Hard Sphere Gas

Tyler Shendruk

March 23, 2010

1 Entropy

N hard spheres in a box. Each sphere excludes a volume ω. There is hard-core repulsion between spheres.

- What's the phase space availiable?
- What's the Hamiltonian, $\mathcal{H}$?

The Hamiltonian is

$$\mathcal{H} = \sum_{i=1}^{N} \left[\frac{\vec{p}_i^2}{2m} + U(\vec{q}_i \right] + \sum_{i,i} U_{ij} \tag{1}$$

Let's not have an imposed field and the iteraction potential is either infinite or zero.

So the number of states is

$$\Omega \propto \int_{\mathcal{H}=E} d^3q_1 d^3q_2 \dots d^3q_N d^3v_1 d^3v_2 \dots d^3v_N$$

As with an ideal gas, the momenta must lie on a hyper-sphere of radius

$$\sum_i |p_i| = \sqrt{2mE} \tag{2}$$

We do this quick, since you did it for an ideal gas. The surface area of a d-dimensional sphere is

$$\begin{aligned} A_d &= S_d R^{d-1} \\ &= \frac{2\pi^{d/2}}{(d/2-1)!} R^{d-1} \end{aligned} \tag{3}$$

So the allowed momentum states must combine to fall on that sphere when $d = 3N$ and the radius is $R = \sqrt{2mE}$. When you do this also remember to just add by hand the quantum statistical term $1/h^{3N}N!$.

$$\begin{aligned} \Omega &= \frac{1}{h^{3N}N!} \int_{\mathcal{H}=E} d^3q_1 d^3q_2 \dots d^3q_N d^3v_1 d^3v_2 \dots d^3v_N \\ &= \frac{1}{h^{3N}N!} \frac{2\pi^{3N/2}}{(3N/2-1)!} (2mE)^{(3N-1)/2} \int d^3q_1 d^3q_2 \dots d^3q_N \end{aligned} \tag{4}$$

Of course, for an ideal gas $\int d^3q_1 d^3q_2 \dots d^3q_N = V^N$ but now the position is limited due to the presence of the other spheres. We **approximate** this by placing them one after another:

- The first particle has volume V availiable
- The second particle has $V - \omega$
- The third particle has $V - 2\omega$
- *etc.*...
- The last particle has $V - (N-1)\omega$.

So then

$$\int d^3q_1 d^3q_2 \dots d^3q_N \approx \prod_{i=1}^{N} [V - (i-1)\omega] \tag{5}$$

We use the approximation (not a very great one) that

$$(V - a\omega)\,[V - (N-a)\omega] \approx \left(V - \frac{N\omega}{2}\right)^2 \tag{6}$$

So then the number of states availiable to a hard-core bead is

$$\Omega = \frac{1}{h^{3N} N!} \frac{2\pi^{3N/2}}{(3N/2-1)!} (2mE)^{(3N-1)/2} \left[V - \frac{N\omega}{2}\right]^N. \tag{7}$$

We take the limit where $N \gg 1$ then

$$\Omega = \frac{1}{h^{3N} N!} \frac{2\pi^{3N/2}}{(3N/2)!} (2mE)^{3N/2} \left[V - \frac{N\omega}{2}\right]^N. \tag{8}$$

and therefore the entropy is

$$\begin{aligned} S &= k_{\mathrm{B}} \ln \Omega \\ &= N k_{\mathrm{B}} \ln \left[\frac{e}{N}\left(V - \frac{N\omega}{2}\right)\left(\frac{4\pi m E e}{3Nh^2}\right)^{3/2}\right] \end{aligned} \tag{9}$$

2 Equation of State

From

$$\begin{aligned} dE &= \text{đ}Q + dW \\ &= TdS + \sum Jdx \\ &= TdS - PdV \end{aligned}$$

we must have

$$\frac{P}{T} = \left.\frac{\partial S}{\partial V}\right|_{E,N} \tag{10}$$

This gives us an equation of state

$$P\left(V - \frac{Nw}{2}\right) = N k_{\mathrm{B}} T \tag{11}$$

3 Maxwell Equation

Hard core gas with an equation of state $P(V - Nb) = Nk_{\mathrm{B}}T$ which has a C_V that is independent of T. Construct a Maxwell equation for $\frac{\partial S}{\partial V}\big|_{T,N}$. Start using Helmholtz free energy where we assume there is no change in species:

$$dF = -SdT - pdV \tag{12}$$

and by definition

$$dF = \left.\frac{\partial F}{\partial T}\right|_V dT + \left.\frac{\partial F}{\partial V}\right|_T dV \tag{13}$$

by comparison

$$-S = \left.\frac{\partial F}{\partial T}\right|_V dT \qquad \text{and} \qquad -p = \left.\frac{\partial F}{\partial V}\right|_T dV \tag{14}$$

and by equivalence of partials

$$\left.\frac{\partial S}{\partial V}\right|_T = \left.\frac{\partial p}{\partial T}\right|_V \tag{15}$$

