The goal of this project is to give you expertise initializing, running, and analyzing materials systems simulations that progress via molecular dynamics. We'll also be honing our sampling experiences from Project 1. We'll be using HOOMD-Blue to perform simulations of single-component systems using spherically-symmetric Lennard-Jones potentials to model interactions in 3D. Your mission is to solve the following prompts and questions. For each answer, please indicate:

- how many steps it took your simulation to relax to equilibrium
  - See relaxation time, I waited at least 50,000, though some of them may have equilibrated before this.
- how many independent samples inform your measurement
  - I had more than 100 independent samples for each measurement
- some measurement of the precision of your measurement/argument (e.g., a standard deviation or other observations that limit your confidence).
  - I have error bars and standardard deviation for each measurement

1. Characterize your model's behavior at N/V = 0.5 in the NVT ensemble:
   A. Above what temperature is your system "hot"? How do you know?
      a. I would say that above the temperature of about 0.95 kbT is hot, that is when the pressure started to go up and the heat capacity started to drop to that of an ideal gas.
   B. Below what temperature is your system "frozen"? How do you know?
      a. Below temperature 0.5 kbT I would say is frozen. The heat capacity was about that of an ideal solid, and then jumped way up, the pressure was very negative.
   C. How does the system's total energy, potential energy, kinetic energy, heat capacity, and structure vary from frozen to too hot?
      a. See graphs, but KE increased linearly, PE increased slowly, then quickly, then slowly again, the total energy was similar to PE. The pressure was slightly negative until it melted, then it went up linearly, surpassing the pressure of an ideal gas, probably due to the finite size of the particles. The structure smoothly lost peaks, until it just had 1, and the rest was ideal.
2. Characterize finite size effects:
   A. How small is too small to be correct? How large is too large to be practical?
      a. You need at least 5X5 for the simulation to even run at the defualt cutoff.
      b. Some of the attributes seem to converge nicely at 12^3 particles: namely pressure, heat capacity, and structure
      c. The KE is set by the temperature, and the only thing that changes is the standard deviation

    d. The PE does not converge, and continues to drop even when I go to 15^3 particles.
    e. These measurements were done at $kbT = 0.7$ as I thought the temperature had a good amount of variance, and we should see the finite size measurements quite nicely.
    f. 15^3 particles took 40 minutes to run 1,000,000 time steps, while this is still pracitcal, this is getting too long for this project. 12^3 particles where I did all the simulations to 15-20 per simulation. Also I had less independent samples as I went to more particles.
    g. No matter how many particles you do the potential energy will drop in the liquid and solid state. (Still need to confirm this).

3. Contrast your system with an ideal gas:
    A. How does the structure of your model vary with state, and how does it compare to particles with no interactions?
        a. The structure smoothly lost peaks, until it just had 1, and the rest was ideal.
        b. None interacting particles should just be a flat line, see graph.
    B. Does the heat capacity of your system depend on state differently than an ideal gas?
        a. Yes, by quite a bit, at first it was close to an ideal solid, then it jumped way up, then dropped back down to about an ideal gas.
    C. Can you derive or numerically determine an equation of state?
        a. I would say the pressure is 0, then linear at a steeper slope than that of an ideal gas.
4. Summarize your observations, challenges, and any revelations you had while working towards 1-3.. Which specific simulation and state point was your favorite (and why)?
    A. The hardest part was honestly getting hoomd to work, I eventually got it on my own PC.
    B. It was cool to see the phase change, I am guessing I went from a solid, liquid, to gas. Though I am not sure why the heat capacity was so high for the liquid.
        a. This might be because it was past the point way a liquid and a gas are distinct.
    C. No simulation points were my favorite as I just ran most of them once. It woul be cool to zone in on the transition points and do more simulations arround those.
    D. Each simulation took about 20 minutes, this was only possible because I did run it on my own PC.
    E. It would be nice to be able to save the structure every so many steps, but I could not figure out how to do this.

In [1]:
```
import hoomd
import gsd.hoomd
import itertools
import os
import matplotlib.pyplot as plt
```

```
import numpy
import freud
import time
```

In [2]:

```
class LennardJones:
    def __init__(self, replicas, sigma=1, epsilon=1, kT=1, r_cut=2.5, dt = 0.005, buffer = 0.4,
                 rho=0.5, period = 10000, log_file = 'log.gsd'):
        self.sigma = sigma
        self.epsilon = epsilon
        self.r_cut = r_cut
        self.kT = kT
        self.dt = dt
        self.buffer = buffer
        self.cpu = hoomd.device.CPU()
        self.sim = hoomd.Simulation(device=cpu, seed=5)
        self.rho = rho
        self.replicas = replicas
        self.N = replicas**3
        self.period = period
        self.log_file = log_file
        self.pos_log = 'pos_'+log_file

        self.setup_system()
        self.setup_LJ()
        self.setup_logger()


    def setup_LJ(self):
      self.integrator = hoomd.md.Integrator(dt=self.dt)
      self.cell = hoomd.md.nlist.Cell(buffer=self.buffer)
      self.lj = hoomd.md.pair.LJ(nlist=self.cell)
      self.lj.params[('A', 'A')] = dict(epsilon=self.epsilon, sigma=self.sigma)
      self.lj.r_cut[('A', 'A')] = self.r_cut*self.sigma
      self.integrator.forces.append(self.lj)
      self.nvt = hoomd.md.methods.ConstantVolume(
            filter=hoomd.filter.All(),
            thermostat=hoomd.md.methods.thermostats.Bussi(kT=self.kT, tau=1)
        )
      self.integrator.methods.append(self.nvt)

    def setup_system(self):
      #a is the spacing between particles
```

```
        self.a = 1/(self.rho**(1/3.0))
        self.grid_particles = freud.data.UnitCell([self.a,self.a,self.a,0,0,0],[[0,0,0]]).generate_system(self.replicas
        self.L = self.grid_particles[0].Lx

        self.frame = gsd.hoomd.Frame()
        self.frame.particles.N = self.N
        self.frame.particles.position = self.grid_particles[1]
        self.frame.configuration.box = [self.L,self.L,self.L,0,0,0]
        self.frame.particles.typeid = [0]*self.N
        self.frame.particles.types = ['A']

        #Finally, save our initial state:
        with gsd.hoomd.open(name='initial_state.gsd', mode='w') as f:
            f.append(self.frame)

    def setup_logger(self):
        self.sim.create_state_from_gsd(filename='initial_state.gsd')
        self.sim.operations.integrator = self.integrator
        self.sim.state.thermalize_particle_momenta(filter=hoomd.filter.All(), kT=self.kT)
        self.thermodynamic_properties = hoomd.md.compute.ThermodynamicQuantities(filter=hoomd.filter.All())
        self.sim.operations.computes.append(self.thermodynamic_properties)
        self.sim.run(0)

        # Logger setup Thermodynamic properties
        self.logger = hoomd.logging.Logger(categories=['scalar',])
        self.logger.add(self.sim, ['timestep'])
        self.logger.add(self.thermodynamic_properties,
                        ['kinetic_energy', 'potential_energy', 'pressure'])
        self.writer = hoomd.write.GSD(
            trigger=hoomd.trigger.Periodic(self.period),
            filename=self.log_file,
            logger = self.logger,
            mode='wb')
        self.sim.operations.writers.append(self.writer)



    def run(self, steps):
      self.sim.run(steps)

    def get_rdf(self, bins = 100, r_max = 5.0):
        positions = self.get_positions
```

```
        box = freud.box.Box.cube(self.L)

        rdf = freud.density.RDF(bins=bins, r_max=r_max)
        rdf.compute(system=(box, positions))
        r, g_r = rdf.bin_centers, rdf.rdf
        return r, g_r

    def get_positions(self):
        return self.sim.state.get_snapshot().particles.position
```

Running the Simulation

In [38]:

```
cpu = hoomd.device.CPU()
sim = hoomd.Simulation(device=cpu, seed=5)

replicas = 12
steps = 1000000
kT = [0.1, 0.5,0.6, 0.75,0.8, 1,1.5]
kT = [0.1, 0.7, 0.9, 1.25, 2]
kT = [0.2, 0.3, 0.4, 0.95, 1.75]
particles = [15]

for i,part in enumerate(particles):
    start_time = time.time()
    print(f'particles {part**3}')
    lj = LennardJones(part, sigma=1, epsilon=1, kT=0.7, r_cut=2.5, dt = 0.005, buffer = 0.4,
                  rho=0.5, period = 500, log_file=f'{part}_log0.7.gsd')
    try:
        lj.run(steps)
        pos = lj.get_positions()
        pos_file = f'{part}_pos_log0.7.csv'
        numpy.savetxt(pos_file, pos, delimiter=',')
    except Exception as e:
        # Print the exception and stack trace
        print(f"An error occurred: {e}")
        traceback.print_exc()
        print(f"Error on time step {lj.sim.timestep}")
    end_time = time.time()
    print(f"Elapsed time: {(end_time - start_time)*1000/steps:.5f} seconds/1000 steps")
    print(f"Elapsed time: {(end_time - start_time):.5f} seconds")
```

```
del lj
```

```
particles 3375
Elapsed time: 2.24137 seconds/1000 steps
Elapsed time: 2241.37302 seconds
```

