Molecular_Dynamics

February 27, 2024

1 Problem Set 2: Classical Dynamics

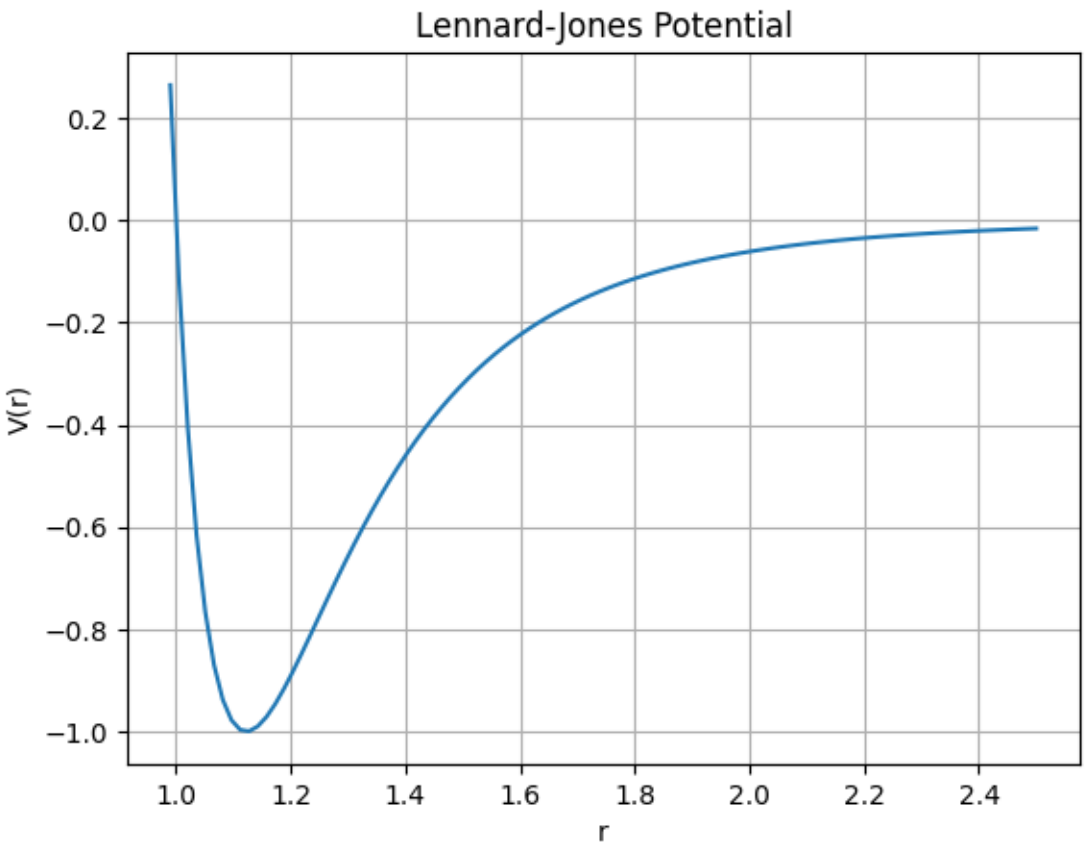
[97]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import sympy as sp
```

1.1 Part A: Diatomic Molecules

1.1.1 A1: The Lennard-Jones Potential

a)

[55]:
```
epsilon = 1
sigma = 1

x1 = np.linspace(0.99, 2.5, 100)
def lj_potential(r, epsilon = 1.0, sigma = 1.0):
    return 4*epsilon*((sigma/r)**12 - (sigma/r)**6)

potential = lj_potential(x1)
plt.plot(x1, potential); plt.xlabel('r'); plt.ylabel("V(r)"); plt.
↪title("Lennard-Jones Potential"); plt.grid(); plt.show()
```


Lennard-Jones Potential
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
V(r)
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
r

$$V(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

$$\frac{d}{dx}V(r) = \frac{d}{dx}\left(4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]\right) = 0$$

$$\frac{d}{dx}\left(4\epsilon\sigma^{12}r^{-12} - 4\epsilon\sigma^{6}r^{-6}\right) = 0$$

$$\frac{-48\epsilon\sigma^{12}}{r^{13}} - \frac{-24\epsilon\sigma^{6}}{r^{7}} = 0$$

$$-\frac{2\epsilon\sigma^{12}}{r^{13}} + \frac{\epsilon\sigma^{6}}{r^{7}} = 0$$

$$r^{6} = 2\sigma^{6}$$

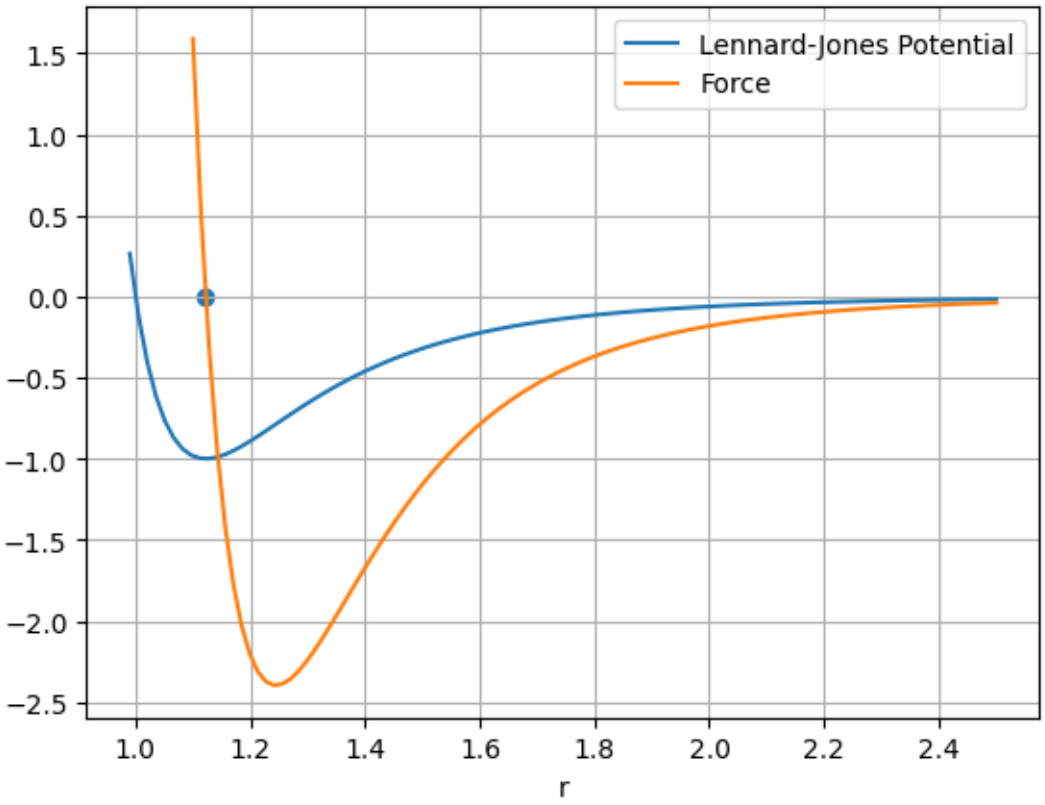
$$r = \sigma\sqrt[6]{2}$$

$$\boxed{\begin{array}{c} r = \sqrt[6]{2} \\ \text{when} \quad [\epsilon, \sigma] = [1, 1] \end{array}}$$

b)

c)

[56]:
```
def lj_force(r, epsilon = 1.0, sigma = 1.0):
    return 24*epsilon*( (2*((sigma/r)**12)) - ((sigma/r)**6))*(1/r)

epsilon = 1.0
sigma = 1.0

x2 = np.linspace(1.1, 2.5, 100)
Force = lj_force(x2)