Substituting the equation of state in for p we find

$$\left.\frac{\partial S}{\partial V}\right|_T = \left.\frac{\partial p}{\partial T}\right|_V = \frac{\partial}{\partial T}\left[\frac{Nk_{\mathrm{B}}T}{V - Nb}\right]_V$$

$$\boxed{\left.\frac{\partial S}{\partial V}\right|_T = \frac{Nk_{\mathrm{B}}}{V - Nb}} \tag{16}$$

4 Dependance

Show that E is a function of T (and N) only. To do this we will show that $\frac{\partial E}{\partial V}\big|_T = 0$. Start with the 1st Law and remember that we know $\frac{\partial S}{\partial V}\big|_T = \frac{Nk_{\mathrm{B}}}{V-Nb}$.

$$\begin{aligned} dE &= TdS - pdV \\ \left.\frac{\partial E}{\partial V}\right|_T &= T\left.\frac{\partial S}{\partial V}\right|_T - p \\ &= T\left.\frac{\partial p}{\partial T}\right|_V - p \\ &= T\frac{Nk_{\mathrm{B}}}{V - Nb} - p \\ &= p - p \end{aligned}$$

$$\boxed{\left.\frac{\partial E}{\partial V}\right|_T = 0} \tag{17}$$

And so $\therefore$ E depends only on T (and N): $E(V, T, N) \Rightarrow E(T, N)$.

5 Ratio

Show the ratio of heat capacities (γ) is $1 + Nk_{\mathrm{B}}/C_V$. To do this we start by considering C_V and substitute the first law as $đQ = dE - pdV$ into the definition.

$$C_V = \left.\frac{đQ}{dT}\right|_V = \left.\frac{dE - pdV}{dT}\right|_V = \left.\frac{dE - 0}{dT}\right|_V = \left.\frac{dE}{dT}\right|_V \tag{18}$$

$\therefore \quad dE = C_V dT$

In light of this, let's consider C_p

$$\begin{aligned} C_p &= \frac{đQ}{dT}\Big|_p \\ &= \frac{dE - pdV}{dT}\Big|_p \end{aligned}$$

But we substitute in $dE = C_V dT$

$$\begin{aligned} C_p &= \frac{C_V dT - pdV}{dT}\Big|_p \\ &= \frac{C_V dT}{dT}\Big|_p - \frac{pdV}{dT}\Big|_p \\ &= C_V - p\frac{dV}{dT}\Big|_p \end{aligned}$$

Use the equation of state ($V = Nk_{\mathrm{B}}T/p + Nb$) in the second term:

$$\begin{aligned} C_p &= C_V - p\frac{dV}{dT}\Big|_p \\ &= C_V - p\frac{d}{dT}\left[\frac{Nk_{\mathrm{B}}T}{p} + Nb\right]_p \\ &= C_V - p\frac{Nk_{\mathrm{B}}}{p} \\ &= C_V - Nk_{\mathrm{B}} \end{aligned}$$

Then

$$\boxed{\gamma = \frac{C_p}{C_V} = \frac{C_V - Nk_{\mathrm{B}}}{C_V} = 1 - \frac{Nk_{\mathrm{B}}}{C_V}} \tag{19}$$

6 Adiabatic Change

Show that an adiabatic change satisfies the equation $p(V - Nb)^\gamma =$ a constant.

$$dE = đQ - pdV$$

But if the process is adiabatic $đQ = 0$ and in the last section we showed that $dE = C_V dT$ for a hard sphere gas. $\therefore$

$$C_V dT = 0 - pdV$$

But from the equation of state $P = \frac{Nk_{\mathrm{B}}T}{V - Nb}$. $\therefore$

$$C_V dT + \frac{Nk_{\mathrm{B}}T}{V - Nb}dV = 0$$

$$C_V \frac{dT}{T} + Nk_{\mathrm{B}}\frac{dV}{V - Nb} = 0$$

Integrate this and remember from the last part of the question that $1 + k_{\mathrm{B}}N/C_V = \gamma$. Also notice that in the following K_I denotes the integration constitant and