Evaluation

Auto correlation

In [3]:
```
def autocorr1D(array):
    ft = numpy.fft.rfft(array - numpy.average(array))
    acorr = numpy.fft.irfft(ft * numpy.conjugate(ft)) / (len(array) * numpy.var(array))
    return acorr[0 : len(acorr) // 2]

def find_sample_distance(array):
    acorr = autocorr1D(array)
    try:
        loc = numpy.where(acorr < 0)[0][0]
    except:
        loc = len(array)
    return loc

def find_independent_samples(array):
    length = len(array)
    dist = find_sample_distance(array)
    return length // dist

def find_max_independent_samples(array, skip = 100, relax_time = 0):
    array = array[relax_time:]
    samples_keep = []
    lengths = []
    max_samples = 0
    start = 0
    for i in range(int(len(array)//skip-1)):
        samples = find_independent_samples(array[i*skip:])
        # For Graphing
        samples_keep.append(samples)
        lengths.append(len(array[i*skip:]))
```

```
        if samples > max_samples:
            max_samples = samples
            start = i*skip

    if max_samples == 0:
        print('No samples')
        start = 0
        max_samples = 1

    #plt.plot(lengths, samples_keep)
    #plt.xlabel('Array Length')
    #plt.ylabel('Samples')
    #plt.show()
    jump = len(array[start:]) // max_samples
    return start+relax_time, max_samples, jump
```

Evaluation Functions

In [4]:

```
def rdf(positions, bins = 100, r_max = 3.0, rho = 0.5):
    L = (len(positions) / rho) ** (1/3)

    box = freud.box.Box.cube(L)

    rdf = freud.density.RDF(bins=bins, r_max=r_max)
    rdf.compute(system=(box, positions))
    r, g_r = rdf.bin_centers, rdf.rdf
    return r, g_r

def get_temp(KE, N, kb = 1):
    return numpy.mean(2*KE)/(N*3-4)

def heat_capacity(energies, T, N,kb=1):
    E_mean = numpy.mean(energies)
    E_sq_mean = numpy.mean(numpy.array(energies) ** 2)

    Cv = (E_sq_mean - E_mean**2) / (kb * T**2) / N
    return Cv

def get_thermo(file):
    # Open the GSD file
```

```
    traj = gsd.hoomd.open(file, 'r')

    # Extract potential energy from all frames
    U = [float(frame.log['md/compute/ThermodynamicQuantities/potential_energy'][0]) for frame in traj]
    KE = [float(frame.log['md/compute/ThermodynamicQuantities/kinetic_energy'][0]) for frame in traj]
    P = [float(frame.log['md/compute/ThermodynamicQuantities/pressure'][0]) for frame in traj]
    E = numpy.array(KE) + numpy.array(U)
    t = [int(frame.configuration.step) for frame in traj]
    return t, KE, U, E, P

def plot(file, N = 1, relax_time = 0):
    # Calculate RDF
    pos_file = 'pos_' + file
    pos_file = pos_file[:-3] + 'csv'
    positions = numpy.loadtxt(pos_file, delimiter=",", skiprows=0)
    r, g_r = rdf(positions)

    t, KE, U, E, P = get_thermo(file)
    KE = numpy.array(KE[relax_time:]) / N
    U = numpy.array(U[relax_time:]) / N
    E = numpy.array(E[relax_time:]) / N
    P = P[relax_time:]
    t = t[relax_time:]
    # Create 2x2 subplots
    fig, axs = plt.subplots(2, 2, figsize=(10, 8))

    # Plot 1: Radial Distribution Function (RDF)
    axs[0, 0].plot(r, g_r, color='blue')
    axs[0, 0].set_xlabel('r')
    axs[0, 0].set_ylabel('g(r)')
    axs[0, 0].set_title('Radial Distribution Function')

    # Plot 2: KE and U with twinx
    ax1 = axs[0, 1]
    ax2 = ax1.twinx()
    ax1.plot(t, KE, color='red', label='KE')
    ax2.plot(t, U, color='green', label='U')
    ax1.set_xlabel('t')
    ax1.set_ylabel('KE', color='red')
    ax2.set_ylabel('U', color='green')
    ax1.set_title('KE and U vs Time')
```

```
    # Plot 3: Pressure vs Time
    axs[1, 0].plot(t, P, color='purple')
    axs[1, 0].set_xlabel('t')
    axs[1, 0].set_ylabel('P')
    axs[1, 0].set_title('Pressure vs Time')

    # Plot 4: Total Energy vs Time
    axs[1, 1].plot(t, E, color='orange')
    axs[1, 1].set_xlabel('t')
    axs[1, 1].set_ylabel('Total Energy')
    axs[1, 1].set_title('Total Energy vs Time')

    # Adjust layout and show plot
    plt.suptitle(file)
    plt.tight_layout()
    plt.show()
```

Evaluation

In [5]:

```
#files = ['0.1','0.5','0.6','0.7','0.75', '0.8','0.9', '1','1.25', '1.5', '2']
files = ['0.1','0.2','0.3','0.4','0.5','0.6','0.7','0.75', '0.8','0.9','0.95', '1','1.25', '1.5','1.75', '2']
#files = ['0.3','0.75', '1.25']
replicas = 12

# Loop through all files in the directory
relax_time = 100
N = replicas**3
data = {}
for file in files:

    data[file] = {}
    filename = 'log' + file + '.gsd'
    t, KE, U, E, P = get_thermo(filename)
    T = get_temp(numpy.array(KE[relax_time:]), N)

    # Calculate RDF
    pos_file = 'pos_' + filename
    pos_file = pos_file[:-3] + 'csv'
    positions = numpy.loadtxt(pos_file, delimiter=",", skiprows=0)
    r, g_r = rdf(positions)
    data[file]['r'] = r
```

```
    data[file]['g_r'] = g_r

    data[file]['t'] = t
    data[file]['KE'] = KE
    data[file]['U'] = U
    data[file]['E'] = E
    data[file]['P'] = P
    data[file]['T'] = T
    data[file]['N'] = len(positions)

    start, samples, jump = find_max_independent_samples(E, skip = 10, relax_time = relax_time)
    data[file]['relax_time'] = start
    data[file]['samples'] = samples
    data[file]['jump'] = jump

    data[file]['E_avg'] = numpy.mean(E[start:]) / N
    data[file]['E_std'] = numpy.std(E[start:]) / N
    data[file]['P_avg'] = numpy.mean(P[start:])
    data[file]['P_std'] = numpy.std(P[start:])
    data[file]['KE_avg'] = numpy.mean(KE[start:]) / N
    data[file]['KE_std'] = numpy.std(KE[start:]) / N
    data[file]['U_avg'] = numpy.mean(U[start:]) / N
    data[file]['U_std'] = numpy.std(U[start:]) / N

    Cv = []
    for i in range(jump):
        Cv.append(heat_capacity(E[start+i::jump], T, N))
    data[file]['Cv'] = Cv
    data[file]['Cv_avg'] = numpy.mean(Cv)
    data[file]['Cv_std'] = numpy.std(Cv)



    print(f'Number of samples {data[file]['samples']};   Start {data[file]['relax_time'] * 500};   Jump {data[file][
    print(f'T set: {file}, T measured: {data[file]['T']:.3g}')
    print(f'Energy per particle {data[file]['E_avg']:.3f} ± {data[file]['E_std']:.3f}')
    print(f'Pressure {data[file]['P_avg']:.3f} ± {data[file]['P_std']:.3f}')
    print(f'Heat Capacity: {data[file]['Cv_avg']:.2f} ± {data[file]['Cv_std']:.2f}\n')
    #plot(filename, N, relax_time)
```

```
Number of samples 286;   Start 285000;   Jump 5
T set: 0.1, T measured: 0.1
Energy per particle -6.539 ± 0.004
Pressure -0.148 ± 0.025
Heat Capacity: 3.30 ± 0.12

Number of samples 194;   Start 515000;   Jump 5
T set: 0.2, T measured: 0.2
Energy per particle -6.298 ± 0.009
Pressure -0.250 ± 0.034
Heat Capacity: 3.22 ± 0.33

Number of samples 175;   Start 825000;   Jump 2
T set: 0.3, T measured: 0.3
Energy per particle -6.085 ± 0.013
Pressure -0.337 ± 0.040
Heat Capacity: 3.13 ± 0.24

Number of samples 78;   Start 375000;   Jump 16
T set: 0.4, T measured: 0.4
Energy per particle -5.880 ± 0.018
Pressure -0.350 ± 0.046
Heat Capacity: 3.62 ± 0.68

Number of samples 308;   Start 75000;   Jump 6
T set: 0.5, T measured: 0.5
Energy per particle -4.807 ± 0.028
Pressure -0.086 ± 0.054
Heat Capacity: 5.35 ± 0.30

Number of samples 146;   Start 50000;   Jump 13
T set: 0.6, T measured: 0.6
Energy per particle -4.256 ± 0.034
Pressure -0.068 ± 0.055
Heat Capacity: 5.44 ± 0.58

Number of samples 231;   Start 75000;   Jump 8
T set: 0.7, T measured: 0.7
Energy per particle -3.678 ± 0.042
Pressure -0.047 ± 0.056
Heat Capacity: 6.10 ± 0.23
```

```
Number of samples 126;   Start 50000;   Jump 15
T set: 0.75, T measured: 0.75
Energy per particle -3.304 ± 0.044
Pressure -0.101 ± 0.059
Heat Capacity: 5.80 ± 0.52

Number of samples 207;   Start 65000;   Jump 9
T set: 0.8, T measured: 0.8
Energy per particle -3.013 ± 0.046
Pressure -0.081 ± 0.058
Heat Capacity: 5.67 ± 0.39

Number of samples 117;   Start 355000;   Jump 11
T set: 0.9, T measured: 0.9
Energy per particle -2.404 ± 0.055
Pressure -0.037 ± 0.057
Heat Capacity: 6.42 ± 0.61

Number of samples 61;   Start 50000;   Jump 31
T set: 0.95, T measured: 0.95
Energy per particle -2.079 ± 0.059
Pressure -0.017 ± 0.055
Heat Capacity: 6.51 ± 0.82

Number of samples 100;   Start 95000;   Jump 18
T set: 1, T measured: 1
Energy per particle -1.861 ± 0.042
Pressure 0.023 ± 0.056
Heat Capacity: 3.09 ± 0.49

Number of samples 633;   Start 50000;   Jump 3
T set: 1.25, T measured: 1.25
Energy per particle -1.287 ± 0.041
Pressure 0.340 ± 0.062
Heat Capacity: 1.90 ± 0.02

Number of samples 950;   Start 50000;   Jump 2
T set: 1.5, T measured: 1.5
Energy per particle -0.802 ± 0.048
Pressure 0.676 ± 0.068
Heat Capacity: 1.76 ± 0.03
```