#x1 and potential are from the last question
plt.plot(x1, potential, label="Lennard-Jones Potential")
plt.plot(x2, Force, label="Force")
plt.scatter(1.1225,0)
plt.xlabel('r');plt.legend();plt.grid();plt.show()
```

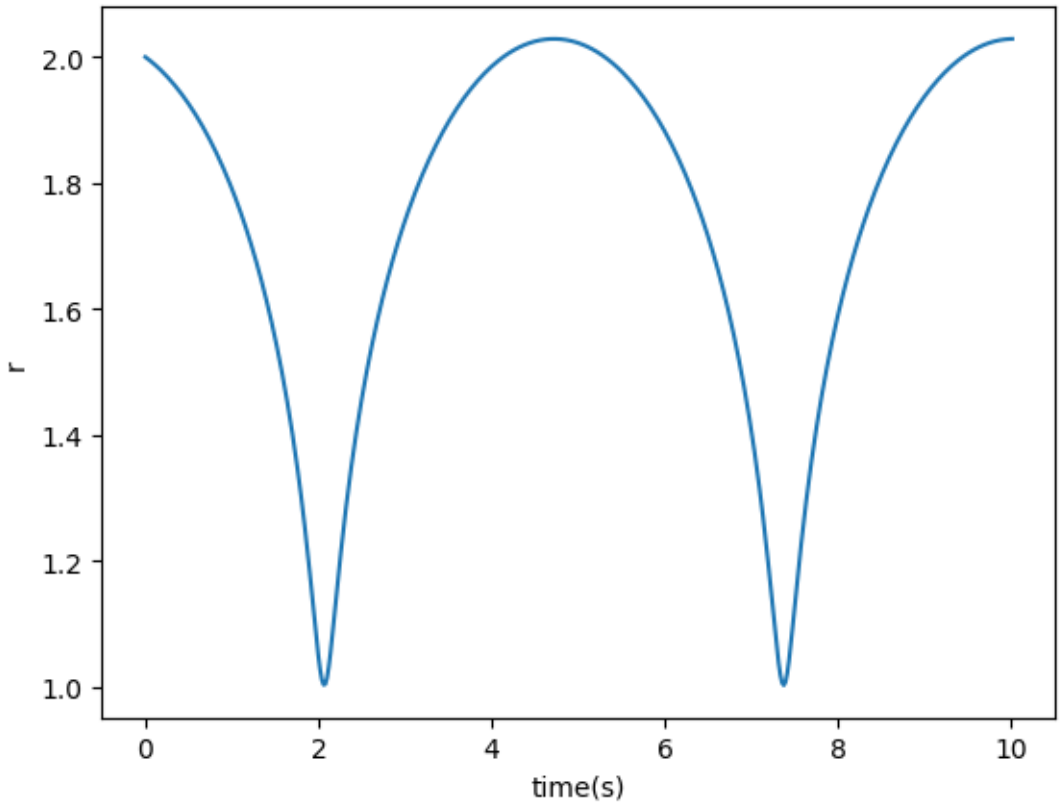


d) The atoms will oscillate. The sign of the force between them at this point marked is negative. This is consistent with my first statement because a negative force will act to bring the atoms closer together.

1.1.2 A2: Dynamics

a)

[57]:
```
def lj_1D_force(r, sigma=1.0, epsilon=1.0):
    return 24*epsilon*( (2*((sigma/r[-1])**12)) - ((sigma/r[-1])**6))*(1/r[-1])

def velocity_verlet(r0, v0, h, m, end_t):
    r = [r0]
    v = [v0]
    t = [0]
    while t[-1] < end_t:
        # step 1: calculate
        F = lj_1D_force(r)
        r.append(r[-1] + (h*v[-1]) + ((h**2)*(F/(2*m))))
        #step 2: evaluate
        F_new = lj_1D_force(r)
        # step 3: calculate
        v.append(v[-1] + ((h/(2*m))*(F + F_new)))
        t.append(t[-1] + h)
    return t, r, v

r0 = 2.0
v0 = -0.1
h = 0.02
end_t = 10.0
m = 1
t, r, v = velocity_verlet(r0, v0, h, m, end_t)
plt.plot(t, r);plt.xlabel("time(s)");plt.ylabel("r");plt.show()
```



b) The curve is sharper for the small radius because the absolute value of the potential's derivative is greater on the left side of the potential than on the right.

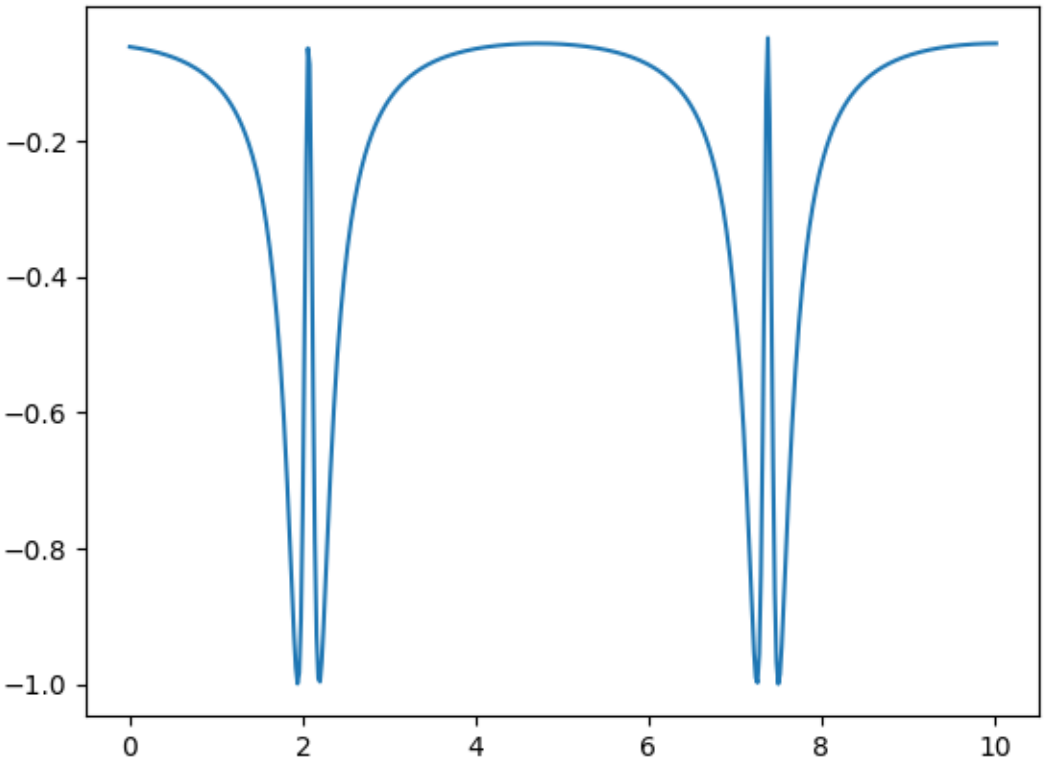
1.2 A3: Conservation of Energy

a)

[58]:
```
def energy_of_system(t, m, v, r_AB, epsilon = 1.0, sigma = 1.0):
    energy = list()
    for i in range(0, len(t)):
        energy_new = ((1/2)*(m/2)*(v[i]**2)) - ((1/2)*(m/2)*(v[i]**2)) +␣
 ↪(4*epsilon*((sigma/r_AB[i])**12 - (sigma/r_AB[i])**6))
        energy.append(energy_new)
    return energy

def rms_energy(energy):
    summation_error = 0
    for i in range(len(energy)):
        summation_error += (energy[i] - energy[0])**2

    return np.sqrt(summation_error/len(energy))
```

```
energy = energy_of_system(t, m, v, r)
error = rms_energy(energy)
plt.plot(t, energy)
print("RMSD with time step",h,":", error)
plt.show()
```

```
RMSD with time step 0.02 : 0.2742331886852962
```



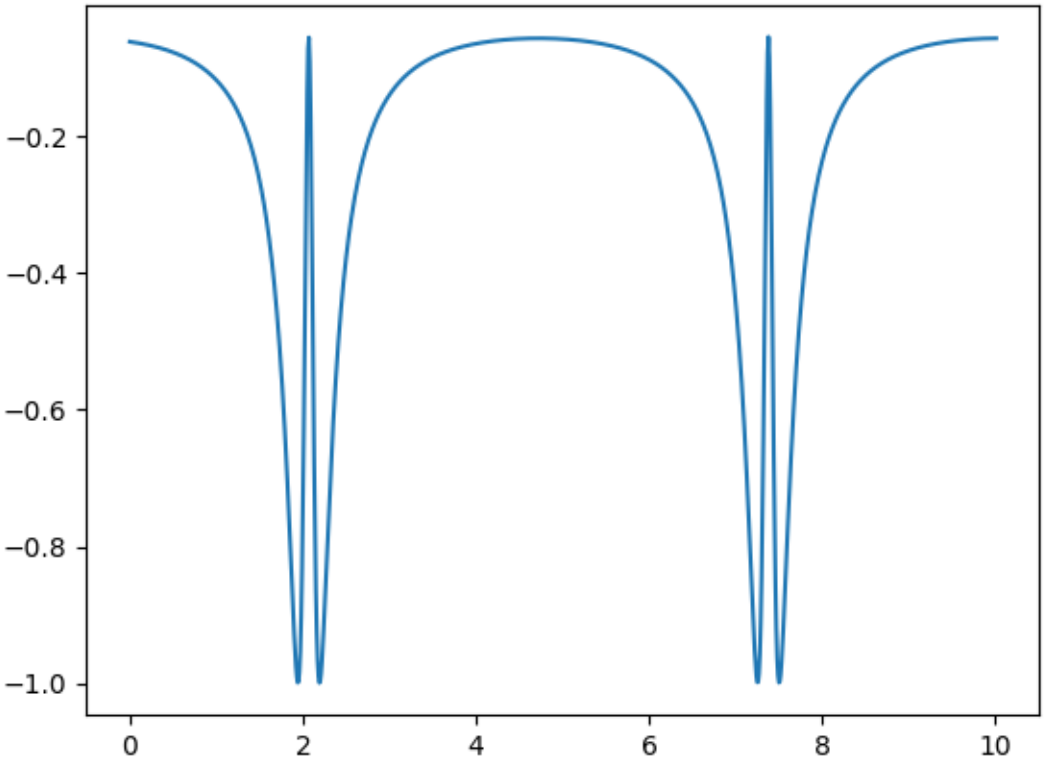
The error is the largest when the atoms are close together (small r_{AB}). The error term is highest at the left side of the potential well.

b)

[59]:
```
r0 = 2.0
v0 = -0.1
h = 0.01
end_t = 10.0
m = 1
t, r, v = velocity_verlet(r0, v0, h, m, end_t)

energy = energy_of_system(t, m, v, r)
error = rms_energy(energy)
```

```
plt.plot(t, energy)
print("RMSD with time step",h,":", error)
plt.show()
```

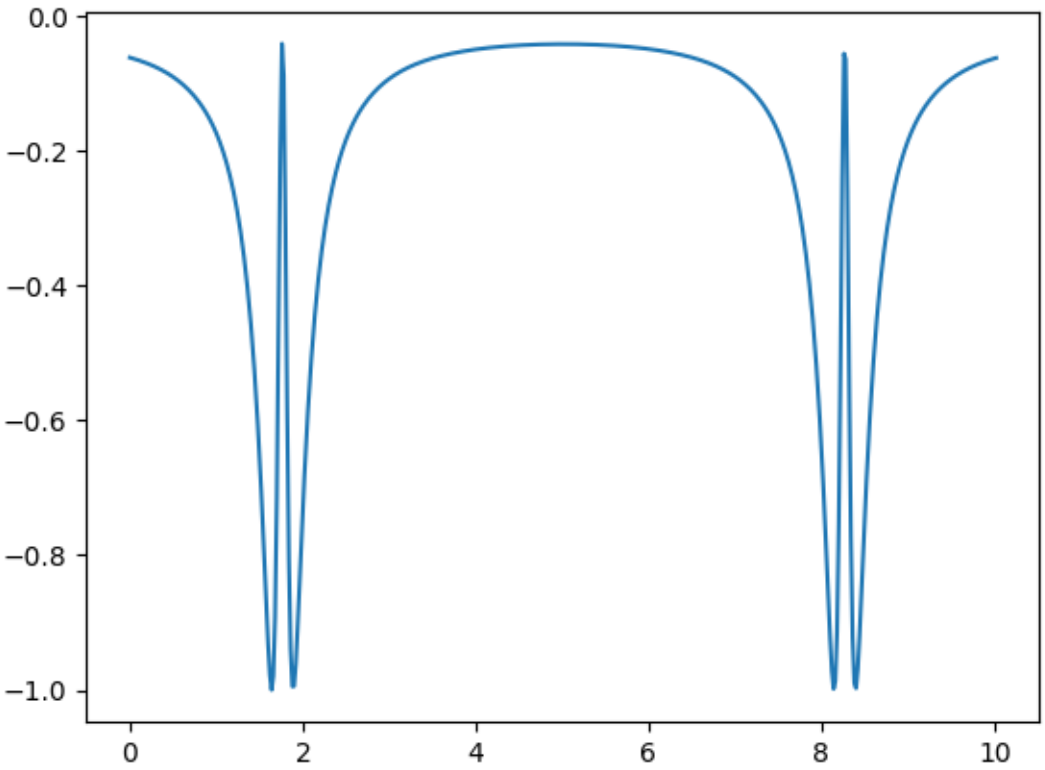
```
RMSD with time step 0.01 : 0.27467903684194883
```



As the step size decreases, so does the error in our energy. We cam say that our algorithm is "recovering accuracy" faster. #### c)

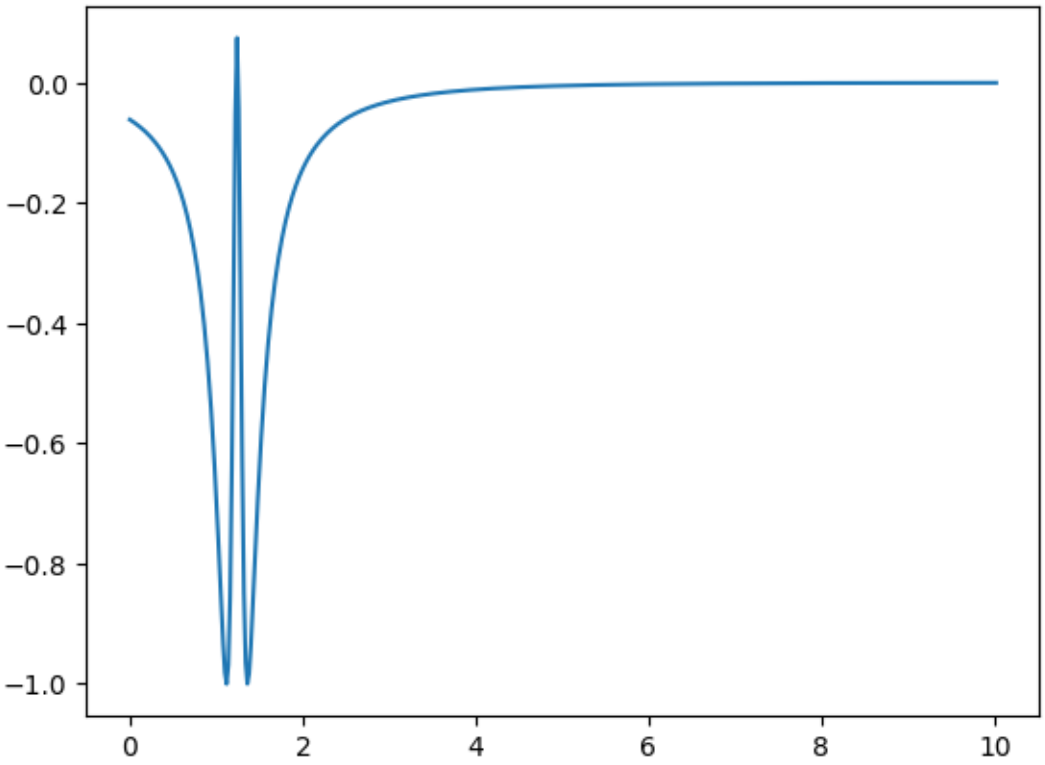
[60]:
```
r0 = 2.0
v0 = -0.2
h = 0.02
end_t = 10.0
m = 1
t, r, v = velocity_verlet(r0, v0, h, m, end_t)

energy = energy_of_system(t, m, v, r)
error = rms_energy(energy)
plt.plot(t, energy)
print("RMSD with time step",h,":", error)
plt.show()
```

```
RMSD with time step 0.02 : 0.27251355237220487
```



[61]:
```
r0 = 2.0
v0 = -0.5
h = 0.02
end_t = 10.0
m = 1
t, r, v = velocity_verlet(r0, v0, h, m, end_t)

energy = energy_of_system(t, m, v, r)
error = rms_energy(energy)
plt.plot(t, energy)
print("RMSD with time step",h,":", error)
plt.show()
```

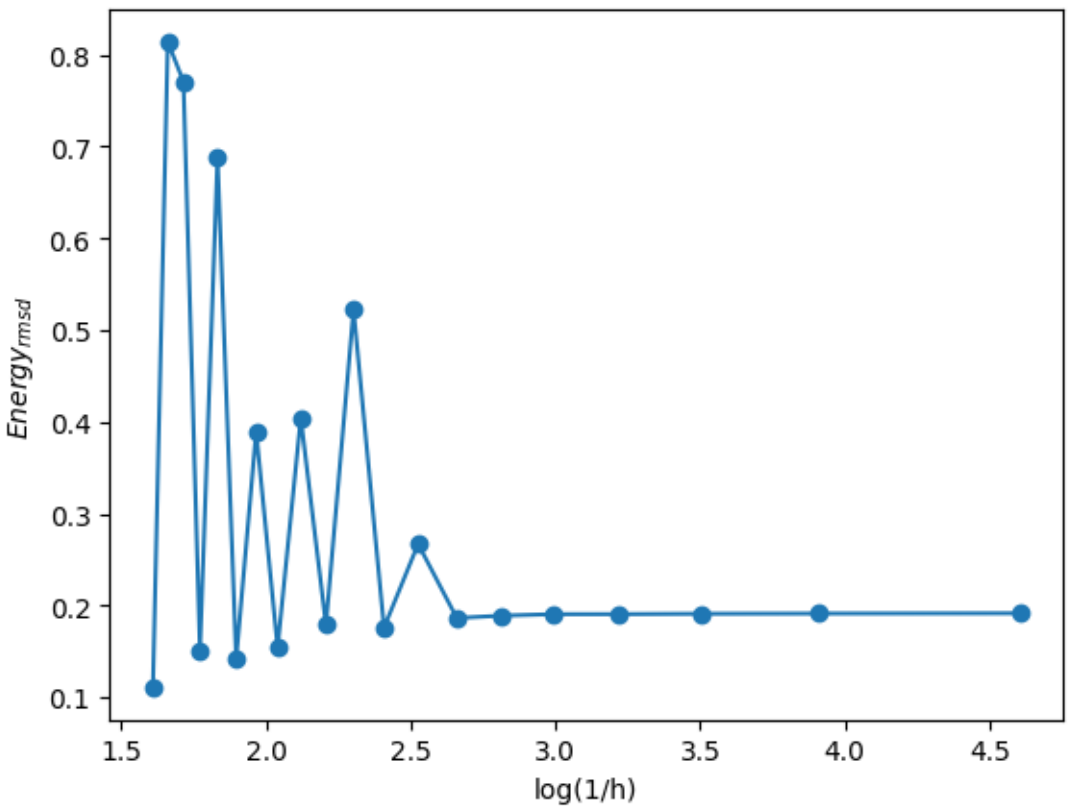
```
RMSD with time step 0.02 : 0.19127402203506075
```

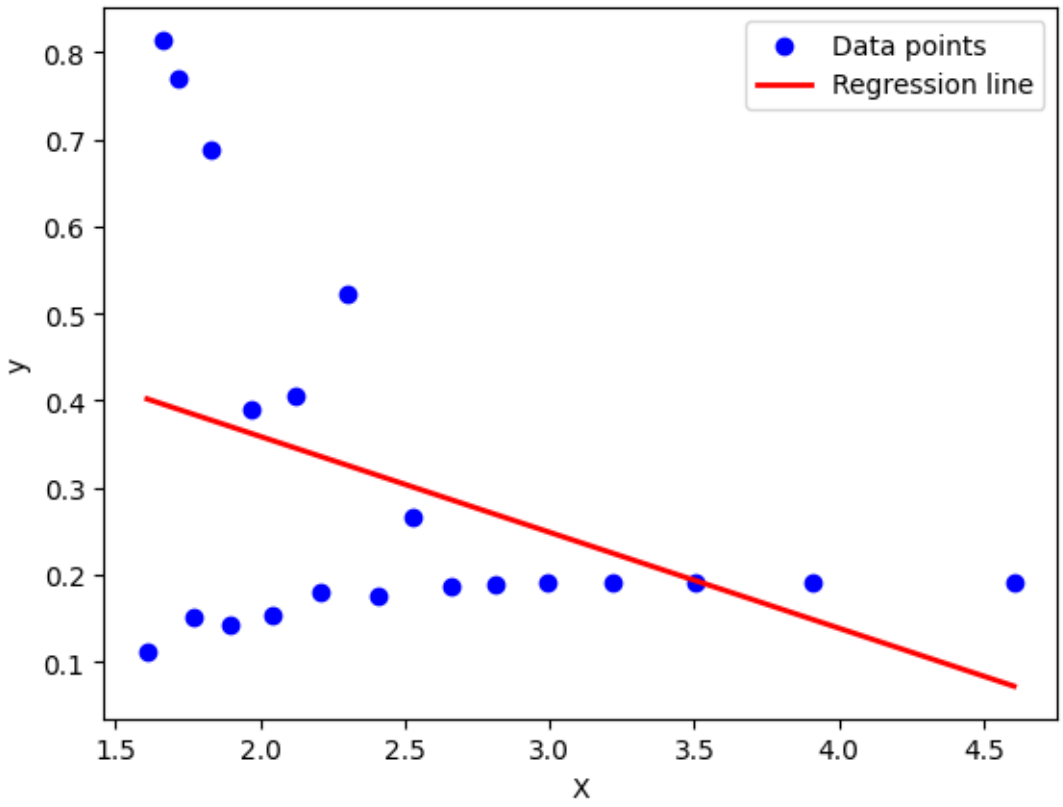


1.2.1 A4: Bonus

[62]:
```
r0 = 2
v0 = -0.5
h = 0.01
m = 1
end_time = 10
error = []
h_list = [round(0.01*x+0.01,2) for x in range(20)]
for h in h_list: # 10 rms energies errors will be calculated from different h␣
 ↪values.
    r0 = 2.0
    v0 = -0.5
    end_t = 10.0
    m = 1
    t, r, v = velocity_verlet(r0, v0, h, m, end_t)

    energy = energy_of_system(t, m, v, r)
    error.append(rms_energy(energy))

h_list_inverse_log = [np.log(1/x) for x in h_list]
```

```
plt.plot(h_list_inverse_log, error);plt.scatter(h_list_inverse_log, error);plt.
  ↪xlabel("log(1/h)");plt.ylabel("$Energy_{rmsd}$")
plt.show()
# linear regression model
H = np.array(h_list_inverse_log)
E = np.array(error)
H = H.reshape(-1, 1)
model = LinearRegression()
model.fit(H, E)
# Create a range of values for plotting the regression line
H_plot = np.linspace(H.min(), H.max(), 100).reshape(-1, 1)

# Predict y values for the plotting range
E_plot = model.predict(H_plot)

# Plot the original data points
plt.scatter(H, E, color='blue', label='Data points')
# Plot the regression line
plt.plot(H_plot, E_plot, color='red', linewidth=2, label='Regression line')
# Label the axes
plt.xlabel('X');plt.ylabel('y');plt.legend();
plt.show()
```





When you increase the energy of the system, the bond breaks. This is why we see the bond length increase linearly.

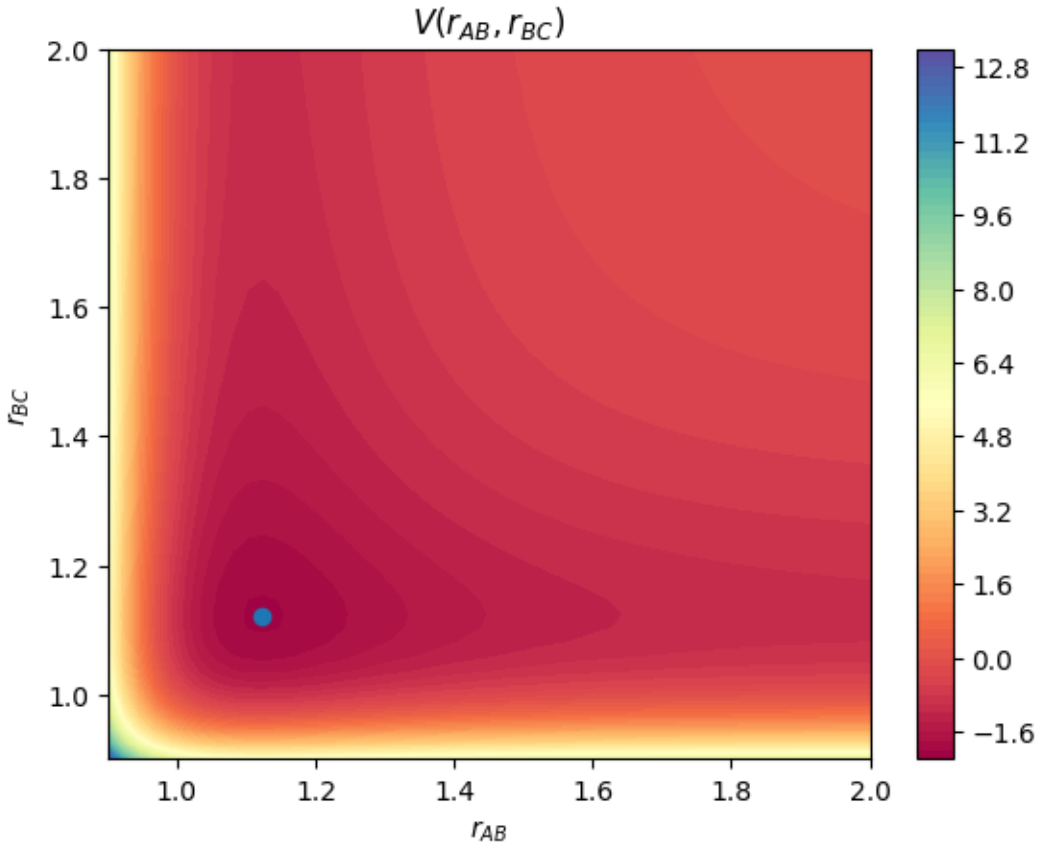
1.3 Part B: Triatomic reaction dynamics with Lennard-Jones potentials

1.3.1 B1: The Lennard-Jones potential

a)

[75]:
```
r_AB = np.linspace(0.9,2.0)
r_BC = np.linspace(0.9,2.0)

def V(r, epsilon=1, sigma=1):
    potential = 4*epsilon*((sigma/r)**12 - (sigma/r)**6)
    return potential


rAB, rBC,= np.meshgrid(r_AB, r_BC)

V_AB_BC = V(rAB) + V(rBC) + V(rAB + rBC)
```

```
plt.contourf(rAB, rBC, V_AB_BC, levels=100, cmap='Spectral')
plt.colorbar()
r_min = 2**(1/6)
plt.scatter(r_min,r_min)

plt.title('$V(r_{AB}, r_{BC})$')
plt.xlabel('$r_{AB}$')
plt.ylabel('$r_{BC}$')
plt.show()
```



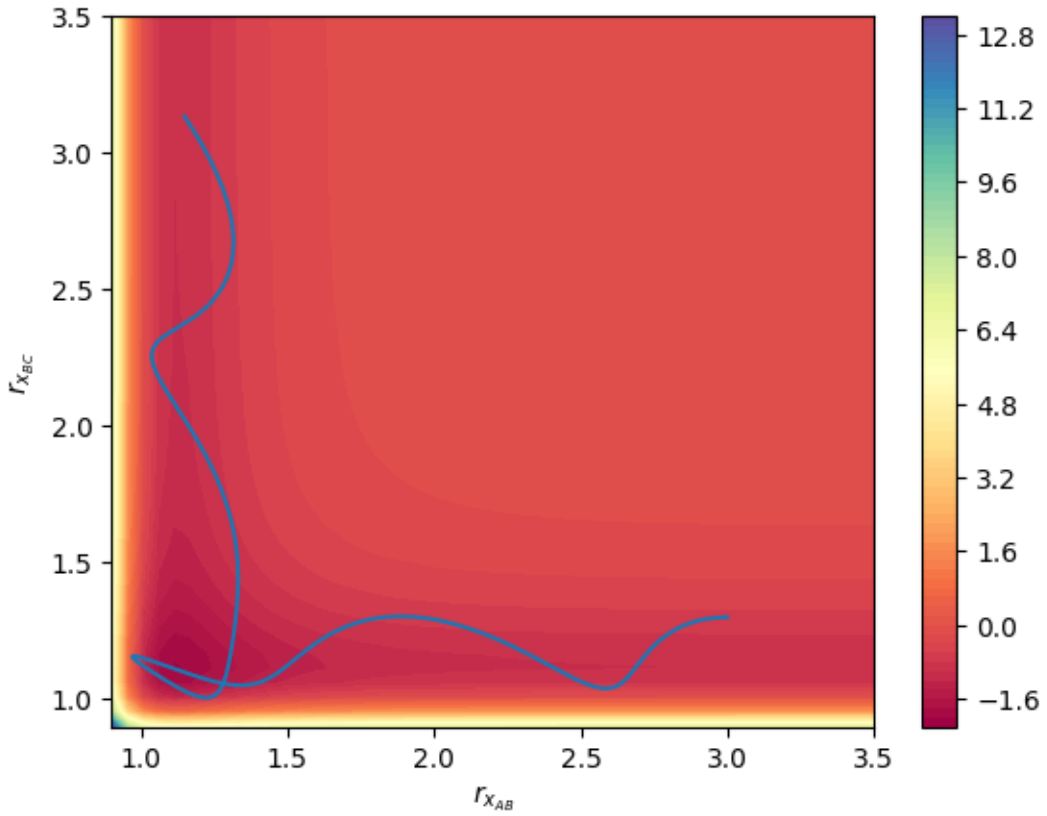
The configuration of the atoms that gives the minimum potential would be when r_AB = r_BC = 2(1/6).**

b)

[64]:
```
def bond_lengths(pos):
    r = [[0 for _ in range(len(pos))] for _ in range(len(pos))]
    for i in range(len(pos)):
        for j in range(len(pos)):
            xij = pos[i][0][-1] - pos[j][0][-1]
```

```
            yij = pos[i][1][-1] - pos[j][1][-1]
            zij = pos[i][2][-1] - pos[j][2][-1]
            r[i][j] = np.sqrt(xij**2 + yij**2 + zij**2)
    return r

def lj_3d(particle_index, pos, r, sigma=1.0, epsilon=1.0):
    #The force vector containing all 3D forces acting on a specified particle.
    f = [0, 0, 0]
    for i in range(len(pos)):
        if i != particle_index:
            f[0] -= (0.5)*(pos[particle_index][0][-1] -␣
 ↪pos[i][0][-1])*((24*epsilon*(sigma**6))/(r[particle_index][i]**8))*(1 -␣
 ↪2*((sigma/r[particle_index][i])**6))
            f[1] -= (0.5)*(pos[particle_index][1][-1] -␣
 ↪pos[i][1][-1])*((24*epsilon*(sigma**6))/(r[particle_index][i]**8))*(1 -␣
 ↪2*((sigma/r[particle_index][i])**6))
            f[2] -= (0.5)*(pos[particle_index][2][-1] -␣
 ↪pos[i][2][-1])*((24*epsilon*(sigma**6))/(r[particle_index][i]**8))*(1 -␣
 ↪2*((sigma/r[particle_index][i])**6))
    return f

def multiple_atoms_Verlet(pos, vel, h, end_time, m):
    t = [0]
    while t[-1] < end_time:
        F = [[0 for _ in range(3)] for _ in range(len(pos))]
        F_new = [[0 for _ in range(3)] for _ in range(len(pos))]
        #Compute all of our r_ij b/c we need them for computing F

        r = bond_lengths(pos)
        for i in range(len(pos)):
            F[i] = lj_3d(i, pos, r)

        for i in range(len(pos)):
            # step 1: calculate
            pos[i][0].append(pos[i][0][-1] + (h*vel[i][0][-1]) +␣
 ↪(((h**2)*F[i][0])/(2*m[i]))) # xi(k+1)
            pos[i][1].append(pos[i][1][-1] + (h*vel[i][1][-1]) +␣
 ↪(((h**2)*F[i][1])/(2*m[i]))) # yi(k+1)
            pos[i][2].append(pos[i][2][-1] + (h*vel[i][2][-1]) +␣
 ↪(((h**2)*F[i][2])/(2*m[i]))) # zi(k+1)

        # step 2: evaluate
        # Because we have computed new positions for each particle, we must␣
 ↪re-compute our new bond lengths.
        r = bond_lengths(pos)
        for i in range(len(pos)):
```

```
            F_new[i] = lj_3d(i, pos, r)

        # step 3: calculate
        for i in range(len(pos)):
            vel[i][0].append(vel[i][0][-1] + ((h/(2*m[i])) * (F[i][0] +␣
 ↪F_new[i][0])))
            vel[i][1].append(vel[i][1][-1] + ((h/(2*m[i])) * (F[i][1] +␣
 ↪F_new[i][1])))
            vel[i][2].append(vel[i][2][-1] + ((h/(2*m[i])) * (F[i][2] +␣
 ↪F_new[i][2])))
        t.append(t[-1] + h)
    return t, pos, vel

# def multiple_atoms_Verlet(pos, vel, h, end_time, m):
A = [[-3.0], [0.0], [0.0]]
B = [[0.0], [0.0], [0.0]]#format particle name = [[x], [y], [z]]
C = [[1.3], [0.0], [0.0]]
pos = [A, B, C]
A_v = [[1.0], [0.0], [0.0]]
B_v = [[0.0], [0.0], [0.0]] #format {particle name}_v  = [[v_x], [v_y], [v_z]]
C_v = [[0.0], [0.0], [0.0]]
vel = [A_v, B_v, C_v]
h = 0.02
m = [1, 1, 1]
end_time = 4
t, pos, vel = multiple_atoms_Verlet(pos, vel, h, end_time, m)

# Converting lists into arrays to make math easier.
A = np.array(pos[0][0])
B = np.array(pos[1][0])
C = np.array(pos[2][0])
vel = np.array(vel)

r_AB_x = np.abs(A-B)
r_BC_x = np.abs(B-C)
r_AC_x = np.abs(A-C)


plt.plot(r_AB_x, r_BC_x)

r_AB = np.linspace(0.9,3.5)
r_BC = np.linspace(0.9,3.5)
rAB, rBC,= np.meshgrid(r_AB, r_BC)
V_AB_BC = V(rAB) + V(rBC) + V(rAB + rBC)
plt.contourf(rAB, rBC, V_AB_BC, levels=100, cmap='Spectral')
plt.colorbar()
plt.xlabel("$r_{x_{AB}}$");plt.ylabel("$r_{x_{BC}}$");plt.show()
```

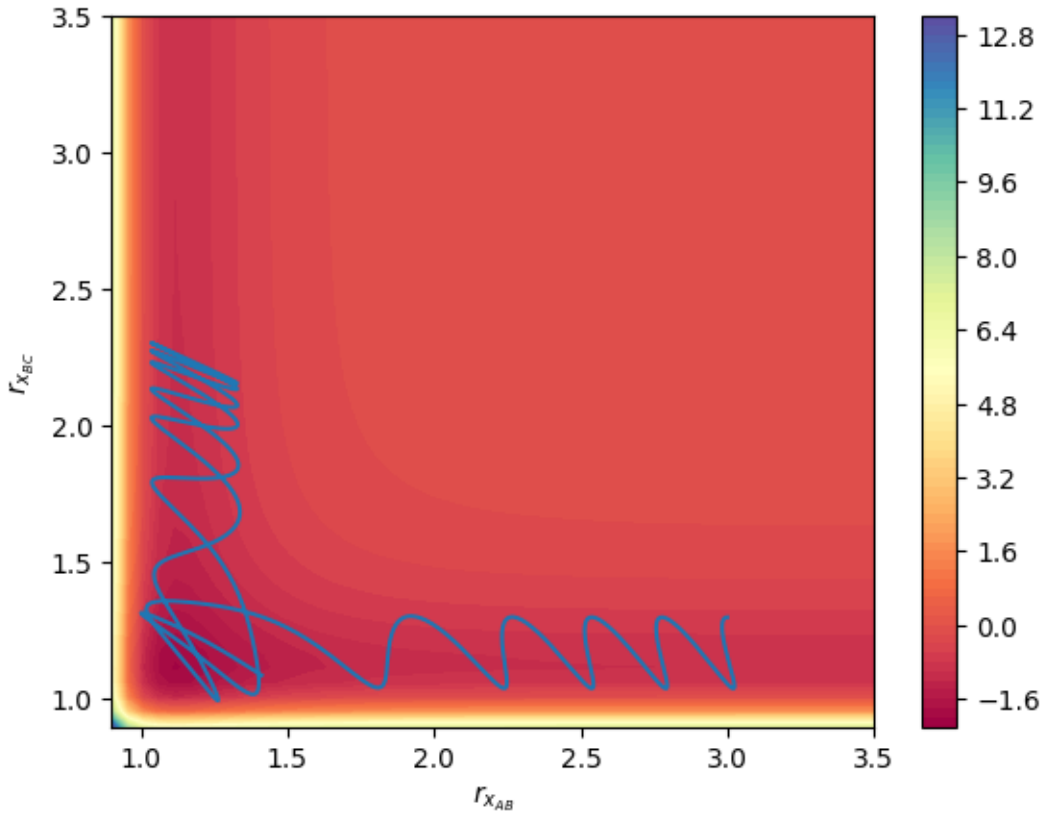


These starting conditions are preparation for simulating what happens when particle with an intial velocity interacts with a pair of bonded particles in a 1-dimensional plane.

c)

[65]:
```
A = [[-3.0], [0.0], [0.0]]
B = [[0.0], [0.0], [0.0]] #format particle name = [[x], [y], [z]]
C = [[1.3], [0.0], [0.0]]
pos = [A, B, C]
A_v = [[0.2], [0.0], [0.0]]
B_v = [[0.0], [0.0], [0.0]] #format {particle name}_v  = [[v_x], [v_y], [v_z]]
C_v = [[0.0], [0.0], [0.0]]
vel = [A_v, B_v, C_v]
h = 0.02
m = [1, 1, 1]
end_time = 15

t, pos, vel = multiple_atoms_Verlet(pos, vel, h, end_time, m)

A_x = np.array(pos[0][0])
```

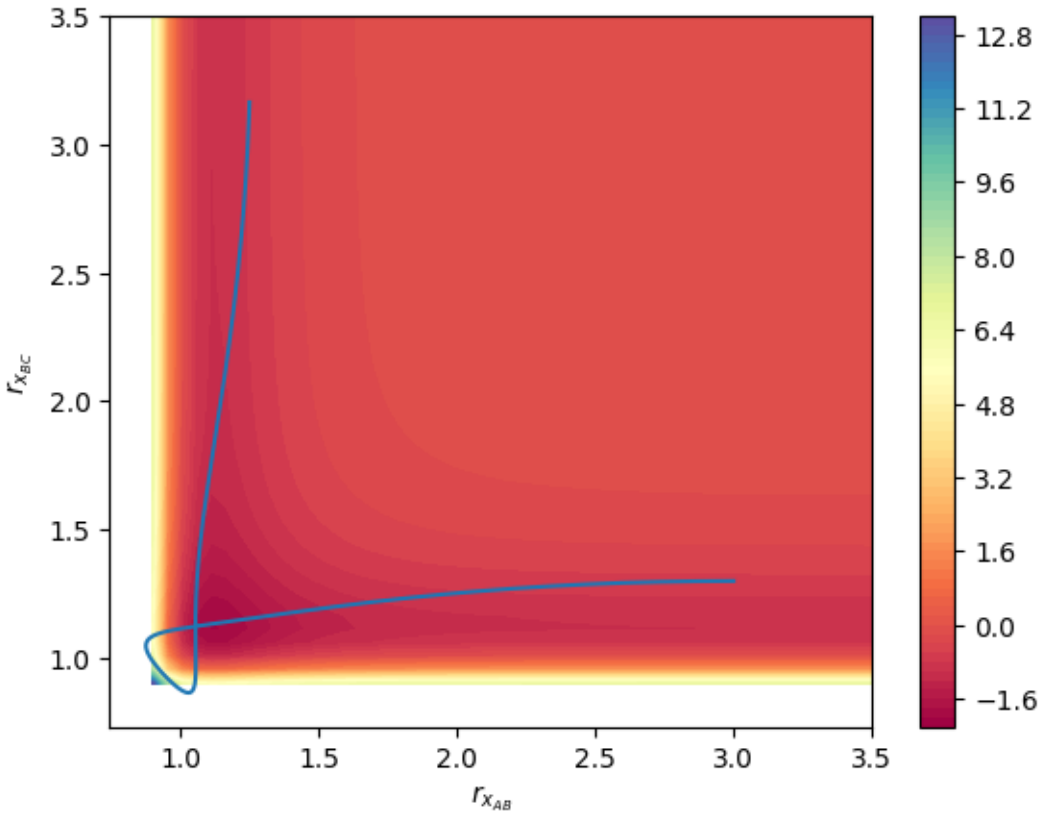
```
B_x = np.array(pos[1][0])
C_x = np.array(pos[2][0])

r_AB_x = np.abs(A_x - B_x)
r_BC_x = np.abs(B_x - C_x)
r_AC_x = np.abs(A_x - C_x)

r_AB = np.linspace(0.9,3.5)
r_BC = np.linspace(0.9,3.5)
rAB, rBC,= np.meshgrid(r_AB, r_BC)
V_AB_BC = V(rAB) + V(rBC) + V(rAB + rBC)
plt.contourf(rAB, rBC, V_AB_BC, levels=100, cmap='Spectral')
plt.colorbar()
plt.plot(r_AB_x, r_BC_x)
plt.xlabel("$r_{x_{AB}}$");plt.ylabel("$r_{x_{BC}}$");plt.show()
```



d)

[76]:
```
A = [[-3.0], [0.0], [0.0]]
B = [[0.0], [0.0], [0.0]] #format particle name = [[x], [y], [z]]
C = [[1.3], [0.0], [0.0]]
```

```
pos = [A, B, C]
A_v = [[5.0], [0.0], [0.0]]
B_v = [[0.0], [0.0], [0.0]] #format {particle name}_v  = [[v_x], [v_y], [v_z]]
C_v = [[0.0], [0.0], [0.0]]
vel = [A_v, B_v, C_v]
h = 0.001
m = [1, 1, 1]
end_time = 1

t, pos, vel = multiple_atoms_Verlet(pos, vel, h, end_time, m)

A_x = np.array(pos[0][0])
B_x = np.array(pos[1][0])
C_x = np.array(pos[2][0])

r_AB_x = np.abs(A_x - B_x)
r_BC_x = np.abs(B_x - C_x)
r_AC_x = np.abs(A_x - C_x)

r_AB = np.linspace(0.9,3.5)
r_BC = np.linspace(0.9,3.5)
rAB, rBC,= np.meshgrid(r_AB, r_BC)
V_AB_BC = V(rAB) + V(rBC) + V(rAB + rBC)
plt.contourf(rAB, rBC, V_AB_BC, levels=100, cmap='Spectral')
plt.colorbar()
plt.plot(r_AB_x, r_BC_x)
plt.xlabel("$r_{x_{AB}}$");plt.ylabel("$r_{x_{BC}}$");plt.show()
```



1.4 Part C: Triatomic reaction dynamics with LEPS potential

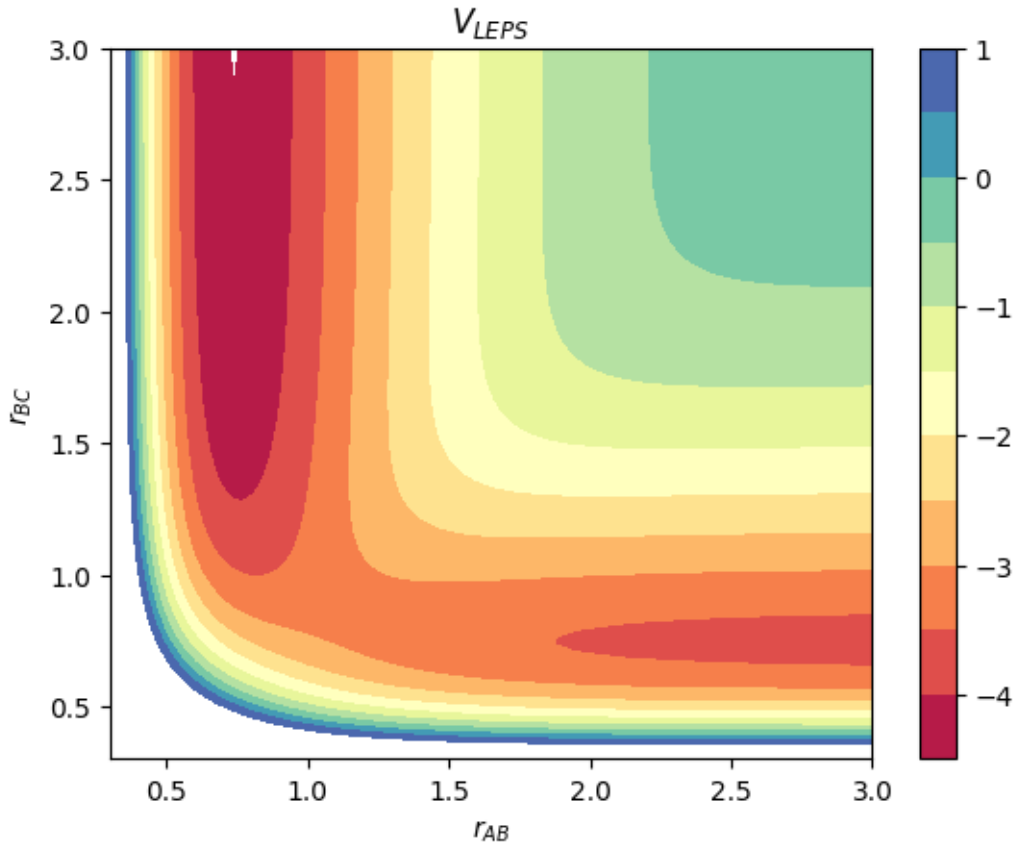
1.4.1 C1: Dynamics on the LEPS potential

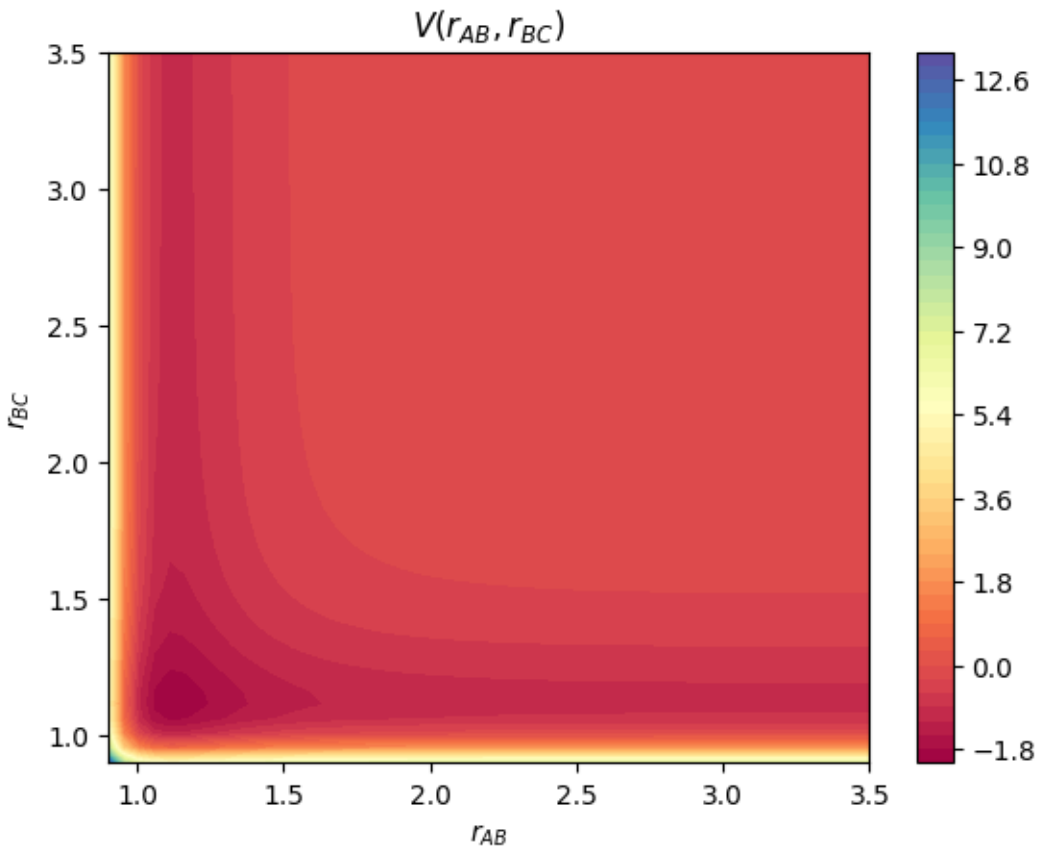
a)

[96]:
```
def Q(r, d, r_0=0.742, alpha=1.942):
    return (d/2) * ((1.5 * np.exp(-2*alpha*(r-r_0))) - (np.exp(-alpha*(r-r_0))))

def J(r, d, r_0=0.742, alpha=1.942):
    return (d/4) * (np.exp(-2*alpha*(r-r_0)) - (6 * np.exp(-alpha*(r-r_0))))


# def V_LEPS(r_AB, r_BC, a, b, c, d_AB, d_BC, d_AC):
#     r_AC = np.abs(r_AB) + np.abs(r_BC)
#     return ((Q(r_AB, d_AB)/(1+a)) + (Q(r_BC, d_BC)/(1+b)) + (Q(r_AC, d_AC)/
 ↪(1+c)) - np.sqrt( ((J(r_AB, d_AB)**2)/((1+a)**2))
#             + ((J(r_AB, d_AB)**2)/((1+a)**2)) + ((J(r_BC, d_BC)**2)/
 ↪((1+b)**2)) + ((J(r_BC, d_BC)**2)/((1+b)**2)) +
#             - ((J(r_AB, d_AB)*J(r_BC, d_BC)) / ((1+a)*(1+b))) - ((J(r_BC,␣
 ↪d_BC)*J(r_AC, d_AC)) / ((1+b)*(1+c)))
```

```
#             - ((J(r_AB, d_AB)*J(r_AC, d_AC)) / ((1+a)*(1+c))) ) )

def V_LEPS(r_AB, r_BC, a, b, c, d_AB, d_BC, d_AC):
    r_AC = np.abs(r_AB) + np.abs(r_BC)  # Simplified geometric relation
    return ((Q(r_AB, d_AB)/(1+a)) + (Q(r_BC, d_BC)/(1+b)) + (Q(r_AC, d_AC)/
 ↪(1+c)) - np.sqrt(
           ((J(r_AB, d_AB)**2)/((1+a)**2)) + ((J(r_BC, d_BC)**2)/((1+b)**2)) +␣
 ↪((J(r_AC, d_AC)**2)/((1+c)**2))
           - ((J(r_AB, d_AB)*J(r_BC, d_BC)) / ((1+a)*(1+b))) - ((J(r_BC,␣
 ↪d_BC)*J(r_AC, d_AC)) / ((1+b)*(1+c)))
           - ((J(r_AB, d_AB)*J(r_AC, d_AC)) / ((1+a)*(1+c))) ) )

a = 0.05
b = 0.30
c = 0.05
d_AB = 4.746
d_BC = 4.746
d_AC = 3.445
r_AB = np.linspace(0.3, 3.0, 100)
r_BC = np.linspace(0.3, 3.0, 100)
R_AB, R_BC = np.meshgrid(r_AB, r_BC)

V = V_LEPS(R_AB, R_BC, a, b, c, d_AB, d_BC, d_AC)

V_levels = [(x*0.5)-4.5 for x in range(12)]
# V_levels = [(x*0.5)-8.0 for x in range(20)]
print(V_levels)

plt.contourf(R_AB, R_BC, V, levels=V_levels, cmap="Spectral")
# plt.contourf(R_AB, R_BC, V, cmap="coolwarm")
plt.colorbar() # Show color scale
plt.title('$V_{LEPS}$')
plt.xlabel('$r_{AB}$')
plt.ylabel('$r_{BC}$')
plt.show()

plt.contourf(rAB, rBC, V_AB_BC, levels=50, cmap='Spectral')
plt.colorbar()
r_min = 2**(1/6)

plt.title('$V(r_{AB}, r_{BC})$')
plt.xlabel('$r_{AB}$')
plt.ylabel('$r_{BC}$')
plt.show()
```

```
[-4.5, -4.0, -3.5, -3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0]
```


V_{LEPS}
3.0
2.5
2.0
1.5
1.0
0.5
r_{BC}
0.5
1.0
1.5
2.0
2.5
3.0
r_{AB}
1
0
−1
−2
−3
−4



The LEPS potential has more minimum points than Lennard-Jones.

b)

[169]:
```
r, r_AB, r_BC, d, a, r_0 = sp.symbols("r r_{AB} r_{BC} d alpha r_0")
d_AB, d_BC, d_AC = sp.symbols("d_AB d_BC d_AC")
a_c, b, c = sp.symbols("a b c")

r_0=0.742
a=1.942
a_c = 0.05
b = 0.30
c = 0.05
d_AB = 4.746
d_BC = 4.746
d_AC = 3.445

r_AC = r_AB + r_BC

#Q Function
```

```
Q = (d/2) * ((1.5 * sp.exp(-2*a*(r-r_0))) - (sp.exp(-a*(r-r_0))))
Q_d = sp.simplify(sp.diff(Q, r))