K^* says that the integration constant has absorbed some other constant or constants or has been operated on in some way such that it remains an arbitary constant.

$$\begin{aligned} C_V \ln T + N k_{\mathrm{B}} \ln (V - Nb) &= K_I \\ \ln T + \left(\frac{N k_{\mathrm{B}}}{C_V} \right) \ln (V - Nb) &= K^* \\ \ln T + (\gamma - 1) \ln (V - Nb) &= K \\ \text{take exponent of all sides} & \\ T (V - Nb)^{\gamma - 1} &= K^* \end{aligned}$$

Now using the equation of state we can substitute in for T as $T = \frac{p(V-Nb)}{N k_{\mathrm{B}}}$.

$$\begin{aligned} T (V - Nb)^{\gamma - 1} &= K \\ p \frac{(V - Nb)}{N k_{\mathrm{B}}} (V - Nb)^{\gamma - 1} &= K \\ p (V - Nb) (V - Nb)^{\gamma - 1} &= K^* \end{aligned}$$

$$\boxed{p (V - Nb)^{\gamma} = K} \tag{20}$$

7 Algorithm

7.1 Initialize Position

We must randomly place the particles in the box:

7.1.1 Nieve Approach

```
void placeParticles(double position[][3]) {
  /*
    This routine randomly places the particles
  */
  int i,j,check = 0;
  int precision = 1E8;
  double dist;

  srand((unsigned)time(NULL));
  do {
    check = 0;
    for(i=0;i<particlePop;i++) {
      position[i][0] = ((double) (rand()%precision) / (double)precision) * sideLength;
      position[i][1] = ((double) (rand()%precision) / (double)precision) * sideLength;
      position[i][2] = ((double) (rand()%precision) / (double)precision) * sideLength;
    }
    for(j=0;j<i;j++) {
      dist = (position[i][0]-position[j][0])*(position[i][0]-position[j][0]);
      dist = dist + (position[i][1]-position[j][1])*(position[i][1]-position[j][1]);
      dist = dist + (position[i][2]-position[j][2])*(position[i][2]-position[j][2]);
      dist = sqrt(dist);
      if(dist < 2.*radius) check = 1;
    }
    printf("Check = %d\n",check);
  } while(check);
}
```

7.1.2 Improved

```
void placeParticles(double position[][3]) {
  /*
    This routine randomly places the particles
  */
  int i,j,check = 0;
  double dist;

  do {
    check = 0;
    for(i=0;i<particlePop;i++) {
      position[i][0] = doubleRand() * (sideLength-2.*radius)+radius;
      position[i][1] = doubleRand() * (sideLength-2.*radius)+radius;
      position[i][2] = doubleRand() * (sideLength-2.*radius)+radius;
    }
    for(j=0;j<i;j++) {
      dist = (position[i][0]-position[j][0])*(position[i][0]-position[j][0]);
      dist = dist + (position[i][1]-position[j][1])*(position[i][1]-position[j][1]);
      dist = dist + (position[i][2]-position[j][2])*(position[i][2]-position[j][2]);
      dist = sqrt(dist);
      if(dist < 2.*radius) check = 1;
    }
  } while(check);
}
```

Whereas the previous algorithm only made sure the centres were in the box, this algorithm ensures all the particles are in the box.

We expect every legal configuration to occur with the same probability. Here we generated both legal and illegal configurations. **But** notice we didn't just reject illegal placements. We rejected the entire configuration. You may be tempted to just replace the illegal particle That's called `random deposition` which I hear is useful for models of adhesion *etc.* but we're looking for equalibrium.