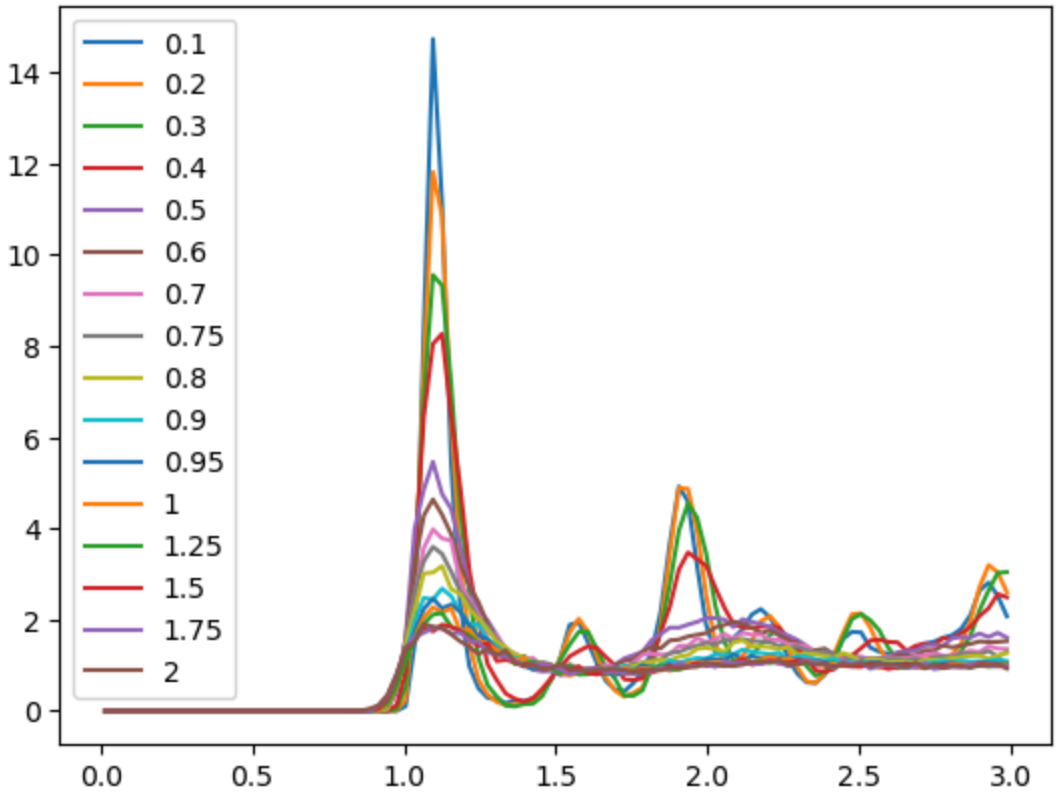
```
Number of samples 950;   Start 50000;   Jump 2
T set: 1.75, T measured: 1.75
Energy per particle -0.338 ± 0.055
Pressure 1.012 ± 0.077
Heat Capacity: 1.72 ± 0.05

Number of samples 950;   Start 50000;   Jump 2
T set: 2, T measured: 2
Energy per particle 0.118 ± 0.063
Pressure 1.337 ± 0.086
Heat Capacity: 1.70 ± 0.05
```

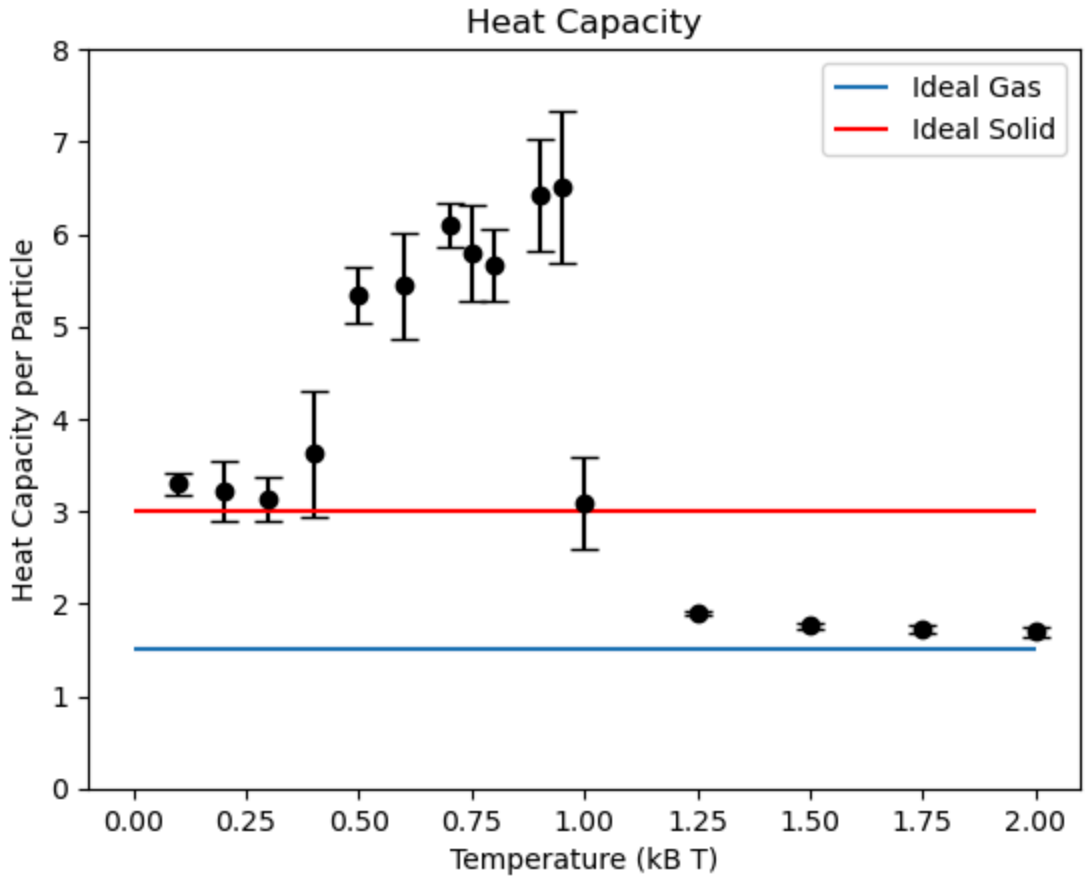
In [6]:

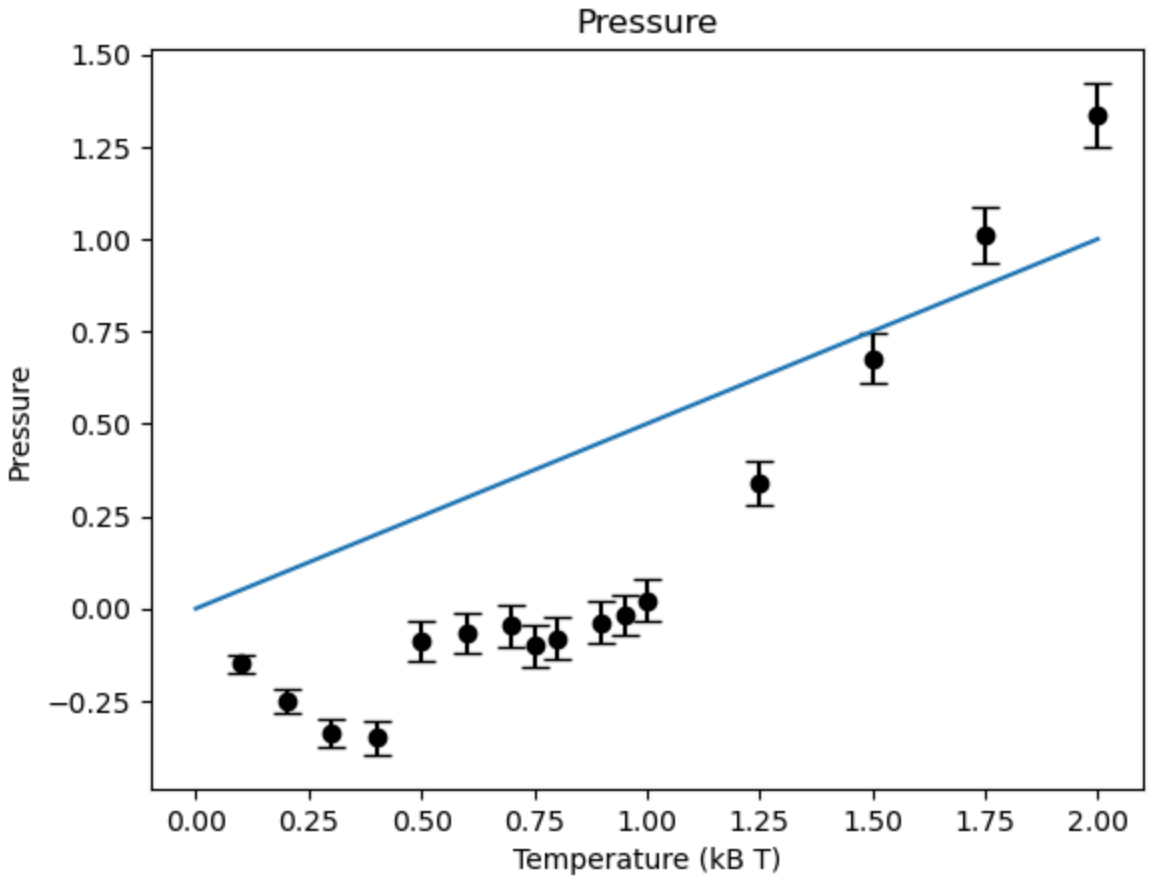
```python
files = ['0.1','0.2','0.3','0.4','0.5','0.6','0.7','0.75', '0.8','0.9','0.95', '1','1.25', '1.5','1.75', '2']
#files = ['0.1','0.5','0.6','0.7','0.75', '0.8','0.9', '1','1.25', '1.5', '2']
#files = ['0.3','0.75', '1.25']

for file in files:
    plt.plot(data[file]['r'], data[file]['g_r'], label=file)
plt.legend()
plt.show()

temp = numpy.linspace(0,2,100)

for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['Cv_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['Cv_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.xlabel('Temperature (kB T)')
plt.ylabel('Heat Capacity per Particle')
plt.title('Heat Capacity')
plt.hlines(1.5, 0,2, label = 'Ideal Gas')
plt.hlines(3, 0,2, label = 'Ideal Solid', color = 'red')
plt.ylim(0,8)
plt.legend()
plt.show()
```

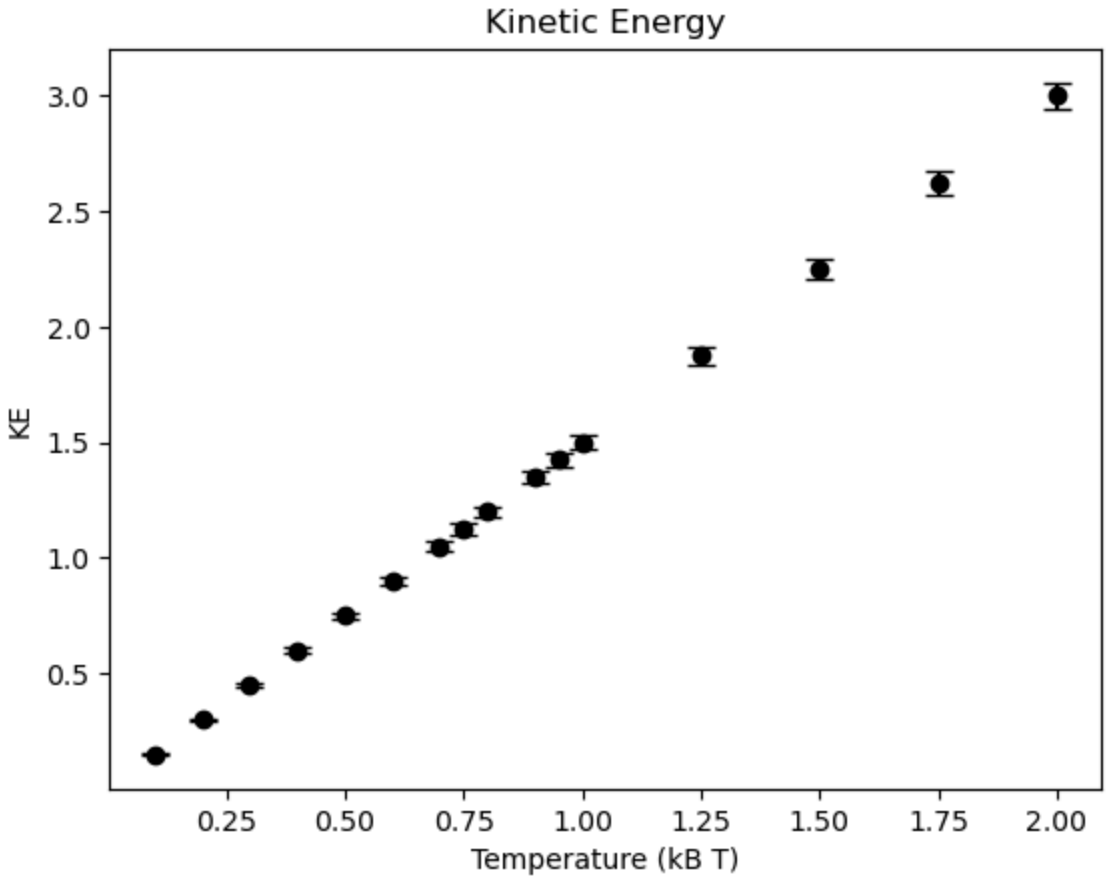
```python
# Pressure

P_ideal = temp * 0.5 * 1

for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['P_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['P_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.plot(temp, P_ideal)
plt.xlabel('Temperature (kB T)')
plt.ylabel('Pressure')
plt.title('Pressure')
plt.show()

# KE
for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['KE_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['KE_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.xlabel('Temperature (kB T)')
plt.ylabel('KE')
plt.title('Kinetic Energy')
plt.show()

# PE
for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['U_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['U_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
```

