So Much Potential

February 6, 2019

1 Computational Exercise 2-4: A Surface with so much potential

We are going to construct what is often referred to as an *ab initio* potential energy surface of the diatomic molecule carbon monoxide. That is, we are going to use various electronic structure theories (Hartree-Fock theory, Configuration Interaction theory, and Density Functional theory) to compute the electronic energy at different geometries of a simple diatomic molecule. We will use the interpolation capabilities of scipy to simplify the evalution of the potential energy at separations for which we did not explicitly evaluate the electronic energy. We will also use scipy to differentiate the interpolated potential energy surface to obtain the forces acting on the atoms at different separations.

We will start by importing the necessary libraries:

```
In [34]: import numpy as np
         from matplotlib import pyplot as plt
         from scipy.interpolate import InterpolatedUnivariateSpline
```

We will now define two arrays: r_array will be an array of values for the *CO* bond length and E_array will hold the electronic energy values corresponding to each separation.

```
In [35]: ### create an array of 20 bond lengths spanning 0.5 - 3.5 angstroms
         ### but store in atomic units of length... note 1 angstrom = 1.88973 a.u. of length
         r_array = np.linspace(0.561,3.087,10)*1.88973
         print(r_array)
         ### fill in this array with your actual energy values... there should be 20 values in
         E_array = [-109.0350050004,-112.3137052599, -112.6994904875, -112.5943503269, -112.443
                    -112.3147098978, -112.2973399066, -112.0120671242, -111.9948057317]


         # TO see the plot of the PES, uncomment the following lines
         plt.plot(r_array, E_array, 'red')
         plt.show()
```

```
[1.06013853 1.59052275 2.12090697 2.65129119 3.18167541 3.71205963
 4.24244385 4.77282807 5.30321229 5.83359651]
```
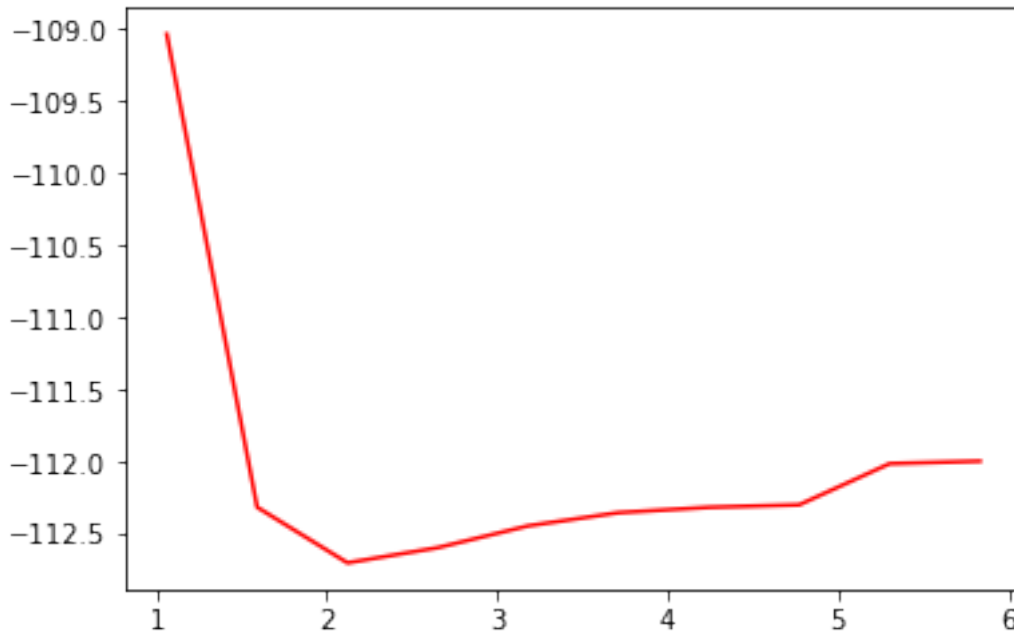
Now that you have the raw data, we will interpolate this data using cubic splines. This will permit us to estimate the potential energy at any arbitrary separation between 0.5 and 3.5 Angstroms (roughly 1 and 5.8 a.u.) with fairly high confidence, and will also allow us to estimate the force

$$F(r) = -\frac{d}{dr}V(r) \tag{1}$$

at any separation between 1.0 and 5.8 a.u. since the derivative of cubic splines are readily available.

```
In [36]: ### use cubic spline interpolation
         order = 3
         ### form the interpolator object for the data
         sE = InterpolatedUnivariateSpline(r_array, E_array, k=order)
         ### form a much finer grid
         r_fine = np.linspace(1.06,5.0,200)
         ### compute the interpolated/extrapolated values for E on this grid
         E_fine = sE(r_fine)
         ### plot the interpolated data
         plt.plot(r_fine, E_fine, 'blue')
         plt.show()

         ### take the derivative to get forces
         fE = sE.derivative()
         F_fine = -1*fE(r_fine)
         ### plot the forces
         plt.plot(r_fine, np.abs(F_fine), 'black')
```

2

```
plt.xlim(1,5)
plt.show()
```
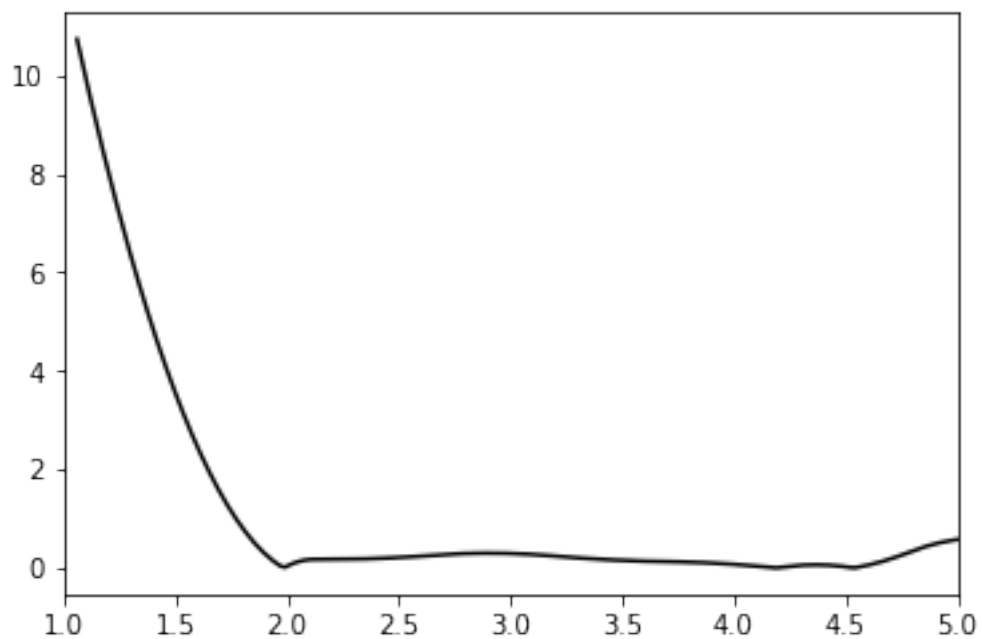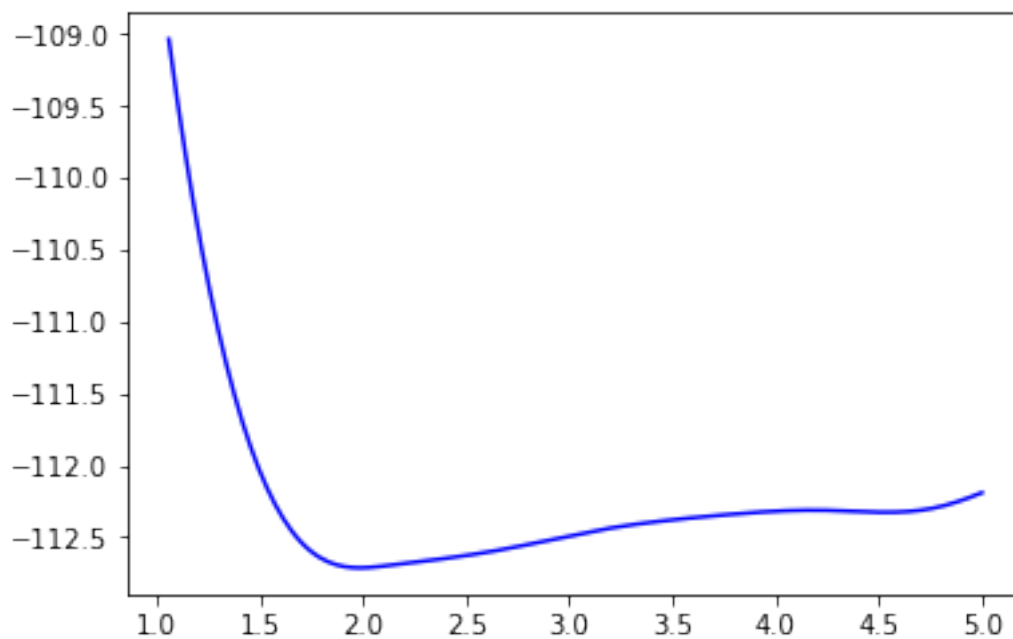

We can estimate the equilibrium bond length by finding the separation at which the derivative of the PES (i.e. the force) goes to zero:

$$\frac{d}{dr}V(r_{eq}) = -F(r_{eq}) = 0 \tag{2}$$

```
In [37]: r_eq_idx = np.argmin(E_fine)
         print(r_fine[r_eq_idx])

1.9905527638190956
```

At this point, take a moment to compare your equilibrium bond length with other teams who have used different levels of theory to compute their potential energy surfaces. Which equilibrium bond length should be most trustworthy?

Recall now that the Harmonic Oscillator potential, which is a reasonable model for the motion of diatomic atomcs near their equilibrium bond length, is given by

$$V(r) = \frac{1}{2}kr^2 \tag{3}$$

and that the vibrational frequency of the molecule is given by

$$\nu = \frac{1}{2\pi}\sqrt{\frac{k}{\mu}} \tag{4}$$

where μ is the reduced mass of the molecule and k is known as the force constant.
We can estimate the force constant as

$$k = \frac{d^2}{dr^2}V(r_{eq}). \tag{5}$$

1.0.1 Question 1: What is the reduced mass of the CO molecule in atomic units?

1.0.2 Question 2: Use your spline fit to the PES of CO to estimate the vibrational frequency of CO. Express your number in atomic units and also convert to a common spectroscopic unit system of your choosing (wavenumbers, nm, microns, Hertz, THz are all acceptable choices).

Next, we want to actually simulate the dynamics of the CO molecule on this *ab initio* potential energy surface. To do so, we need to solve Newton's equations of motion subject to some initial condition for the position (separation) and momentum (in a relative sense) of the particles. Newton's equations can be written

$$F(r) = \mu\frac{d^2}{dr^2} \tag{6}$$

where μ is the reduced mass in atomic units and $F(r)$ is the Force vs separation in atomic units that was determined previously.

4

1.0.3 Question 3: What will be the accelation of the bond stretch when C is separated by O by 3 atomic units? You can express your acceleration in atomic units, also.

If the acceleration, position, and velocity of the bond stretch coordinate are known at some instant in time t_i, then the position and velocity can be estimated at some later time $t_{i+1} = t_i + \Delta t$:

$$r(t_i + \Delta t) = r(t_i) + v(t_i)\Delta t + \frac{1}{2}a(t_i)\Delta t^2 \tag{7}$$

and

$$v(t_i + \Delta t) = v(t_i) + \frac{1}{2}\left(a(t_i) + a(t_i + \Delta t)\right)\Delta t. \tag{8}$$

This prescription for updating the velocities and positions is known as the Velocity-Verlet algorithm.

Note that we need to perform 2 force evaluations per Velocity-Verlet iteration... one corresponding to position $r(t_i)$ to update the position, and then a second time at the updated position $r(t_i + \Delta t)$ to complete the velocity update. To be able to define the very first update, an initial position and velocity must be specified. Typically, these are chosen at random from a sensible range of values. In this case, we will initialize the position to be a random number between 1.0 and 5.0 and the velocity to be a random number between 0.0006 and 0.006, both in atomic units.

```
In [72]: ### get random position and velocity for CO within a reasonable range
         r_init = np.random.uniform(1.0,3.0)
         v_init = np.random.uniform(-0.006,0.006)
         mu = 13625.
         dt = 0.5

         ### get force on particle
         F_init = -1*fE(r_init)
```

Next we need to define our Velocity-Verlet function and call it to update our position and velocity. The following partially-complete function will update the position. You should complete the function to also update the velocity.

```
In [77]: def Velocity_Verlet(r_curr, v_curr, mu, f_interp, dt):
             ### get acceleration at current time
             a_curr = -1*f_interp(r_curr)/mu

             ### use current acceleration and velocity to update position
             r_fut = r_curr + v_curr * dt + 0.5 * a_curr * dt**2

             ### use r_fut to get future acceleration a_fut
             a_fut = -1*f_interp(r_fut)/mu
             ### use current and future acceleration to get future velocity v_fut
             v_fut = v_curr + 0.5 * (a_curr + a_fut) * dt

             result = [r_fut, v_fut]

             return result
```

5

```python
### Now use r_init and v_init and run velocity verlet update 1000 times, plot results
r_vs_t = np.zeros(5000)
v_vs_t = np.zeros(5000)
t_array = np.zeros(5000)
print(r_init, v_init)

r_vs_t[0] = r_init
v_vs_t[0] = v_init
result_array = Velocity_Verlet(r_init, v_init, mu, fE, dt)
print(result_array)
for i in range(1,5000):
    tmp = Velocity_Verlet(result_array[0], result_array[1], mu, fE, dt)
    result_array = tmp
    t_array[i] = dt*i
    r_vs_t[i] = result_array[0]
    v_vs_t[i] = result_array[1]

### Plot the trajectory of bondlength vs time:
plt.plot(t_array, r_vs_t, 'red')
plt.show()

### plot the phase space trajectory of position vs momentum
plt.plot(mu*v_vs_t, r_vs_t, 'blue')
plt.show()
```
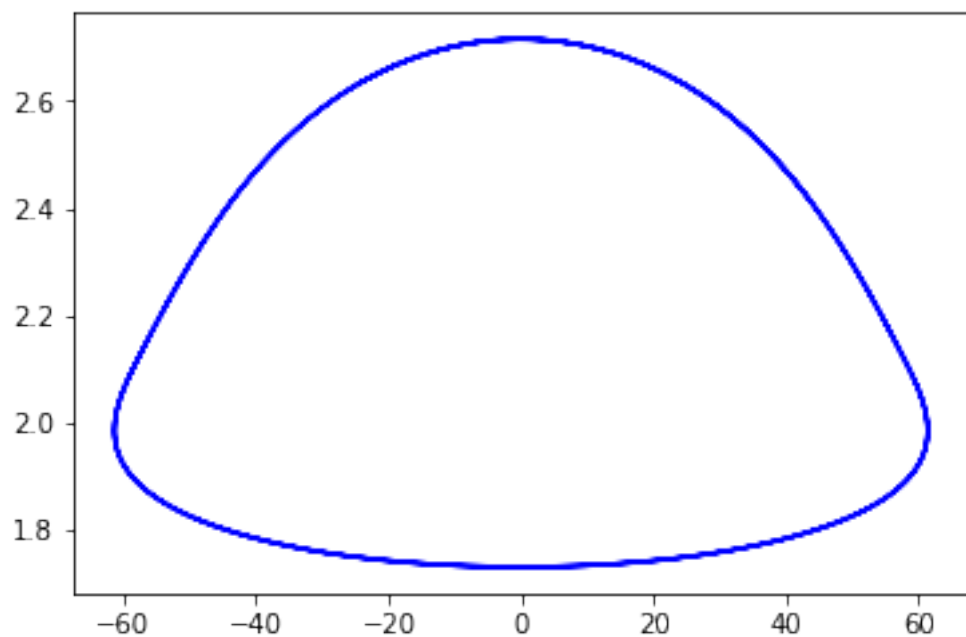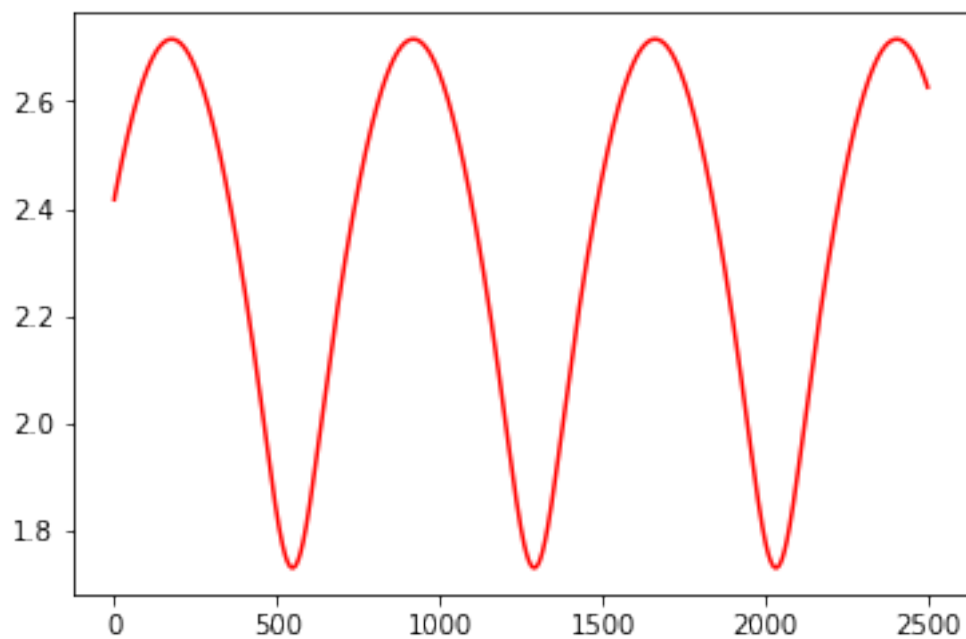
```
2.4164401915461533 0.003190117445975249
[2.418033465186262, 0.003182971874427781]
```

1.1 Input file templates for the CISD, RHF, and B3LYP levels of theory follow:

$CONTRLMAXIT = 199SCFTYP = RHFCITYP = GUGARUNTYP = ENERGYCOORD = ZMTEND\ CONTRLUNIT = BOHREND\ SYSTEMMWORDS = 100END\ BASISGBASIS =$

N311NGAUSS = 6END SCFDIRSCF = .T.DISS = .T.DAMP = .T.END CIDRTIEXCIT = 2NFZC = 0NDOC = 7NVAL = 19END DATACOC1CO11.5END CONTRLMAXIT = 199SCFTYP = RHFRUNTYP = ENERGYCOORD = ZMTEND CONTRLUNIT = BOHREND SYSTEMMWORDS = 100END SCFDIRSCF = .T.DISS = .T.DAMP = .T.END BASISGBASIS = N311NGAUSS = 6END DATACOC1CO11.5END CONTRLMAXIT = 199SCFTYP = RHFDFTTYP = B3LYPRUNTYP = ENERGYCOORD = ZMTEND CONTRLUNIT = BOHREND SYSTEMMWORDS = 100END BASISGBASIS = N311NGAUSS = 6END SCFDIRSCF = .T.DIIS = .T.DAMP = .T.END DATACOC1CO11.5END