7.2 Velocity

The velocity initialization doesn't have this. Just like an ideal gas they must exist on a dN-dimensional sphere of radius $\sqrt{2E/m}$ where E is the total energy of the system. The total energy is related to the temperature by

$$\frac{E}{dN} = \frac{1}{2}k_{\mathrm{B}}T \tag{21}$$

Each of the particles is given a random velocity drawn from a Gaussian distribution with standard deviation

$$\sigma = \sqrt{\frac{2}{m}\frac{E}{dN}} \tag{22}$$

The routine that initializes the velocity of each particle is

```
void placeVelocities(double velocity[][3]) {
  /*
    This routine randomly sets the particles' velocities
  */
  int i,j,check = 0;
  double stdev;
```

```
  stdev = 2.*energy/(mass*3.*particlePop);
  stdev = sqrt(stdev);

  for(i=0;i<particlePop;i++) {
    velocity[i][0] = gaussRand() * stdev;
    velocity[i][1] = gaussRand() * stdev;
    velocity[i][2] = gaussRand() * stdev;
  }
}
```

The radius of the hyper-sphere **must** be $\sqrt{2mE}$ the previous code didn't demand that - although it was true on average. A better way is

```
void placeVelocities(double velocity[][3]) {
  /*
    This routine randomly sets the particles' velocities
  */
  int i,j,check = 0;
  double stdev,sqrtDim,sum;

  stdev = 2.*energy/(mass*3.*particlePop);
  stdev = sqrt(stdev);
  sum = 0.;
  sqrtDim = 1./sqrt(3.*(double)particlePop);

  for(i=0;i<particlePop;i++) {
    velocity[i][0] = gaussRand() * sqrtDim;
    velocity[i][1] = gaussRand() * sqrtDim;
    velocity[i][2] = gaussRand() * sqrtDim;
    sum = sum + velocity[i][0]*velocity[i][0] + velocity[i][1]*velocity[i][1] + velocity[i][2]*velocity[i][2];
  }

  sum = sqrt(sum);
  for(i=0;i<particlePop;i++) {
    velocity[i][0] = velocity[i][0]*stdev / sum;
    velocity[i][1] = velocity[i][1]*stdev / sum;
    velocity[i][2] = velocity[i][2]*stdev / sum;
  }
}
```

7.3 Streaming

Once they are in the box they just move like billiard balls. They go in straight lines without any force acting on them until a collision event occurs. At that point a pair of particles feel an impulse and instantaneously change velocity. Streaming the particles is simple since there is no potential:

```
void stream(double streamTime,double position[][3],double velocity[][3]) {
  int i;
  for(i=0,i<particlePop;i++) {
    position[i][0] = position[i][0] + streamTime*velocity[i][0];
    position[i][1] = position[i][1] + streamTime*velocity[i][1];
    position[i][2] = position[i][2] + streamTime*velocity[i][2];
}
```

7.4 Collision Time

To determine the time of the next pair collision we must go through the entire set of pairs, allowing those to to evolve and see if and when they will collide. A collision occurs between particles i, j when the distance between centres equals

twice the radius r of the spheres:

$$\begin{aligned}\vec{x}_i(t) - \vec{x}_j(t) &= \vec{x}_i(t_0) - \vec{x}_j(t_0) + (\vec{v}_i - \vec{v}_j)\,(t - t_0) \\ &= \Delta\vec{x} + \Delta\vec{v}\,(t - t_0) \\ &= 2r \end{aligned} \tag{23}$$

But we don't really feel like working with the vectors so we square the equation and solve for t

$$t_{1,2} = t_0 + \frac{-\left(\Delta x \cdot \Delta v\right) \pm \sqrt{\left(\Delta x \cdot \Delta v\right)^2 - \left(\Delta v\right)^2 \left[\left(\Delta x\right)^2 - 4r^2\right]}}{\left(\Delta v\right)^2}. \tag{24}$$

The routine for this looks like

```
double particleParticleTime(double x1[3],double v1[3],double x2[3],double v2[3])
{
  /*
    Returns time of collision between
    particle 1 and particle 2 - infinity is set to 2^500
  */
  double collisionTime = pow(2,500);
  double deltaX[3], deltaV[3], dotXX, dotVV, dotXV;
  double Y;
  int i;

  for(i=0;i<3;i++) {
    deltaX[i] = x2[i]-x1[i];
    deltaV[i] = v2[i]-v1[i];
  }
  dotXX = dot(deltaX,deltaX);
  dotVV = dot(deltaV,deltaV);
  dotXV = dot(deltaX,deltaV);
  Y = dotXV*dotXV - fabs(dotVV)*(fabs(dotXX)-4.*radius*radius);

  // The particles must be approaching each other: doxXV<0
  // and the square root must be real
  // and can't be zero
  if(Y > numPrecision && dotXV < 0.) {
    collisionTime = - (dotXV+sqrt(Y))/dotVV;

    // Reject negative times
    if( collisionTime < 0. ) {
      collisionTime = - (dotXV-sqrt(Y))/dotVV;
    }
  }
  else collisionTime = pow(2,500);

  return collisionTime;
}
```