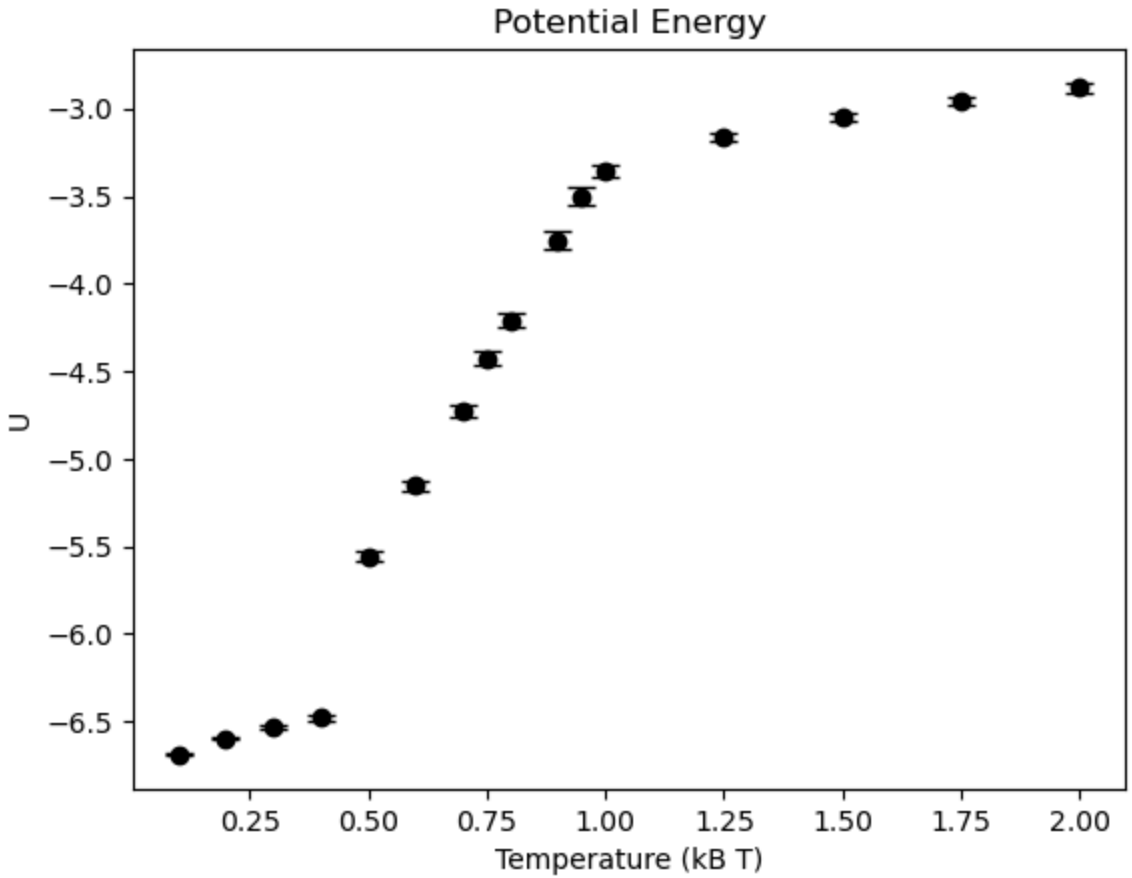
```
        color = 'k'
    )
plt.xlabel('Temperature (kB T)')
plt.ylabel('U')
plt.title('Potential Energy')
plt.show()

# E
for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['E_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['E_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.xlabel('Temperature (kB T)')
plt.ylabel('E')
plt.title('Total Energy')
plt.show()
```

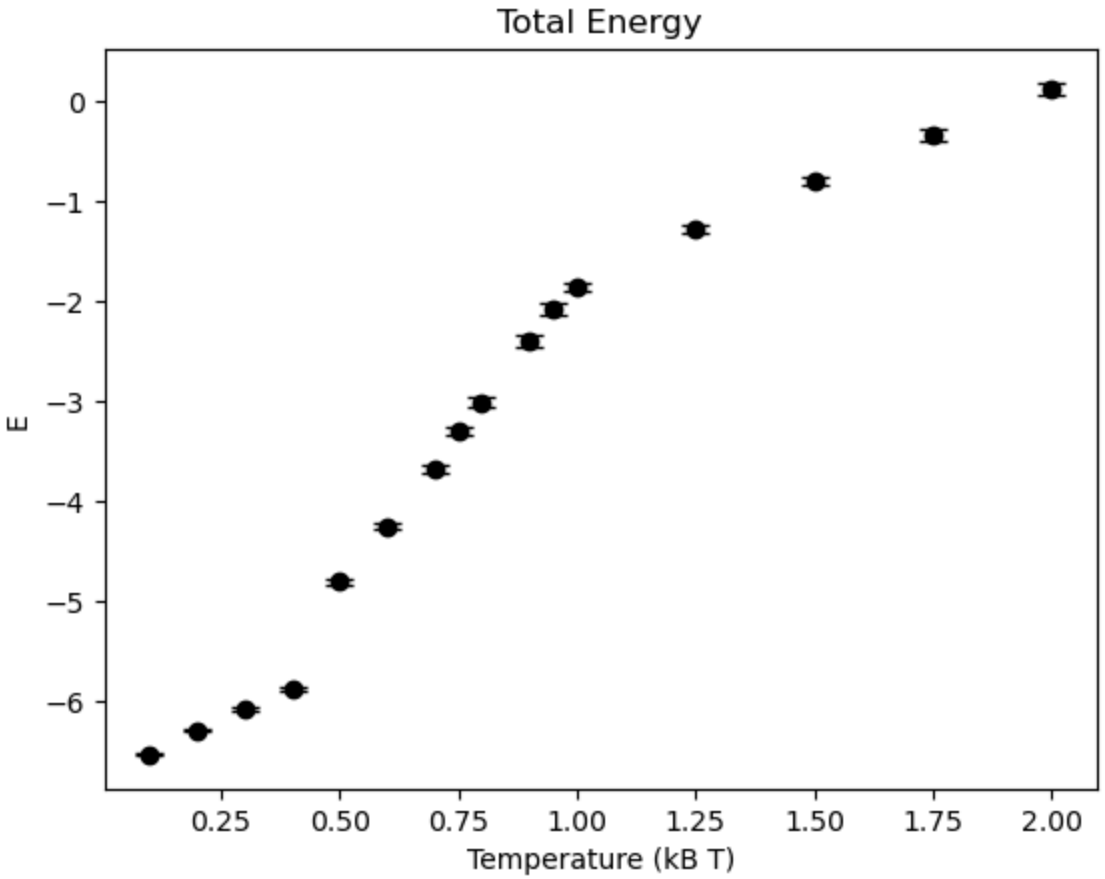

0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.75
0.8
0.9
0.95
1
1.25
1.5
1.75
2
0
2
4
6
8
10
12
14
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Heat Capacity
Ideal Gas
Ideal Solid
Heat Capacity per Particle
Temperature (kB T)
0
1
2
3
4
5
6
7
8
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Pressure
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
Pressure
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Temperature (kB T)


Kinetic Energy
KE
Temperature (kB T)
3.0
2.5
2.0
1.5
1.0
0.5
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Potential Energy
−3.0
−3.5
−4.0
−4.5
−5.0
−5.5
−6.0
−6.5
U
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Temperature (kB T)



Evaluation of Size

In [8]:
```
files = ['5','6', '7', '8', '9', '10', '11', '15']
replicas = 12

# Loop through all files in the directory
relax_time = 100

for file in files:
    N = int(file)**3
    data[file] = {}
    filename = file + '_log0.7.gsd'
    t, KE, U, E, P = get_thermo(filename)
```

```
    T = get_temp(numpy.array(KE[relax_time:]), N)

    # Calculate RDF
    pos_file = file + '_pos_log0.7.gsd'
    pos_file = pos_file[:-3] + 'csv'
    positions = numpy.loadtxt(pos_file, delimiter=",", skiprows=0)
    r, g_r = rdf(positions, r_max = 3)
    data[file]['r'] = r
    data[file]['g_r'] = g_r

    data[file]['t'] = t
    data[file]['KE'] = KE
    data[file]['U'] = U
    data[file]['E'] = E
    data[file]['P'] = P
    data[file]['T'] = T
    data[file]['N'] = len(positions)

    start, samples, jump = find_max_independent_samples(E, skip = 10, relax_time = relax_time)
    data[file]['relax_time'] = start
    data[file]['samples'] = samples
    data[file]['jump'] = jump

    data[file]['E_avg'] = numpy.mean(E[start:]) / N
    data[file]['E_std'] = numpy.std(E[start:]) / N
    data[file]['P_avg'] = numpy.mean(P[start:])
    data[file]['P_std'] = numpy.std(P[start:])
    data[file]['KE_avg'] = numpy.mean(KE[start:]) / N
    data[file]['KE_std'] = numpy.std(KE[start:]) / N
    data[file]['U_avg'] = numpy.mean(U[start:]) / N
    data[file]['U_std'] = numpy.std(U[start:]) / N

    Cv = []
    for i in range(jump):
        Cv.append(heat_capacity(E[start+i::jump], T, N))
    data[file]['Cv'] = Cv
    data[file]['Cv_avg'] = numpy.mean(Cv)
    data[file]['Cv_std'] = numpy.std(Cv)



    print(f'Number of samples {data[file]['samples']};   Start {data[file]['relax_time'] * 500};   Jump {data[file][
```

```
    print(f'N set: 0.7, T measured: {data[file]['T']:.3g}')
    print(f'Particle Number: {int(file)**3}')
    print(f'Energy per particle {data[file]['E_avg']:.3f} ± {data[file]['E_std']:.3f}')
    print(f'Pressure {data[file]['P_avg']:.3f} ± {data[file]['P_std']:.3f}')
    print(f'Heat Capacity: {data[file]['Cv_avg']:.2f} ± {data[file]['Cv_std']:.2f}\n')
    #plot(filename, N, relax_time)
```

