MECH 578 Project Report: 2D Gas Simulation

March 19, 2015

John LIAN 260408431

Step 1

In 2D, an area element in the velocity space can be defined using polar coordinates. That is:

$$dA = dC_x C_y = C dC d\theta. \tag{1}$$

Since we know the Maxwell distribution function for the 1D velocities as

$$f(C_x) = \left(\frac{m}{2\pi kT}\right)^{\frac{1}{2}} \exp\left[-\frac{m}{2kT}C_x^2\right], \tag{2}$$

it is possible to integrate this function over the 2D velocity space with the area element defined in Eq. 1 to find the number of particles that have velocity between C and $C + dC$:

$$\begin{aligned}
\chi(C) &= \int_0^{2\pi} \left(\frac{m}{2\pi kT}\right)^{\frac{1}{2}} \left(\frac{m}{2\pi kT}\right)^{\frac{1}{2}} \exp\left[-\frac{m}{2kT}C^2\right] C d\theta \\
\chi(C) &= \left(\frac{m}{2\pi kT}\right) C \exp\left[-\frac{m}{2kT}C^2\right] \int_0^{2\pi} d\theta \\
\chi(C) &= \left(\frac{m}{2\pi kT}\right) 2\pi C \exp\left[-\frac{m}{2kT}C^2\right] \\
\chi(C) &= \left(\frac{m}{kT}\right) C \exp\left[-\frac{m}{2kT}C^2\right]
\end{aligned}$$

Step 2

A computer program was written in MATLAB to simulate molecular dynamics in 2D. The algorithm was implemented with the help of *Molecular Dynamics Simulation* by Haile, J. M.. Please see the appendix for the complete copy of the program.

The program is an event-driven simulation of 2D gas particles inside a 2D box. Time steps are not uniform in this simulation. Each frame is resolved by finding the next collision event and the time stamp, then solving for all the particle parameters (positions, velocities) and potential collision times with every other particle.

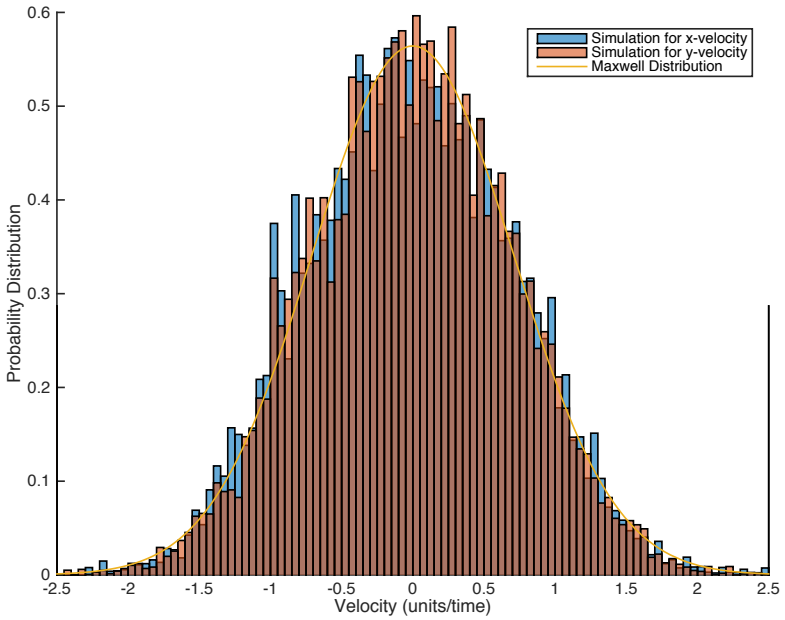


Figure 1: Comparison between simulated and theoretical velocity components $\dot{x}$ and $\dot{y}$ distribution for 400 2D gas particles over 10000 collision events

Step 3

The MATLAB program was implemented to be able to output real-time animation of the simulation. In order to achieve fast simulation time as well as quick graphics output, the program is selective in updating only the necessary information for each frame. Since each frame is driven by a collision event, the program only updates the particle parameters and potential collision times for those involved in the event instead of all particles. At the same time, instead of calling a `plot()` function for every frame, the program uses `set()` for the `xdata` and `ydata` in the existing plot. In practice, this combined effort increases simulation performance by as much as 10000%.

The difficulty in implementing the more efficient algorithm may have caused some bugs, however. As defined by the project definition, a 400 particles simulation ran over 10000 collision events was completed. Some faulty velocity and speed distributions were found for this simulation: the particles were disproportionally likely to have the speed of 1. Debugging efforts revealed that the program may be registering additional, illegitimate collision events (a speculation). An analysis of the kinetic energy showed no obvious errors as the energy stays constant throughout the simulation. Checks were performed to make sure that no particles ever step outside of the walls. The resulting distributions , seen in Figure 1 and 2, suggest that the program is *mostly* correct since the simulation matches closely with theory except for the over-representation of unity speed. Nevertheless, the bug could not be identified.

Interestingly, if the number of particles is reduced to 100 (or less), the resulting distributions, shown in Figure 3 and 4, are much betters matches to the theory. The exact reason for this could not be determined.

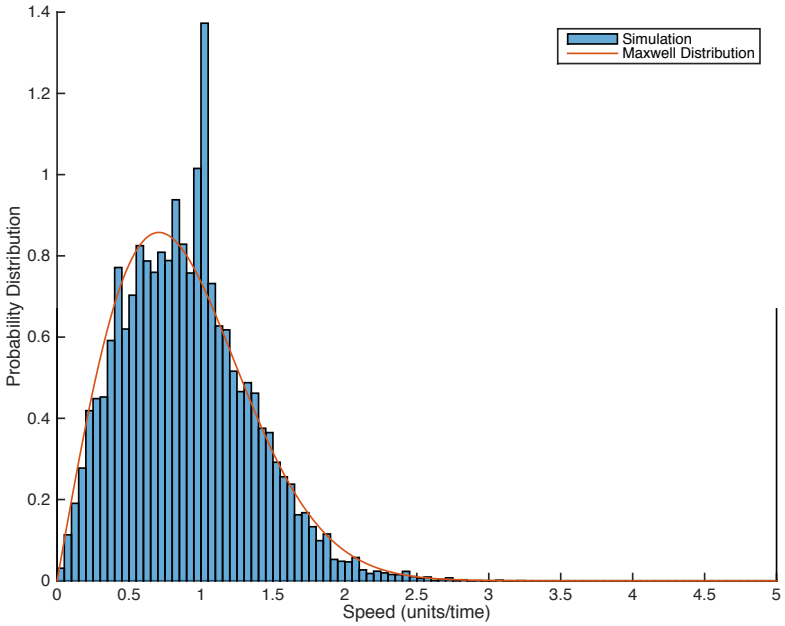


Figure 2: Comparison between simulated and theoretical speed c distribution for 400 2D gas particles over 10000 collision events

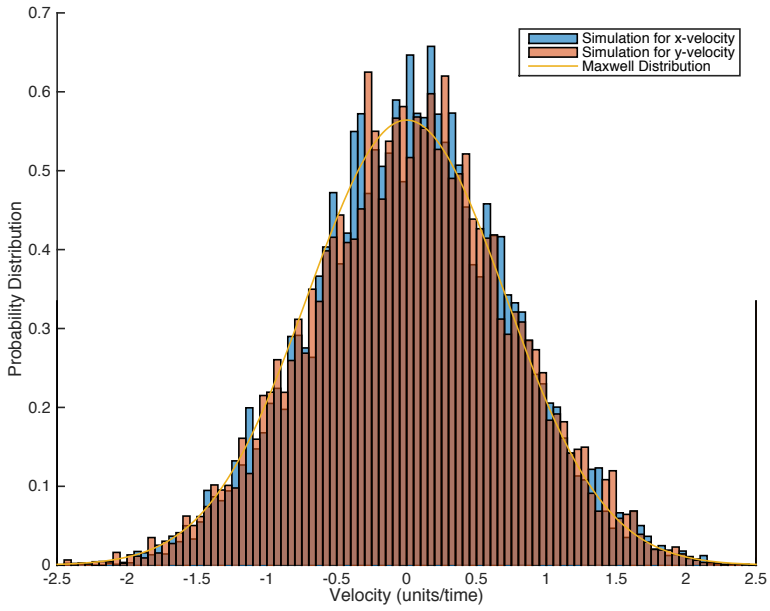


Figure 3: Comparison between simulated and theoretical velocity components $\dot{x}$ and $\dot{y}$ distribution for 100 2D gas particles over 10000 collision events

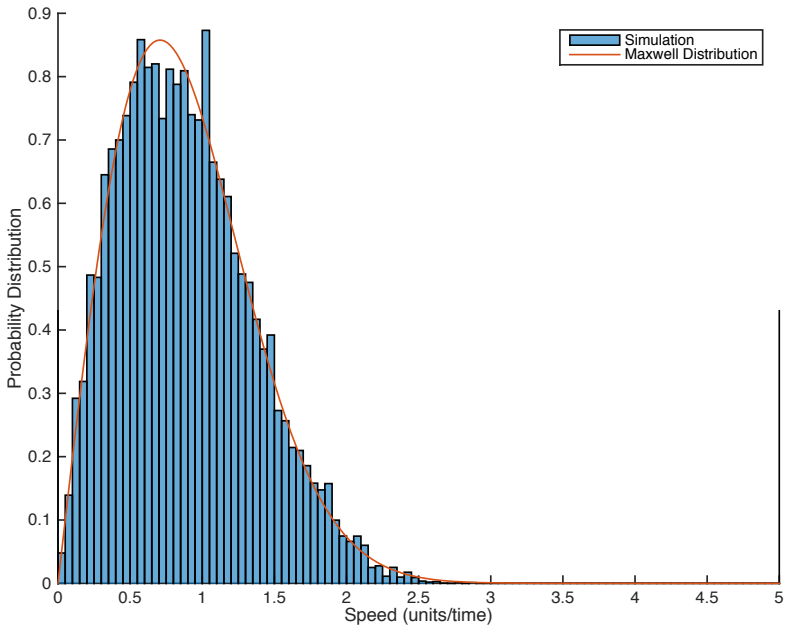


Figure 4: Comparison between simulated and theoretical speed c distribution for 100 2D gas particles over 10000 collision events

Appendix I: MATLAB Code for 2D Gas Simulation

Contents


```
% Project: 2D gas simulation
% John Lian 260408431
% Feb 2015

clear all

% Input parameters
nMolecules = 400;
dimensionSize = sqrt(nMolecules);
nCollisions = 1000;

% Particle-particle collision array
% Same size as number of pairs
parHitTable = zeros(0.5*nMolecules*(nMolecules-1),3);

% Particle-wall collison array
wallHitTable = zeros(nMolecules,3);

% Particle diameter
d = 0.1;

% Particle mass
mass = 1;

% The wall
left = 0;
right = dimensionSize+1;
bottom = 0;
top = dimensionSize+1;
wall = [left right bottom top];
deltaWallHitTime = zeros(length(wall),1);

% Initialize arrays consisting of the positions for each molecule
pos = zeros(nMolecules,2);
vel = zeros(nMolecules,2);
```

```
xy = zeros(nMolecules,2,nCollisions);
uv = zeros(nMolecules,2,nCollisions);
c = zeros(nMolecules,1,nCollisions);
E = zeros(nCollisions,1);
T = zeros(nCollisions,1);
```

Initialization

```
disp('Initializing...');

t = 0;
count = 0;
% Initialize array for the first frame/timestep
for i = 1:dimensionSize
for j = 1:dimensionSize
count = count+1;
pos(count,1) = i;
pos(count,2) = j;
theta = 2*pi*rand(1,1);
vel(count,1) = cos(theta);
vel(count,2) = sin(theta);
end
end

% Test for recording the positions and velocities
xy(:,:,1) = pos(:,:);
uv(:,:,1) = vel(:,:);


count = 0;
% Initialize the particle collision table
for n = 1:nMolecules-1
for m = n+1:nMolecules
count = count+1;

% Calculate the vector relative position and velocity
dr = pos(n,:)-pos(m,:);
dv = vel(n,:)-vel(m,:);

% Calculate relavent constants based on input data
% See Haile Section 3.2.1
situationA = dot(dv,dr);
situationB = situationA^2 - norm(dv)^2*(norm(dr)^2-d^2);
```

```
parHitTable(count,1) = n;
parHitTable(count,2) = m;

% Checking if the two situations are satisfied for collision
if (situationA < 0 && situationB >= 0)
% Solve for time of closest approach.
parHitTable(count,3) = min(t+(-situationA+sqrt(situationB))/norm(dv)^2,...
   t+(-situationA-sqrt(situationB))/norm(dv)^2);
end
end
end
parHitTable(parHitTable==0) = NaN;

count = 0;
% Initialize wall collision table
for n = 1:nMolecules
for m = 1:length(wall)
count = count+1;

wallHitTable(count,1) = n;
wallHitTable(count,2) = m;

% Use appropriate velocity components for the appropriate walls
if m == 1 || m == 2
wallHitTable(count,3) = t+(wall(m)-pos(n,1))/vel(n,1)-(d/2)/abs(vel(n,1));
else
wallHitTable(count,3) = t+(wall(m)-pos(n,2))/vel(n,2)-(d/2)/abs(vel(n,2));
end
end
end
wallHitTable(wallHitTable<=0) = NaN;
```

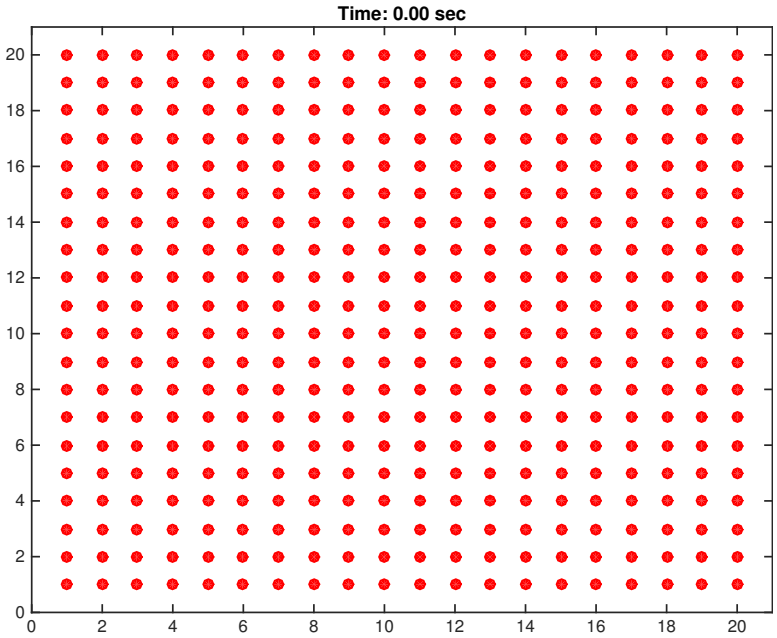
```
Initializing...
```

Plotting the initial frame

```
% set up first frame
figure('Color', 'white');

h = plot(xy(:,1,1),xy(:,2,1),...
'o','Color','red','MarkerFaceColor','red','MarkerSize',6);
xlim([0 dimensionSize+1]);
ylim([0 dimensionSize+1]);
```

```
ht = title(sprintf('Time: %0.2f sec', T(1)));
drawnow update
```



Math for collision and things

```
for frame = 2:nCollisions

% Determine which collision is going to happen next by finding the minimum
% of collision time
[nextParHitTime,pairFlag] = min(parHitTable(:,3));
[nextWallHitTime,wallFlag] = min(wallHitTable(:,3));


if (nextParHitTime < nextWallHitTime)
% 2 particles will collide before a wall collision occurs
nInvolved = 2;

dt = nextParHitTime-t;
parA = parHitTable(pairFlag,1);
parB = parHitTable(pairFlag,2);

% Advance time to collsion time
t = t+dt;

% Update all positions
pos = pos+vel*dt;
```

```
% Find post collision velocities and update
rHat = (pos(parB,:)-pos(parA,:))/norm(pos(parB,:)-pos(parA,:));
va = vel(parA,:)-dot((vel(parA,:)-vel(parB,:)),rHat)*rHat;
vb = vel(parB,:)+dot((vel(parA,:)-vel(parB,:)),rHat)*rHat;

% Update velocities
vel(parA,:) = va;
vel(parB,:) = vb;


% Clear collision time for this event
parHitTable(pairFlag,3) = NaN;

else
% A wall collision will happen first
nInvolved = 1;

dt = nextWallHitTime-t;
parA = wallHitTable(wallFlag,1);
wallHit = wallHitTable(wallFlag,2);

% Advance time to collsion time
t = t+dt;

% Update all positions
pos = pos+vel*dt;


% Update the velocity of the colliding particle
if wallHit == 1 || wallHit == 2
vel(parA,1) = -vel(parA,1);
else
vel(parA,2) = -vel(parA,2);
end

% Clear collision time for this event
wallHitTable(wallFlag,3) = NaN;

end


if isnan(pos(pos<0)) == 0
error('Negative position detected!!!');
end
```

```
% Update particle collision table for the relevant particles

for count = 1:nInvolved
if count == 1
% Find all the pairs for particles involved
[p2p2Check,~] = find(parHitTable == parA);
p2w2Check = parA*4-3:parA*4;
else
[p2p2Check,~] = find(parHitTable == parB);
p2w2Check = parB*4-3:parB*4;
end
for i = 1:nMolecules-1

n = parHitTable(p2p2Check(i),1);
m = parHitTable(p2p2Check(i),2);

% Calculate the vector relative position and velocity
dr = pos(n,:)-pos(m,:);
dv = vel(n,:)-vel(m,:);

% Calculate relavent constants based on input data
% See Haile Section 3.2.1
situationA = dot(dv,dr);
situationB = situationA^2 - norm(dv)^2*(norm(dr)^2-d^2);

% Checking if the two situations are satisfied for collision
if (situationA < 0 && situationB >= 0)
% Solve for time of closest approach.
tc = min(t+(-situationA+sqrt(situationB))/norm(dv)^2,...
 t+(-situationA-sqrt(situationB))/norm(dv)^2);
parHitTable(p2p2Check(i),3) = tc;
end
end


% Update wall collision table for the relevant particle

for i = 1:length(wall)

n = wallHitTable(p2w2Check(i),1);
m = wallHitTable(p2w2Check(i),2);

% Use appropriate velocity components for the appropriate walls
if m == 1 || m == 2
deltaWallHitTime(m) = (wall(m)-pos(n,1))/vel(n,1)-(d/2)/abs(vel(n,1));
```

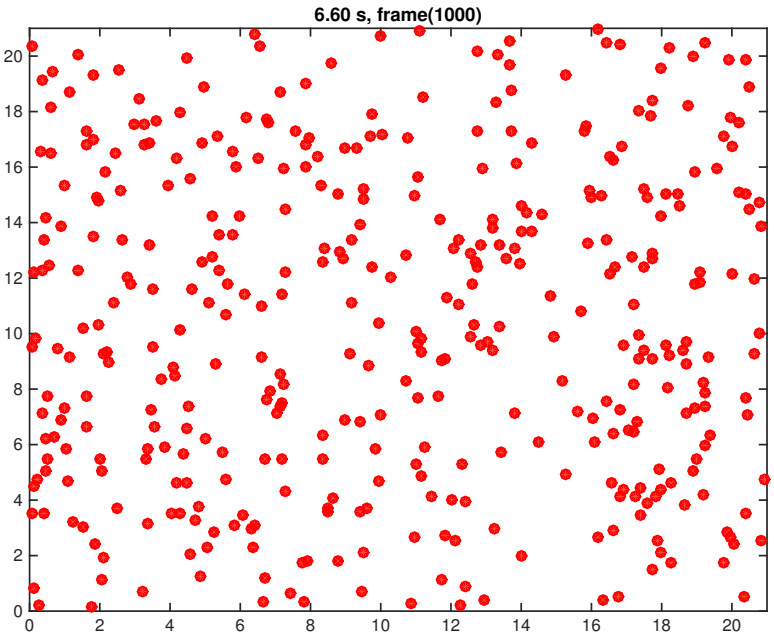
```
else
deltaWallHitTime(m) = (wall(m)-pos(n,2))/vel(n,2)-(d/2)/abs(vel(n,2));
end
if deltaWallHitTime(m) <= 1e-10
deltaWallHitTime(m) = NaN;
end
wallHitTable(p2w2Check(i),3) = t + deltaWallHitTime(m);
end


end

% Record the positions and velocities
xy(:,:,frame) = pos(:,:);
uv(:,:,frame) = vel(:,:);
c(:,1,frame) = sqrt(sum(vel.^2,2));
E(frame) = (1/2)*mass*sum(sum(vel.^2,2));
T(frame) = t;

set(h, 'XData', xy(:,1,frame));
set(h, 'YData', xy(:,2,frame));
set(ht, 'String', sprintf('%0.2f s, frame(%d)', T(frame),frame));
drawnow update

end
```



Probability distribution

```
figure(2)
hold on
edges = [0 0:0.05:5 5];
hc = histogram(c,edges,'Normalization','pdf');
% Boltzmann
k = 1;
% Temperature
T = 1/2;
f1 = @(y) (mass/(k*T))*y*exp(-mass*(y^2)/(2*k*T));
fplot(f1,[0 5])
xlabel('Speed (units/time)')
ylabel('Probability Distribution')
legend('Simulation', 'Maxwell Distribution')
hold off

figure(3)

edges = [-2.5 -2.5:0.05:2.5 2.5];
hold on
hu = histogram(uv(:,1,:),edges,'Normalization','pdf');
hv = histogram(uv(:,2,:),edges,'Normalization','pdf');
f2 = @(y) sqrt(mass/(2*pi*k*T))*exp(-mass*(y^2)/(2*k*T));
fplot(f2,[-2.5 2.5])
xlabel('Velocity (units/time)')
ylabel('Probability Distribution')
legend('Simulation for x-velocity', 'Simulation for y-velocity'...
,'Maxwell Distribution')
hold off
```

Note: for figures here please see Step 3.