The other thing we have to worry about is collisions with container walls. These are even easier to handle. We put our box in the positive octant. Then we just have to check if a particle's position is negative or greater than our box length. The routine is

```
double particleWallTime(double position[3],double velocity[3]) {
  /*
    Returns time of collision between a
    particle and a wall
```

```
  */
  double tempTime,collisionTime;
  int i;

  collisionTime=pow(2,500);

  // Zero wall
  for(i=0;i<3;i++) {
    tempTime = (0.+radius-position[i])/velocity[i];
    if(tempTime > numPrecision && tempTime < collisionTime ) collisionTime = tempTime;
  }
  // Other wall
  for(i=0;i<3;i++) {
    tempTime = (sideLength-radius-position[i])/velocity[i];
    if(tempTime > numPrecision && tempTime < collisionTime ) collisionTime = tempTime;
  }

  return collisionTime;
}
```

The collision time is just the smallest of all these times. The particles are all allowed to stream for that time and then the particles that were the first to collide must change their velocities.

7.5 Impact

The collisions are **elastic** and the surfaces are **smooth** and so it is easy to determine the new velocity by breaking the vectors up into those in the collision plane and perpendicular to it.

```
void particleParticleCollision(double x1[3],double v1[3],double x2[3],double v2[3])
{
  /*
    Computes the velocities of the balls after the collision
  */
  double norm[3] = {0};
  double deltaX[3] = {0};
  double deltaV[3] = {0};
  double changeV[3] = {0};
  double dotXX;
  int i;

  // Find normal vector
  for(i=0;i<3;i++) deltaX[i] = x2[i]-x1[i];
  dotXX = dot(deltaX,deltaX);
  for(i=0;i<3;i++) norm[i] = deltaX[i]/sqrt(dotXX);

  for(i=0;i<3;i++) deltaV[i] = v2[i]-v1[i];

  // New velocities
  for(i=0;i<3;i++) changeV[i] = norm[i]*dot(deltaV,norm);


  for(i=0;i<3;i++) {
    v1[i] = v1[i]+changeV[i];
    v2[i] = v2[i]-changeV[i];
  }
}

void particleWallCollision(double x[3],double v[3])
{
  int i,j;
```

```
  double norm[3]={0.};
  double test, dist = sideLength;

  // Zero wall
  for(i=0;i<3;i++) {
    test = x[i];
    if(test < dist ) {
      for(j=0;j<3;j++) norm[j] = 0.;
      norm[i] = -2.;
      dist = test;
    }
  }
  // Other wall
  for(i=0;i<3;i++) {
    test = sideLength-x[i];
    if(test < dist ) {
      for(j=0;j<3;j++) norm[j] = 0.;
      norm[i] = -2.;
      dist = test;
    }
  }
  // Alter the velocity
  for(i=0;i<3;i++) {
    v[i] = v[i] + norm[i]*v[i];
  }
}
```

7.6 Putting it all together

So we have an algorithm that

1. Places the particles
2. Sets velocities
3. Determines the time to the next pair collision
4. Determines the time to the next wall collision
5. Streams the particles till the next collision
6. Solves the implact problem
7. Returns to Step 3