Number of samples 271;   Start 50000;   Jump 7
N set: 0.7, T measured: 0.703
Particle Number: 125
Energy per particle -2.796 ± 0.137
Pressure -0.291 ± 0.182
Heat Capacity: 4.75 ± 0.38

Number of samples 316;   Start 50000;   Jump 6
N set: 0.7, T measured: 0.7
Particle Number: 216
Energy per particle -3.007 ± 0.109
Pressure -0.228 ± 0.144
Heat Capacity: 5.26 ± 0.18

Number of samples 126;   Start 50000;   Jump 15
N set: 0.7, T measured: 0.7
Particle Number: 343
Energy per particle -3.170 ± 0.090
Pressure -0.189 ± 0.122
Heat Capacity: 5.60 ± 0.49

Number of samples 90;   Start 275000;   Jump 16
N set: 0.7, T measured: 0.7
Particle Number: 512
Energy per particle -3.311 ± 0.080
Pressure -0.151 ± 0.104
Heat Capacity: 6.60 ± 1.24

Number of samples 67;   Start 425000;   Jump 17
N set: 0.7, T measured: 0.7
Particle Number: 729
Energy per particle -3.428 ± 0.065
Pressure -0.121 ± 0.088
Heat Capacity: 6.21 ± 1.02

Number of samples 24;   Start 815000;   Jump 15
N set: 0.7, T measured: 0.7
Particle Number: 1000
Energy per particle -3.515 ± 0.057
Pressure -0.097 ± 0.080
Heat Capacity: 6.53 ± 1.80

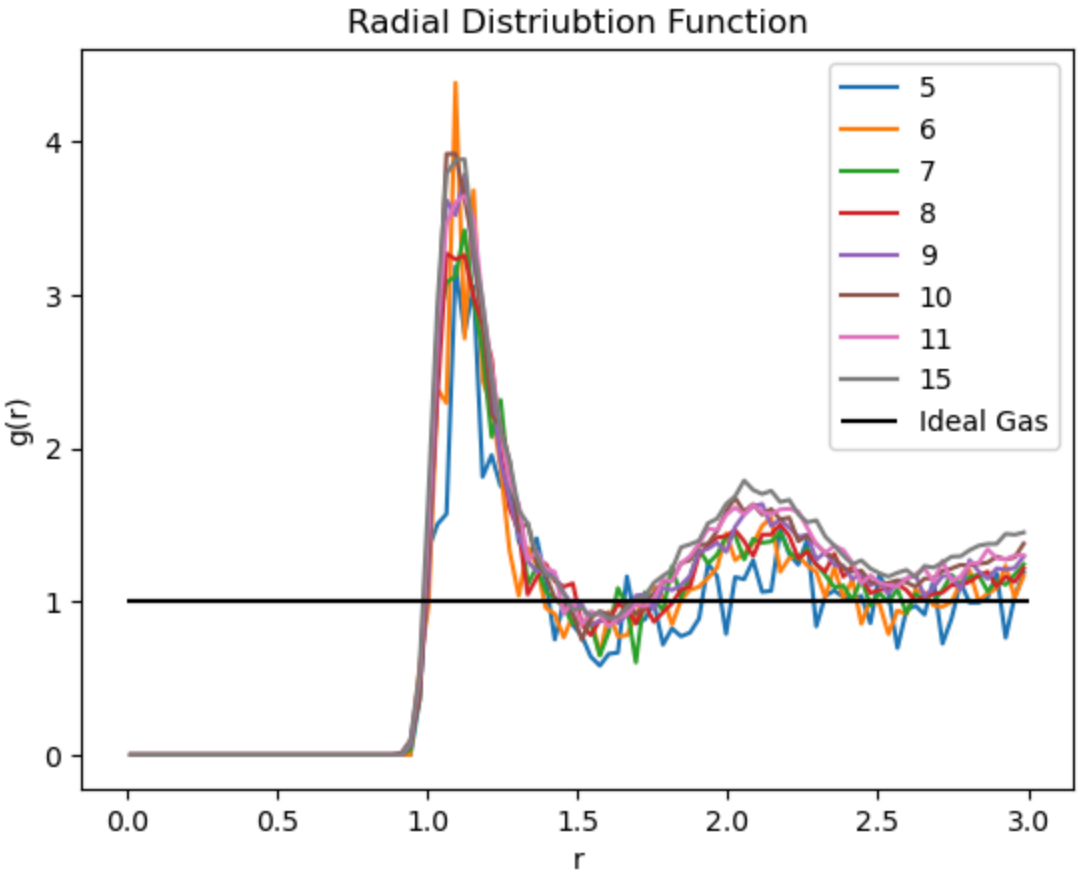
```
Number of samples 47;   Start 810000;   Jump 8
N set: 0.7, T measured: 0.7
Particle Number: 1331
Energy per particle -3.631 ± 0.046
Pressure -0.055 ± 0.062
Heat Capacity: 5.63 ± 1.05

Number of samples 23;   Start 965000;   Jump 3
N set: 0.7, T measured: 0.7
Particle Number: 3375
Energy per particle -3.777 ± 0.028
Pressure -0.034 ± 0.038
Heat Capacity: 5.27 ± 0.04
```

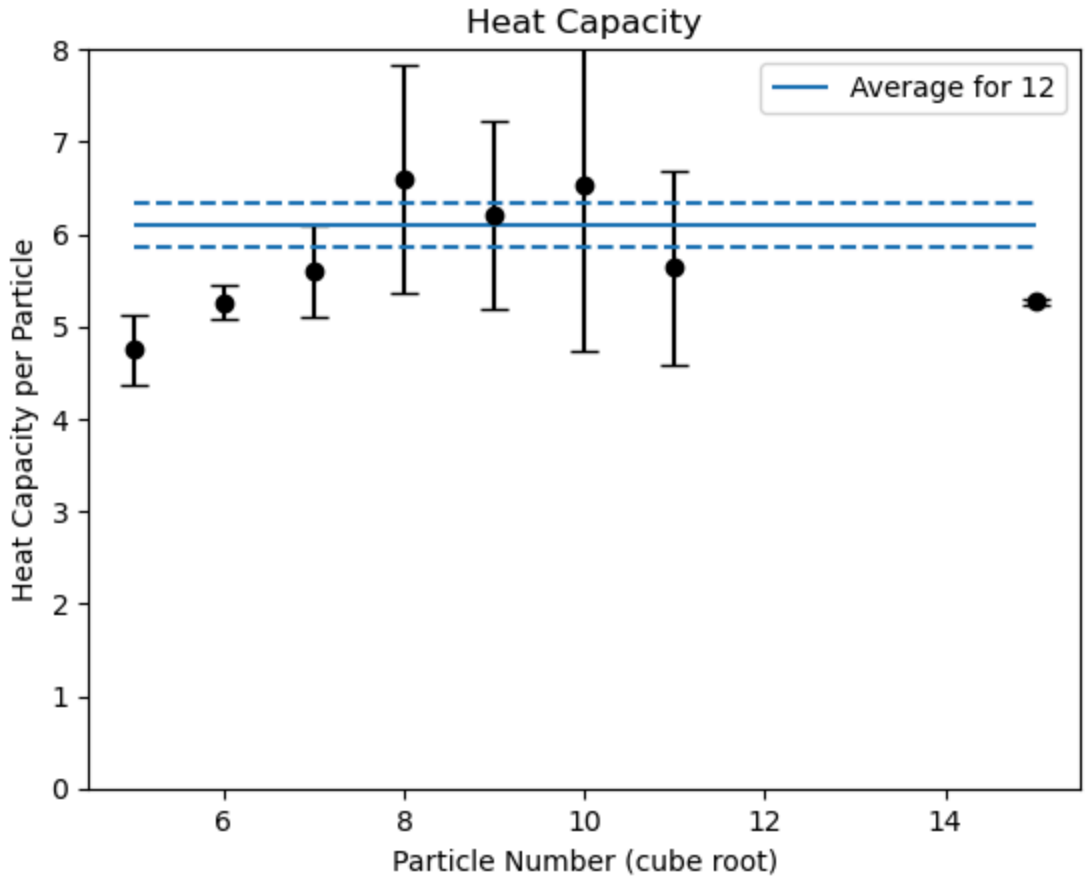
In [12]:

