Compute energy levels of Helium Atom with Gaussian Linear Combination of Atomic Orbitals

We expect to numerically approximate the energy evels of the Helium atom with a linear combination of atomic orbitals. In this case, we use Gaussian eigenfunctions to expand the wavefunction:

$$|\psi >= \sum_i d_i e^{\alpha_i (r_1^2 + r_2^2)}$$

We then adjust the values of d_i and α_i such that they minimize the operator $S/(T+V)$. We will choose the BFGS algorithm to accomplish the minimization. So, we need only compute the matrix elements of S, T, V. To calculate those integrals, we have to calculate:

$S_{ij} < \psi_i | \psi_j >$

$T_{ij} < \psi_i | T | \psi_j >$

$V_{ij} < \psi_i | V | \psi_j >$

Each of these integrals therefore looks like:

$$\int_{r_1} \int_{r_2} d^3 r_1 d^3 r_2 e^{-\alpha_i (r_1^2 + r_2^2)} F(r_1, r_2) e^{-\alpha_j (r_1^2 + r_2^2)}$$

where F is either "1" (the identity operator), T, or V, and the 3-d differential element is $d^3 r = r^2 \cos(\theta) d\theta d\phi$. Note: all of the ϕ integrals are symmetric and only give factors of 2π.

For the Helium atom, we have

$T = -\frac{1}{2}\nabla_1^2 - \frac{1}{2}\nabla_2^2$

$V = -\frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}}$

Luckily, most of these integrals are not difficult because they factorize into r_1 and r_2 separately.

S matrix elements

$$S_{ij} = \int_{r_1} d^3 r_1 e^{-(\alpha_i+\alpha_j)r_1^2} \int_{r_2} d^3 r_2 e^{-(\alpha_i+\alpha_j)r_1^2}$$

$$S_{ij} = \frac{\pi^3}{(\alpha_i + \alpha_j)^3}$$

T matrix elements

In 3d spherical coordinates ignoring variations in θ and ϕ, $\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial}{\partial r}\right)$, this also factorizes into two separate r_1 and r_2 integrals, so simplifying (you can use mathematica for this if you want) we get:

$$T_{ij} = -\frac{1}{2}\int_{r_1}\int_{r_2} d^3 r_1 d^3 r_2 e^{-\alpha_i(r_1^2+r_2^2)}(\nabla_1^2 + \nabla_2^2)e^{-\alpha_j(r_1^2+r_2^2)}$$

$$T_{ij} = \frac{6\alpha_i\alpha_j\pi^3}{(\alpha_i + \alpha_j)^4}$$

V matrix elements

The potential has three terms $(2/r_1, 2/r_2, 1/r_{12})$. The first two are straightforward and also factorize into two separate r_1 and r_2 integrals. The third requires an integral also over $\cos(\theta)$ so that piece becomes

$$\int_{r_1}\int_{r_2} r_1^2 r_2^2 dr_1 dr_2 \sin(\theta_{12}) d\theta_{12} \frac{e^{-\alpha_i(r_1^2+r_2^2)}e^{-\alpha_j(r_1^2+r_2^2)}}{\sqrt{r_1^2 + r_2^2 - 2r_1 r_2 \cos(\theta_{12})}}$$

Adding all of the pieces together we would get

$$V_{ij} = \frac{-(8pi^{5/2})}{(\alpha_i + \alpha_j)^{5/2}} + \frac{\sqrt{2}\pi^{5/2}}{(\alpha_i + \alpha_j)^{5/2}}$$

Code

The code is very similar to the case for Hydrogen, but we need to adjust the `S_ij`, `T_ij`, and `V_ij` methods to input two indices `alpha_i` and `alpha_j`. We store these values in an array `alpha`, and initialize all of the pieces to have `d_i=1.0` and `alpha_i=0.5`, and pick 3 Gaussians to sum.