In computer code this looks like

```
int main(int argc, char* argv[])
{
  double pairTime=0., wallTime=0.,collisionTime=0., runTime=0.;
  double tempTime;
  double position[particlePop][3],velocity[particlePop][3];
  int i,j,p1,p2,w1;
  int t;

  // Initialize random number generator
  srand((unsigned)time(NULL));
  // Initiialize positions
  placeParticles(position);
  placeVelocities(velocity);

  while(runTime < totalTime) {
```

```
    printf("t=%e\n",runTime);

    // Determine the time to the next pair collision
    // Loop through pairs
    pairTime=pow(2,500);
    for(i=0;i<particlePop;i++) for(j=i;j<particlePop;j++) {
tempTime = particleParticleTime(position[i],velocity[i],position[j],velocity[j]);
if(tempTime < pairTime && tempTime > numPrecision) {
  pairTime = tempTime;
  p1=i;
  p2=j;
}
    }

    // Determine the next wall collision
    wallTime=pow(2,500);
    for(i=0;i<particlePop;i++) {
      tempTime = particleWallTime(position[i],velocity[i]);
      if(tempTime < wallTime && tempTime > numPrecision) {
wallTime = tempTime;
w1=i;
      }
    }

    // The next collision is min(pairTime,wallTime)
    if (pairTime < wallTime) collisionTime = pairTime;
    else collisionTime = wallTime;

    // Stream till then
    stream(collisionTime,position, velocity);
    runTime = runTime + collisionTime;

    // Apply the collision
    if (pairTime < wallTime) {
      particleParticleCollision(position[p1],velocity[p1],position[p2],velocity[p2]);
    }
    else {
      particleWallCollision(position[w1],velocity[w1]);
    }

    for(i=0;i<particlePop;i++) for(j=0;j<3;j++) {
      if(position[i][j]<0. || position[i][j]>sideLength) {
        printf("Particle escaped box\n");
return 0;
      }
    }

  }

  return 0;
}
```

8 Practical Concerns

8.1 Random Numbers

Random numbers are generated by non-linear algorithms. Unfortunately, because the computer's memory is finite these are always periodic. Plus, they always need a seed. When chosing how to generate random numbers you often have to chose a balance between "randomness" and speed. Since we're not worried about it. I just used the compilers random number but turned it into a

double from and integer.

```
double doubleRand()
{
    return rand()/((double)(RAND_MAX));
}
```

But that returns a random number drawn from a uniform probability distribution. If we want to draw from a gaussian distribution there are ways to transform one distribution to another. The way to get a gaussian is by the Box-Muller transformation

```
float gaussRand(void){
/*
   Box-Muller transformation to turn a uniform random number
   between 0-1 into a gaussian distribution of mean 0 and
   StDev of 1.
*/
float x1,x2,w,y1,y2;
do {
x1=2.0*doubleRand()-1.0;
x2=2.0*doubleRand()-1.0;
w=x1*x1+x2*x2;
}while (w>=1.0);
w=sqrt((-2.0*log(w))/w);
y1=x1*w;
y2=x2*w;
return y1;
}
```

8.2 Stability

A very important practical concern is numerical stability. This kept me up all night. Trying to get this program to run before I assigned it to you. I kept getting collision times of zero, which caused a second collision to be initiated. The second collision sends the bead back towards the wall. This spirals out of control into infinite times and positions and other oddities.

Correction Based on the suggestions of Alex, I altered the `particleWallTime()` subroutine to include a check to make sure the particle is approaching the wall (as was already in the `particleParticleTime()` routine) and the stability issue was resolved.

9 Computational Experiments - Measuring Macroscopic Quantities

9.1 Output at Regular Time Intervals

The time step is not constant ecause of the way that the hard sphere algorithm works. We actually have to create a routine to calculate values at regular intervals. You can calculate what step you are at by finding the truncated value of the current time t by the time interval of each step δt (plus 1) *i.e.* step $n_{\min}$ is

$$n_{\min} = (int)\left(\frac{t}{\delta t}\right) + 1.$$

Multiple steps may happen between collisions so the first step in that interval is $n_{\min}$ and the last one before the collision time $t_{\rm coll}$ is

$$n_{\max} = (int)\left(\frac{t_{\rm coll}}{\delta t}\right).$$

9.2 Temperature

With a molecular picture of temperature, it is a very simple task to calculated the term $k_{\rm B}T$. Notice that since nothing in our numerical world has units, we would much rather deal with $k_{\rm B}T$ as a single variable. That's easy enough for most applications. The temperature of the system is defined by

$$k_{\rm B}T = \left(\frac{2}{\mathcal{D}N}\right)\sum_i \frac{1}{2}\left\langle mv_i^2\right\rangle \tag{25}$$

where N is the number of constituent molecules and $\mathcal{D}$ is the spatial dimension.

Potential Question: *Write a routine to measure the temperature at different times. Output a graph or table of it's values.*

Potential Question: *Comment on the expected natural fluctuations. Why does the gas behave this way? Imagine a gas model where those properties of the fluctuations would be different. Describe such a gas.*

9.3 Pressure

It is convinient for us that the particles are in a container with rigid walls. We have all found the pressure of a gas from basic kinetic theory before. For our numerical experiment, we simply don't average over all the particles. Rather we average over the time of our experiment.

When a particle collides with a wall (say the x-wall), it imparts an impulse to the wall

$$\Delta p = 2mv_x.$$

The total force on the wall over some observation time Δt is simply

$$F = \frac{\Delta p}{\Delta t}$$

and the pressure is the force per unit area. Remember you have six walls.

Potential Question: *Calculate the pressure of your gas.*

Potential Question: *How does the numerically measured pressure compare to expectations (based only on input values).*

9.4 Heat Capacity

From Eq. 18, we can determine the heat capacity. Since the heat capacity depends on the system size, it is best to use the specific heat capacity $c_V = C_V/N$.

Potential Question: *By plotting a variety of initial energies make a plot that determines the specific heat capacity, c_V.*

Potential Question: *How does this compare to an ideal gas? a diatomic gas?*

9.5 Partition Function

You would think that all the important physics happens in the algorithm as you run time. But this isn't true. Recall how we are throwing out initial configurations? Even this can tell us important physics. The acceptance rate is equivalent to the probability of having that configuration.

Potential Question: *Plot the acceptance rate against density (volume fraction, η). Discuss.*

The acceptance rate reflects the number of availiable configurations which in turn is proportional to the partition function, Z. The partition function for an ideal gas is something that you will find in the lectures:

$$Z_{\text{ideal}} = Z_0 = \int d\vec{x} = V^N.$$

The velocity distribution of a hard sphere gas remains unchanged from the ideal case and only the spatial component of the number of availiable states changes. This causes the partition function for an ideal gas to be

$$\begin{aligned} Z &= Z_0 p_{\text{acc}}(\eta) \\ &= V^N p_{\text{acc}}(\eta). \end{aligned}$$

9.6 Collision Rate

The collision rate is most likely the simplest thing for us to calculate and is very important in the material you skipped in the textbook (Kadar chapter 3).

Potential Question: *Plot collision rate vs density.*

9.7 Periodic Boundary Conditions

Potential Question: *How many particles would we need to simulate to reach the thermodynamic limit?*

An approximation on a gas in an infinite volume is obtained by periodic boundary conditions. The idea is this: a particle exits a side and is instantaneously teleported to the other side. Just like in Pacman.

Potential Question: *What is the topology of this space?*

Two things are needed to achieve periodic boundary conditions:

1. The particle must reside in the "central box". Computers can do this easily by the `mod` operation.
2. Each particle must have multiple position vectors and interactions must choose the correct one. The difference in positions is modulated and that difference must be used in the collision calculations.

Be aware that we have done this to remove the fact that we are not in the thermodynamic limit but we will be left with residual errors referred to as *finite size effects*. A particle may cause a disturbance to it's neighbours which propogates a large distance (hydrodynamic effects). It is possible that the length scale of those effects may be greater than the boxsize. If this is true than the particle will hydrodynamically interact with itself.

Potential Question: *How could you measure pressure in a system with no walls?*

9.8 Transport Properties

Pick any hard sphere. Imagine it's trajectory. When the gas is made up of large numbers of spheres, the "tagged" particle will undergo a random walk. The average distance that it is able to travel is related to it's diffusion coefficient, D. The mean square displacement is

$$R^2 = \left\langle (\vec{r}(t_2) - \vec{r}(t_2))^2 \right\rangle . \tag{26}$$

The average should be done over all the particles. It is related to the diffusion coefficient through the Einstein relation

$$R^2(t) = 2\mathcal{D}Dt \qquad t \to \infty \tag{27}$$

9.9 Histograms

Building probability distributions for things the the velocity, the speed or the density is a very important practice. The parent distributions are estimated by histograms which have been build up from bins. Plotting programs will often bin for you but the concept is quite simple:

1. Find the maximum and minimum value
2. Divide the interval by the number of bins you want to use
3. Go through the list of quantities. Each quantity will belong to a bin. Use an if statement or someother method to find out which bin.
4. Increment that bin.

Potential Question: *Find the x-velocity distribution.*

Potential Question: *Find the speed distribution.*

Potential Question: *Find the density distribution.*

9.10 Dimensionality

Potential Question: *What trivial modification is needed to simulate hard disks?*

Potential Question: *Are there any numbers that need to be changed throughout the code? What would be an easy patch to handle this?*