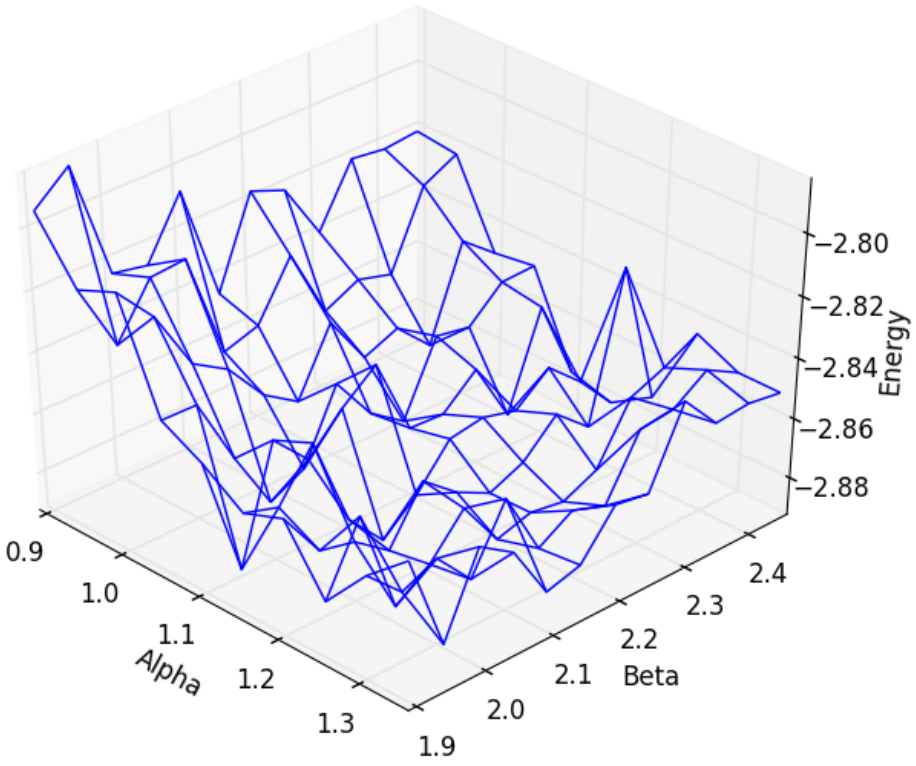
The previous trial function does not account for any electron-electron interactions that happen. To account for this interaction it is clear at least two variational parameters must be used. The trial wavefunction that will be used is:

$$\psi(r_1, r_2) = e^{-\alpha r_1} e^{-\beta r_2} + e^{-\beta r_1} e^{-\alpha r_2}$$

The local energy is found to be:

$$E_L(r_1, r_2) = -\frac{1}{2}(\alpha^2 + \beta^2) + \frac{1}{\psi}\left[\frac{1}{r_1}(\alpha e^{-\alpha r_1} e^{-\beta r_2} + \beta e^{-\beta r_1} e^{-\alpha r_2}) + \frac{1}{r_2}(\beta e^{-\alpha r_1} e^{-\beta r_2} + \alpha e^{-\beta r_1} e^{-\alpha r_2})\right] - V$$

The variational method was applied with 50 walkers doing 60000 steps, waiting 10000 steps to thermalize and taking points every 600 steps, for different values of α and β. A minimum is found when energy is -2.8904, which is 0.46% off of the actual value. A graph of the energy versus both parameters is seen below. The minimum was again taken from the lowest point measured.



4 Analysis

While the graphs for either trial wavefunction for helium are not as smooth as the one for the harmonic oscillator, they both show that the energy reaches a minimum, and this minimum is above the actual value, as expected by the variational principle. The primary issue with using this method is the statistical errors associated with it.

It is clear from looking at the harmonic oscillator that these errors are less the closer the trial wavefunction is to the actual wavefunction. In the case of the harmonic oscillator, the exact form of the wavefunction was

picked and gives the exact answer when the exact wavefunction is used. In the case of the Helium atom, you can't write the exact wavefunction down, due to the fact that this is a three-body problem.

This means that the statistical error can't be eliminated, only be reduced. By simply generating more points, whether by increasing the number of walkers or increasing the number of points taken per walker, would reduce the statistical errors. The issue is the computing time needed to perform these calculations. A standard Monte-Carlo method converges at a rate of $1/\sqrt{N}$, meaning quadrupling the number of samples will halve the error. The method used here should converge better than that, but it is clear that much computing time will be needed to reduce the statistical errors significantly.

5 Conclusion

This method for determining the ground state energy of systems is shown to be effective. The second trial wavefunction gave a ground state energy for Helium was within 0.46% of the experimental value. Three things could improve this result using this method.

The first is using a method that actually minimizes the parameter instead of picking values around where the minimum might be. This wouldn't be too difficult to implement, but would depend on very small amount of statistical error.

The second is to use more complicated trial wavefunctions. The primary issue with this is not computational; it is in doing the math to evaluate the local energy function. The issue comes from having to evaluate $\nabla^2_{r_1} + \nabla^2_{r_2}$ of wavefunctions that might contain terms containing r_{12}, which is not a trivial problem.

The third is to just take more points for the Monte-Carlo integration. This is the easiest to implement, but would increase computation time drastically.

Also, it would be useful to investigate the statistical errors created by this method. Having a better understanding of these would allow fits to be put to the data points to determine a statistically sound minimum energy value and be able to know what sort of error this value has.

Overall, this method is effective, but more research needs to be done in regards to the error associated with this method to provide better results.

References

[1] G. Wysin, *Numerical Quantum Mechanics Techniques*, 2000, `http://www.phys.ksu.edu/personal/wysin/Quantum/index.html`

[2] D. Martin, *Computing the ground state energy of helium*, 2007, `http://www.daniel-martin.co.uk/files/qmc.pdf`

[3] *Five Trial Wavefunctions for two-electron Atoms and Ions*, `http://www.users.csbsju.edu/~frioux/stability/HEATOM.pdf`

6 Code

```
## Variational Monte-Carlo Method by Chris Dopilka December 2, 2011
## Determines ground state energy of Helium Atom for trial wavefunction
## Code in place for 1 or 2 variation parameters and the plots
## Python 2.7.2 used

# Libraries needed:
# Requires numpy, matplotlib
# http://numpy.scipy.org/
# http://matplotlib.sourceforge.net/
import math
import random
import matplotlib.pyplot as plt
```

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

random.seed()

# paramaters for simulation
number_walkers = 50
steps = 60000
therm_step = 600
therm = 10000
trials1 = 10
trials2 = 12
dR = 1

# the paramaters to be varied
# Paramaters are later described in list A = [alpha, beta]
A1 = 0.9
dA1 = 0.05
A2 = 1.9
dA2 = 0.05

# Used for generating initial position and the next step
DIST = 2
def ran_pos():
        return DIST * (2*random.random()-1)

def rand():
        return random.random()*2 - 1


## Used for plotting 3d
Z = np.empty((trials1,trials2))
X = np.empty((trials1,trials2))
Y = np.empty((trials1,trials2))

for i in range(0,trials1):
        for j in range(0,trials2):
                X[i][j] = A1+i*dA1
                Y[i][j] = A2+j*dA2

## Used for plotting 2d
# Y = []
# X = []
# for i in range(0,trials1):
        # X.append(A1+i*dA1)

## Describes both vectors r_1, r_2
## Provides methods for finding their norm
## the distance between them
## and the dot product with the vector r_12
class Position:
        def __init__(self,R):
                self.R = R
```

```
        def norm(self,i):
                return math.sqrt(self.R[i][0]**2+self.R[i][1]**2+self.R[i][2]**2)

        def dist(self):
                sum = 0
                for i in range(0,3):
                        sum += (self.R[1][i] - self.R[0][i])**2
                return math.sqrt(sum)

        def dot(self,i):
                sum = 0
                for j in range(0,3):
                        sum += self.R[i][j]*(self.R[0][j] - self.R[1][j])
                return sum / (self.norm(i)*self.dist())

        # Picks new point
        def step(self):
                R_new = [[0,0,0],[0,0,0]]
                for i in range(0,2):
                        for j in range(0,3):
                                R_new[i][j] = self.R[i][j] + rand()*dR
                return R_new

## Probability distribution
# Is wavefunction squared
def prob(R,A):
        r1 = R.norm(0)
        r2 = R.norm(1)

        ## Trial Wavefunction #1
        #return (math.exp(-A[0]*(R.norm(0) + R.norm(1))))**2

        ## Trial Wavefunction #2
        return (math.exp(-A[0]*r1 - A[1]*r2)+math.exp(-A[1]*r1 - A[0]*r2))**2

## Local Energy function
def energy(R,A):
        r = R.dist()
        r1 = R.norm(0)
        r2 = R.norm(1)
        V = (2/r1+2/r2 - 1 / r)

        ## trial wavefunction #1
        #return -A[0]**2 + A[0]/r1 + A[0]/r2 - V

        ## trial wavefunction #2
        wave = math.exp(-A[0]*r1 - A[1]*r2)+math.exp(-A[1]*r1 - A[0]*r2)
        return -0.5*(A[0]**2+A[1]**2) + ((A[0]*math.exp(-A[0]*r1 -A[1]*r2) +A[1]*math.exp(-A[1]*r1 -A[0]

##Used to extract minimum value
B = [0,0]
min = [0,B]

## Actual simulation
```

```
# one loop for each paramater varried
for j in range(0,trials1):
        for l in range(0,trials2):
                avg = 0
                A = [A1+j*dA1, A2+l*dA2]

                # loop over for mulltiple walkers
                for k in range(0,number_walkers):

                        # creates initial point
                        Ri = [[ran_pos(),ran_pos(),ran_pos()], [ran_pos(),ran_pos(),ran_pos()]]
                        R = Position(Ri)

                        #for storred monte-carlo points
                        points = []

                        for i in range(0,steps):
                                # generate new point
                                R_new = Position(R.step())

                                # determine probability of acceptance
                                # catch division by zero for debugging
                                try:
                                        p = prob(R_new,A)/prob(R,A)
                                except ZeroDivisionError:
                                        print R,A

                                # check if we should accept new position
                                if p >= random.random():
                                        R = R_new

                                # if we've termalized, take every therm_step points
                                if i > therm:
                                        if (i-therm) % therm_step == 0:
                                                points.append(R)

                        # Evaluate integral
                        sum = 0
                        for i in range(0,len(points)):
                                sum += energy(points[i],A)
                        avg += sum/len(points)

                # average together value of integral
                # for different walkers
                # and keep minimum
                del points
                if avg/number_walkers < min[0]:
                        min[0] = avg/number_walkers
                        min[1] = A

                #prints energy value for each set of paramaters A
                #print A, avg/number_walkers

                # for plotting 3d
```

```
                Z[j][l] = avg/number_walkers

                # for plotting 2d
                #Y.append(avg/number_walkers)

# Report minimum energy found and parameters used
print min

## Make 2d Plot
# plt.xlabel('Alpha')
# plt.ylabel('Energy')
# plt.plot(X,Y,'bo',X,Y)
# plt.grid(True)
# plt.show()

## Make 3d Plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('Alpha')
ax.set_ylabel('Beta')
ax.set_zlabel('Energy')
ax.plot_wireframe(X,Y,Z)
plt.show()
```