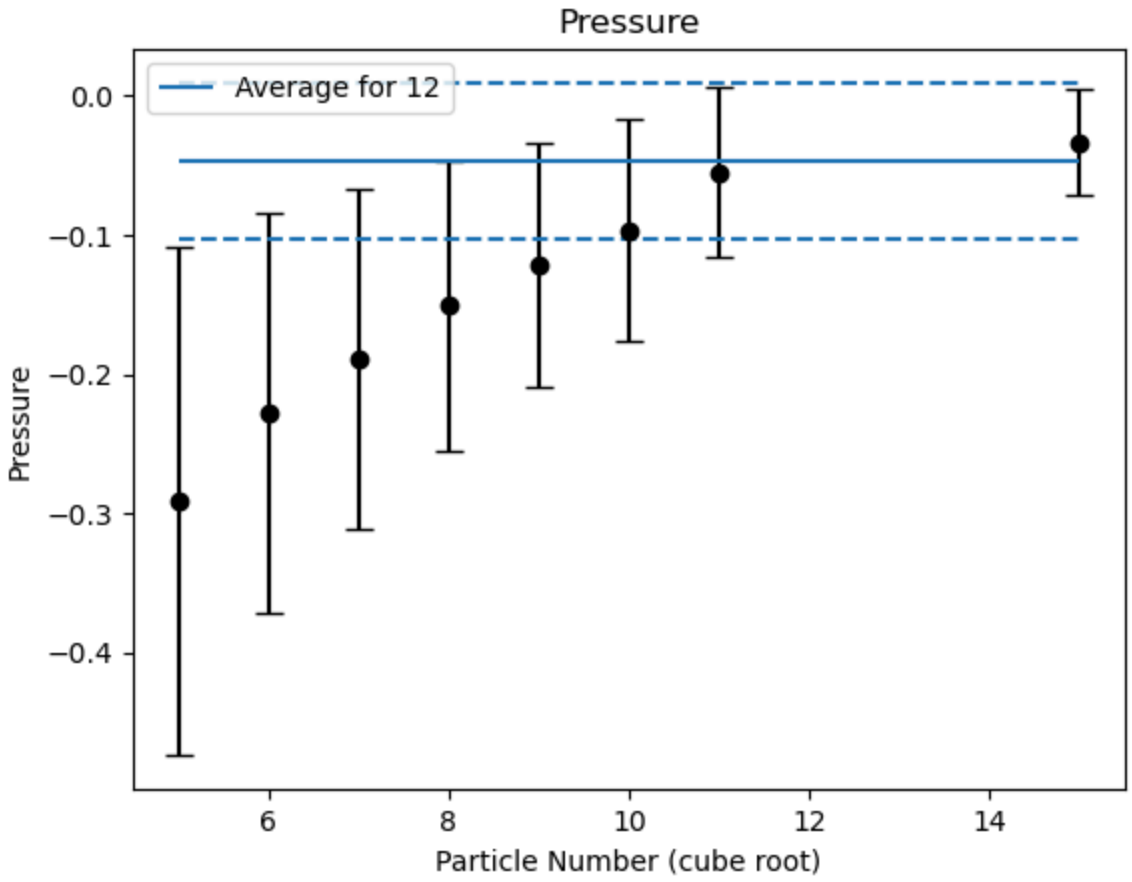
```python
files = ['5','6', '7', '8', '9', '10', '11', '15']

for file in files:
    plt.plot(data[file]['r'], data[file]['g_r'], label=file)

plt.xlabel('r')
plt.ylabel('g(r)')
plt.title('Radial Distriubtion Function')
plt.hlines(1, 0, 3, label='Ideal Gas', color = 'k')
plt.legend()
plt.show()

temp = numpy.linspace(0,2,100)

for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['Cv_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['Cv_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.xlabel('Particle Number (cube root)')
plt.ylabel('Heat Capacity per Particle')
plt.title('Heat Capacity')
plt.hlines(data['0.7']['Cv_avg'], 5, 15, label ='Average for 12')
```

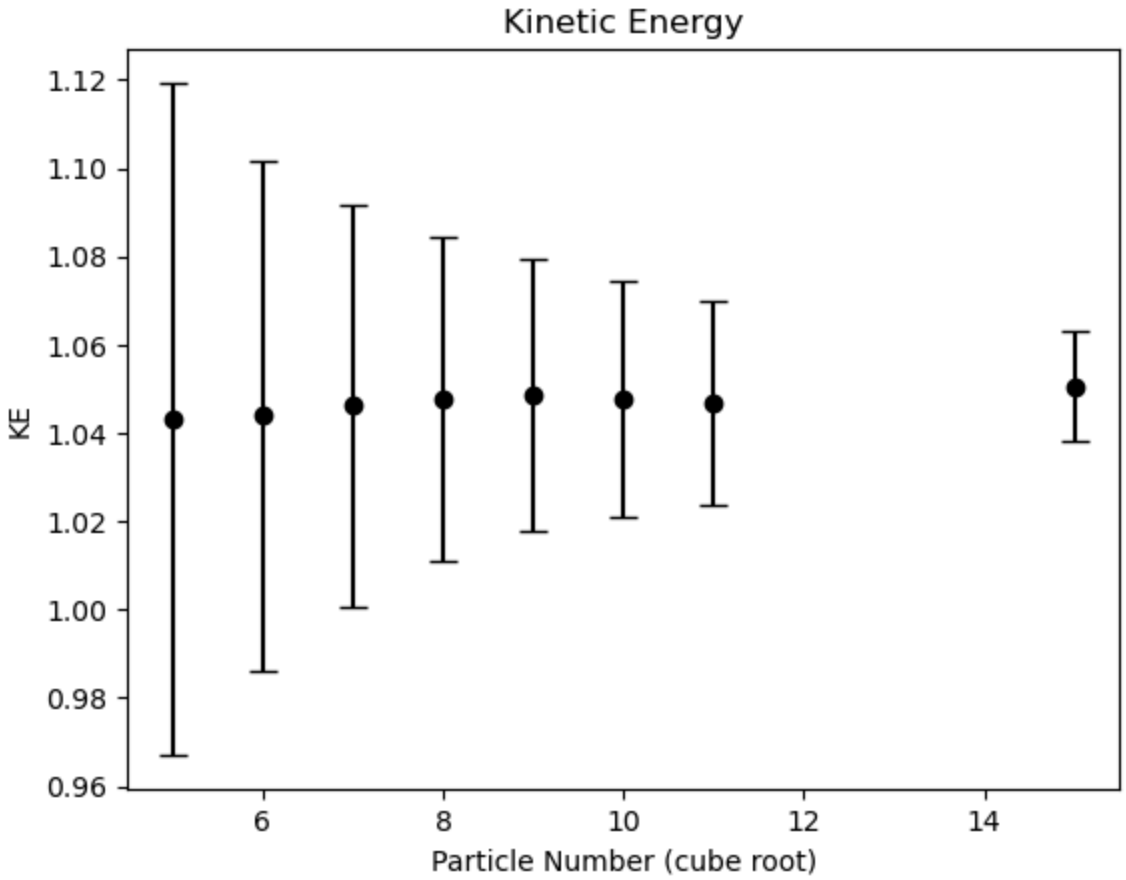
```
plt.hlines([data['0.7']['Cv_avg'] + data['0.7']['Cv_std'],data['0.7']['Cv_avg'] - data['0.7']['Cv_std']],
           5, 15, linestyles = 'dashed')
plt.ylim(0,8)
plt.legend()
plt.show()

# Pressure

P_ideal = temp * 0.5 * 1

for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['P_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['P_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.hlines(data['0.7']['P_avg'], 5, 15, label ='Average for 12')
plt.hlines([data['0.7']['P_avg'] + data['0.7']['P_std'],data['0.7']['P_avg'] - data['0.7']['P_std']],
           5, 15, linestyles = 'dashed')
plt.xlabel('Particle Number (cube root)')
plt.ylabel('Pressure')
plt.title('Pressure')
plt.legend()
plt.show()

# KE
for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['KE_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['KE_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.xlabel('Particle Number (cube root)')
plt.ylabel('KE')
plt.title('Kinetic Energy')
plt.show()
```