#Q Derivatives
Q_AB = Q.subs([(r, r_AB), (d, d_AB)])
Q_AB_d = sp.simplify(sp.diff(Q_AB, r_AB))

Q_AC = Q_d.subs([(r, r_AC), (d, d_AC)])
Q_AC_d = sp.simplify(sp.diff(Q_AC, r_AB))

#J Function
J = ((d/4) * (sp.exp(-2*a*(r-r_0)) - (6 * sp.exp(-a*(r-r_0)))))
J_2_dev = sp.simplify(sp.diff(J**2, r))

#J^2 Derivatives
J_AB = J.subs([(r,r_AB),(d, d_AB)])
J_AB_2_d = sp.simplify(sp.diff(J_AB**2, r_AB))

J_BC = J.subs([(r,r_BC),(d, d_BC)])
J_BC_2_d = sp.simplify(sp.diff(J_BC**2, r_AB))

J_AC = J.subs([(r,r_AC),(d, d_AC)])
J_AC_2_d = sp.simplify(sp.diff(J_AC**2, r_AB))

#J*J_other derivatives
J_AB_BC = J_AB*J_BC
J_AB_BC_d = sp.simplify(sp.diff(J_AB_BC, r_AB))

J_BC_AC = J_BC*J_AC
J_BC_AC_d = sp.simplify(sp.diff(J_BC_AC, r_AB))

J_AB_AC = J_AB*J_AC
J_AB_AC_d = sp.simplify(sp.diff(J_AB_AC, r_AB))

#derivative of LEPS Potential
d_rAB_V = ( ((1/(1+a_c))*Q_AB_d) + ((1/(1+c))*Q_AC_d) -
        ( ((1/(1+a_c)**2)*J_AB_2_d) + ((1/(1+c)**2)*J_AC_2_d)
            - ((1/((1+a_c)*(1+b)))*J_AB_BC_d) - ((1/((1+b)*(1+c)))*J_BC_AC_d)
            - ((1/((1+a_c)*(1+c)))*J_AB_AC_d)) )
d_rAB_V
```

[169]:

$$-0.732600732600732\cdot\left(82.2531220782092e^{1.942r_{AB}} - 58.4078112979045e^{3.884r_{AB}}\right)\left(21.1774258697758e^{1.942r_{BC}} - 30.0$$
$$0.732600732600732\cdot\left(21.1774258697758e^{1.942r_{BC}} - 30.0761129237407e^{3.884r_{BC}}\right)\left(59.7054373281565e^{1.942r_{AB}+1.942r_{BC}}\right.$$
$$1664.9452006354e^{-7.768r_{AB}-7.768r_{BC}} - 2293.7097016591e^{-7.768r_{AB}-3.884r_{BC}} -$$
$$3546.82482475009e^{-5.826r_{AB}-5.826r_{BC}} + 2443.13942209926e^{-5.826r_{AB}-3.884r_{BC}} +$$
$$2443.13942209926e^{-5.826r_{AB}-1.942r_{BC}} + 1679.0631150653e^{-3.884r_{AB}-3.884r_{BC}} -$$
$$2313.15922905658e^{-3.884r_{AB}-1.942r_{BC}} + 662.559767378742e^{-3.884r_{AB}-3.884r_{BC}} -$$

$26.1379243384322e^{-1.942r_{AB}-1.942r_{BC}}+3159.92634080525e^{-7.768r_{AB}}-6731.57602164476e^{-5.826r_{AB}}+2951.71203800659e^{-3.884r_{AB}}+18.5421623167951e^{-1.942r_{AB}}$

[171]:
```
#Q Derivatives
Q_BC = Q.subs([(r, r_BC), (d, d_BC)])
Q_BC_d = sp.simplify(sp.diff(Q_BC, r_BC))

Q_AC = Q_d.subs([(r, r_AC), (d, d_AC)])
Q_AC_d = sp.simplify(sp.diff(Q_AC, r_BC))

#J^2 Derivatives
J_AB = J.subs([(r,r_AB),(d, d_AB)])
J_AB_2_d = sp.simplify(sp.diff(J_AB**2, r_BC))

J_BC = J.subs([(r,r_BC),(d, d_BC)])
J_BC_2_d = sp.simplify(sp.diff(J_BC**2, r_BC))

J_AC = J.subs([(r,r_AC),(d, d_AC)])
J_AC_2_d = sp.simplify(sp.diff(J_AC**2, r_BC))

#J*J_other derivatives
J_AB_BC = J_AB*J_BC
J_AB_BC_d = sp.simplify(sp.diff(J_AB_BC, r_BC))

J_BC_AC = J_BC*J_AC
J_BC_AC_d = sp.simplify(sp.diff(J_BC_AC, r_BC))

J_AB_AC = J_AB*J_AC
J_AB_AC_d = sp.simplify(sp.diff(J_AB_AC, r_BC))

#derivative of LEPS Potential
d_rBC_V = ( ((1/(1+a_c))*Q_BC_d) + ((1/(1+c))*Q_AC_d) -
         (((1/(1+a_c)**2)*J_BC_2_d) + ((1/(1+c)**2)*J_AC_2_d)
            - ((1/((1+a_c)*(1+b)))*J_AB_BC_d) - ((1/((1+b)*(1+c)))*J_BC_AC_d)
            - ((1/((1+a_c)*(1+c)))*J_AB_AC_d)) )
#d_rBC_V.subs([(a, )])
d_rBC_V
```

[171]:
$-0.732600732600732\cdot\left(21.1774258697758e^{1.942r_{AB}}-30.0761129237407e^{3.884r_{AB}}\right)\left(82.2531220782092e^{1.942r_{BC}}-58.4\right.$
$0.90702947845805\cdot\left(21.1774258697758e^{1.942r_{AB}}-30.0761129237407e^{3.884r_{AB}}\right)\left(59.7054373281565e^{1.942r_{AB}+1.942r_{BC}}\right.$
$1664.9452006354e^{-7.768r_{AB}-7.768r_{BC}}-3546.82482475009e^{-5.826r_{AB}-5.826r_{BC}}-$
$1852.61168210927e^{-3.884r_{AB}-7.768r_{BC}}+1973.3049178494e^{-3.884r_{AB}-5.826r_{BC}}+$
$1679.0631150653e^{-3.884r_{AB}-3.884r_{BC}}+662.559767378742e^{-3.884r_{AB}-3.884r_{BC}}+$
$1973.3049178494e^{-1.942r_{AB}-5.826r_{BC}}-1868.32091577646e^{-1.942r_{AB}-3.884r_{BC}}-$
$26.1379243384322e^{-1.942r_{AB}-1.942r_{BC}}+3159.92634080525e^{-7.768r_{BC}}-6731.57602164476e^{-5.826r_{BC}}+$
$2951.71203800659e^{-3.884r_{BC}}+18.5421623167951e^{-1.942r_{BC}}$

c)

[ ]:
```
def bond_lengths(pos):
    r = [[0 for _ in range(len(pos))] for _ in range(len(pos))]
    for i in range(len(pos)):
        for j in range(len(pos)):
            xij = pos[i][0][-1] - pos[j][0][-1]
            yij = pos[i][1][-1] - pos[j][1][-1]
            zij = pos[i][2][-1] - pos[j][2][-1]
            r[i][j] = np.sqrt(xij**2 + yij**2 + zij**2)
    return r

def LEPS_f(particle_index, pos, r, sigma=1.0, epsilon=1.0):
    #The force vector containing all 3D forces acting on a specified particle.
    for i in range(len(pos)):
        if i != particle_index:
            f = d_rAB_V.subs([r_AB, ])
    return f

def multiple_atoms_Verlet(pos, vel, h, end_time, m):
    t = [0]
    while t[-1] < end_time:
        F = [[0 for _ in range(1)] for _ in range(len(pos))]
        F_new = [[0 for _ in range(1)] for _ in range(len(pos))]
        #Compute all of our r_ij b/c we need them for computing F

        r = bond_lengths(pos)
        for i in range(len(pos)):
            F[i] = LEPS_f(i, pos, r)

        # step 1: calculate
        for i in range(len(pos)):
            pos[i].append(pos[i][-1] + (h*vel[i][-1]) + (((h**2)*F[i])/
 ↪(2*m[i]))) # xi(k+1)

        # step 2: evaluate
        # Because we have computed new positions for each particle, we must␣
 ↪re-compute our new bond lengths.
        r = bond_lengths(pos)
        for i in range(len(pos)):
            F_new[i] = LEPS_f(i, pos, r)

        # step 3: calculate
        for i in range(len(pos)):
            vel[i].append(vel[i][-1] + ((h/(2*m[i])) * (F[i] + F_new[i])))
        t.append(t[-1] + h)
    return t, pos, vel
```

```
def Verlet_Algo_LEPS(r):
    return t, r, v

A = [-3.0]
B = [0.0]
C = [0.8]
A_v = [1.0]
B_v = [0.0]
C_v = [0.0]
pos = [A, B, C]
vel = [A_v, B_v, C_v]
h = 0.01
end_time = 5
m = [1,1,1]
```