Import path and add our software

In [1]:

```
import os
import sys
import numpy as np
import scipy.optimize
```

Add the cpt BFGS minimization, initialize constants for 2 Gaussians

In [2]:

```
# physical constants
hbar = 1.0                    # Planck's constant / 2pi
m = 1.0                       # electron mass
e = 1.0                       # proton charge

# LCAO variational wave function
# psi = sum( d_i g(alpha_i, r) ) for i = 0, 1, 2, ...
# assume d_0 = 1 and vary alpha_0, d_1, alpha_1, d_2, alpha_2, .
..
# vector of variational parameters
p = [ 1.0, 1.0, 0.5, 1.0, 0.5, 1.0, 0.5 ]       # initial guess
for [ alpha_0, d_1, alpha_1 ]
N = int( (len(p) + 1) / 2 ) # number of Gaussians

accuracy = 1.0e-6             # desired accuracy for numerical ope
rations
```

Define the g function, matrix elements, minimization function and derivative

In [3]:

```
class LCAOGauss:
    def __init__(self, p):
        self.d = p[0::2]
        self.alpha = p[1::2]
        self.ii = np.arange(len(self.alpha))

        self.jj = np.arange(len(self.alpha))
        self.i, self.j = np.meshgrid(self.ii,self.jj)

    def Sij(self):  # matrix elements of S
        return np.pi**3.0/(self.alpha[self.i] + self.alpha[self.j])**3.0

    def Tij(self):  # matrix elements of T
        return (6.0*self.alpha[self.i]*self.alpha[self.j]*hbar**2.0*np.pi**3.0)/((self.alpha[self.i] + self.alpha[self.j])**4.0*m)

    def Vij(self):  # matrix elements of V
        return -(8.0*e**2.0*np.pi**2.5)/(self.alpha[self.i] + self.alpha[self.j])**2.5 + (np.sqrt(2.0)*e**2.0*np.pi**2.5)/(self.alpha[self.i] + self.alpha[self.j])**2.5

    def E(self):            # energy as function of N alpha_i and d_i
        S = H = 0.0
        fac = (self.alpha[self.i] * self.alpha[self.j])**(3.0/4.0)* self.d[self.i] * self.d[self.j]
        Hvals = fac * (self.Tij() + self.Vij() )
        Svals = fac * self.Sij()
        H = np.sum(Hvals)
        S = np.sum(Svals)
        return H / S

    def __call__(self, p):              # function for BFGS minimization
        # assume p = [ d0, alpha_0, d_1, alpha_1, d_2, alpha_2, ... ]
        self.alpha = np.where( p[1::2] > accuracy, p[1::2], accuracy)
        self.d = p[::2]
        e = self.E()
        return e

    def norm(self):                     # norm of LCAO
        Sijvals = self.Sij() * self.d[self.i] * self.d[self.j]
        return np.linalg.norm(Sijvals)
```

Drive the simulation

In [4]:

```
print(" Variational method for Helium using Gaussian LCAO")
print(" Minimize <psi|H|psi>/<psi|psi> using BFGS algorithm")
lcao = LCAOGauss(p)
res = scipy.optimize.minimize(lcao, p, tol=accuracy)
alpha = res.x[1::2]
d = res.x[::2]
A = lcao.norm()
d /= A

print("Energy level = %6.2f" % (res.fun))
print("Eigenfunction expansion:")
print( "%3s %7s %7s" % ( 'i', 'alpha', 'd') )
for i in range(len(alpha)):
    print ( "%3d %6.5f %6.5f" % (i, alpha[i], d[i]) )
```

```
 Variational method for Helium using Gaussian LCAO
 Minimize <psi|H|psi>/<psi|psi> using BFGS algorithm
Energy level =  -2.47
Eigenfunction expansion:
  i   alpha       d
  0 0.55235 0.03818
  1 2.08719 0.02296
  2 2.08720 0.02296
```

Results

We can see that the energy level we compute is around -2.5, which is in agreement with our expectations.

In [ ]:

```

```