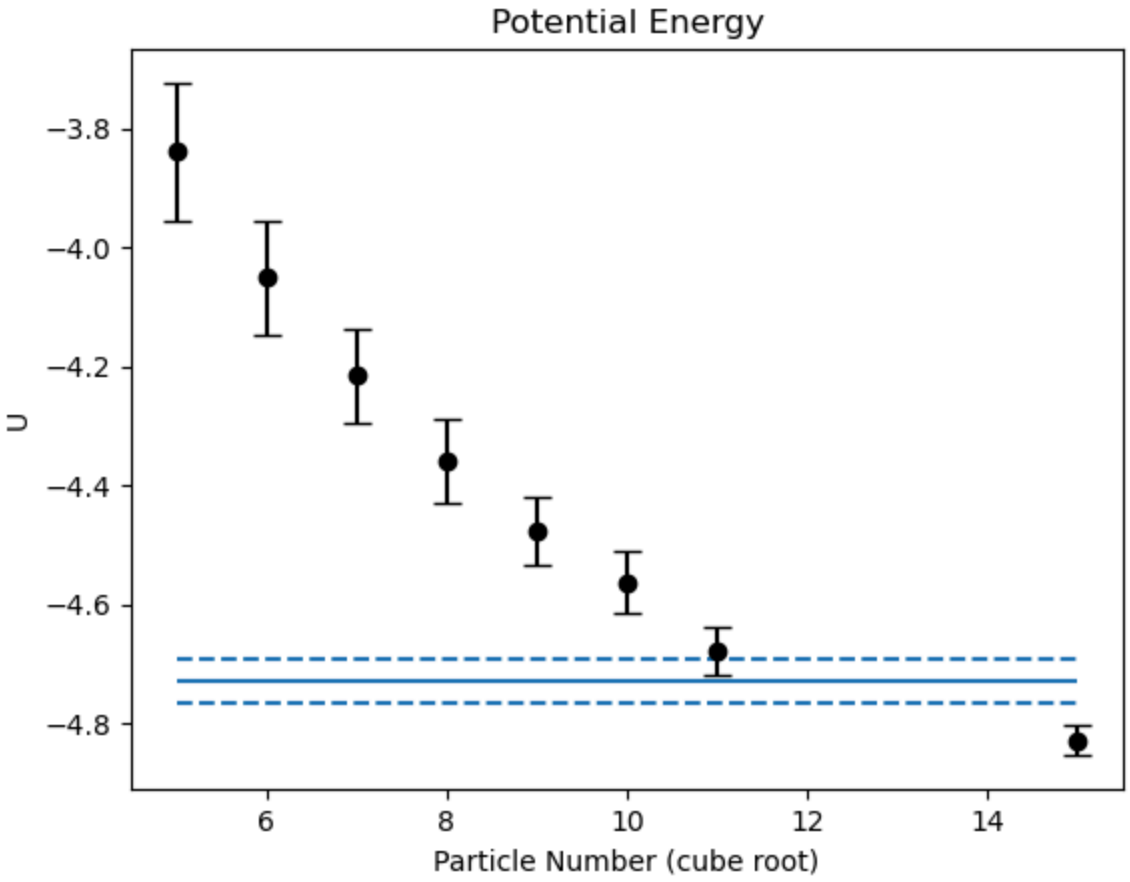
```python
# PE
for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['U_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['U_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.hlines(data['0.7']['U_avg'], 5, 15, label ='Average for 12')
plt.hlines([data['0.7']['U_avg'] + data['0.7']['U_std'],data['0.7']['U_avg'] - data['0.7']['U_std']],
           5, 15, linestyles = 'dashed')
plt.xlabel('Particle Number (cube root)')
plt.ylabel('U')
plt.title('Potential Energy')
plt.show()

# E
for file in files:
    plt.errorbar(
        float(file),  # X-axis: Temperature
        data[file]['E_avg'],  # Y-axis: Heat Capacity
        yerr=data[file]['E_std'],  # Error bars (standard deviation)
        fmt='o',  # 'o' makes it a scatter plot
        capsize=5,  # Adds small caps to error bars
        color = 'k'
    )
plt.hlines(data['0.7']['E_avg'], 5, 15, label ='Average for 12')
plt.hlines([data['0.7']['E_avg'] + data['0.7']['E_std'],data['0.7']['E_avg'] - data['0.7']['E_std']],
           5, 15, linestyles = 'dashed')
plt.xlabel('Particle Number (cube root)')
plt.ylabel('E')
plt.title('Total Energy')
plt.show()
```

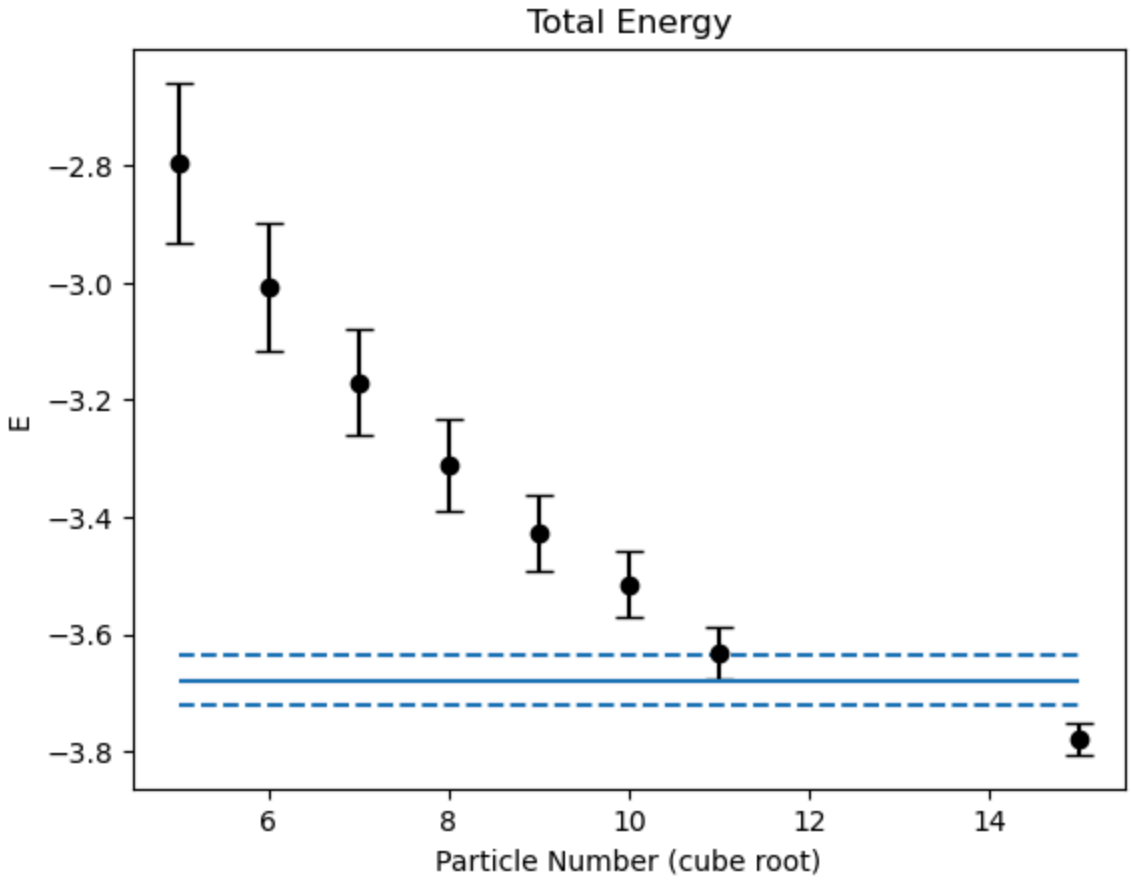

Radial Distriubtion Function
g(r)
r
0
1
2
3
4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
5
6
7
8
9
10
11
15
Ideal Gas


Heat Capacity
Average for 12
Heat Capacity per Particle
Particle Number (cube root)
0
1
2
3
4
5
6
7
8
6
8
10
12
14


Pressure
Average for 12
0.0
−0.1
−0.2
−0.3
−0.4
Pressure
6
8
10
12
14
Particle Number (cube root)


Kinetic Energy
KE
1.12
1.10
1.08
1.06
1.04
1.02
1.00
0.98
0.96
6
8
10
12
14
Particle Number (cube root)


Potential Energy
−3.8
−4.0
−4.2
−4.4
−4.6
−4.8
U
6
8
10
12
14
Particle Number (cube root)


Total Energy
−2.8
−3.0
−3.2
−3.4
−3.6
−3.8
E
6
8
10
12
14
Particle Number (cube root)

In [ ]:

In [ ]: