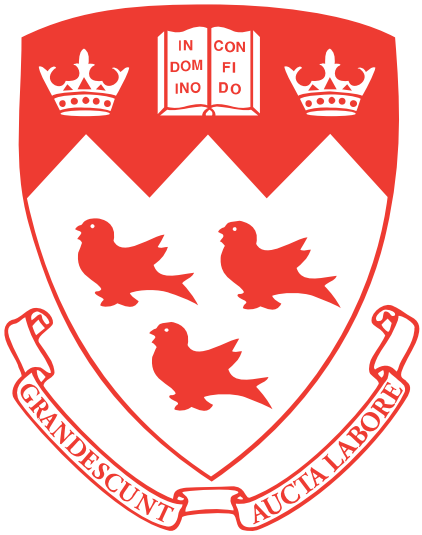
Semester Project Part 1 - Progress Report



McGill University

Giuliano Francescangeli
Department of Mechanical Engineering

Submitted to McGill University in partial fulfilment of the requirements of the Undergraduate Honours Program.

1 Step 1

The differential area element in 2D is defined using polar coordinates as,

$$dA = dc_x c_y = CdCd\theta \tag{1}$$

Knowing the Maxwell distribution function for 1D velocities is defined as,

$$f(c_i) = (\frac{m}{2\pi kT})^{\frac{1}{2}} e^{-(\frac{m}{2kT}c_i^2)} \tag{2}$$

Integrating Eq. (2) over a 2D velocity space results in the desired probability density function (PDF). The PDF describes the probability distribution of speeds which determines the probability of finding a differential number of particles/molecules in a resultant velocity range from C to $C + dC$ along the differential area element defined in Eq. (1).

$$\chi(C) = \int_C^{C+dC} \int_0^{2\pi} (\frac{m}{2\pi kT})^{\frac{1}{2}} (\frac{m}{2\pi kT})^{\frac{1}{2}} e^{-(\frac{m}{2kT}C^2)} Cd\theta dC$$

$$\chi(C) = (\frac{m}{2\pi kT}) \int_C^{C+dC} e^{-(\frac{m}{2kT}C^2)} CdC \int_0^{2\pi} d\theta$$

$$\chi(C) = 2\pi(\frac{m}{2\pi kT}) \int_C^{C+dC} e^{-(\frac{m}{2kT}C^2)} CdC$$

Eq. (3) defines the cumulative distribution function (CDF).

$$\chi(C) = (\frac{m}{kT}) \int_C^{C+dC} e^{-(\frac{m}{2kT}C^2)} CdC \tag{3}$$

The integral sign over C to $C + dC$ can be dropped to obtain the PDF, as defined in Eq. (4).

$$\chi(C) = C(\frac{m}{kT}) e^{-(\frac{m}{2kT}C^2)} \tag{4}$$

2 Step 2

Figure 1 depicts the resulting Molecular Dynamic (MD) simulation written in Matlab (using 20000 collisions), the algorithm of which was implemented using [1] as reference, and can be

found in the Appendix. This algorithm uses non-uniform time-steps which are determined based on the next nearest collision, either between molecules or between a molecules and one of the (left, right, bottom, top) walls; the positions and velocities of all molecules involved in the current collision are then updated to produce the current scene/frame. The velocities of each molecule in the current collision induced frame is stored to later sample and produce a distribution (depicted using a histogram in step 3) and compared against the theoretical 2D Maxwell distribution of velocities. Note that any molecules that seem to be escaping the box are in fact about to experience a collision with the corresponding wall, all molecules remain within the box throughout the simulation.

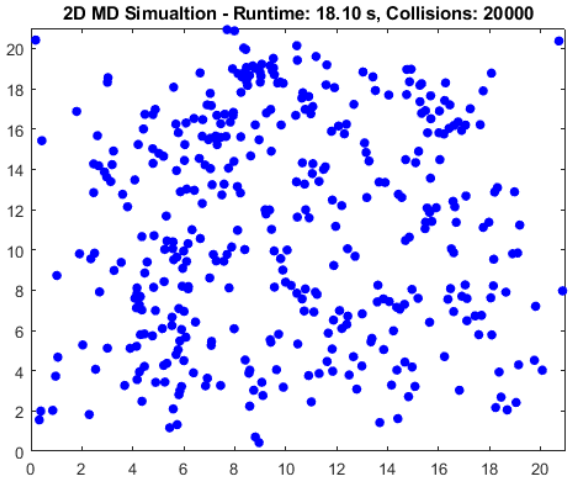


Figure 1: 2D MD simulation of 400 gas molecules for 20000 collisions

3 Step 3

Figures 2 and 3 depict the 2D Maxwell distribution ran for 20000 and 4000 collisions respectively. The results show that as the number of collisions are increased the sampling becomes more accurate and a better fit to the theoretical Maxwell distribution. Note that after around 10000 collisions for 400 molecules the fit of the experimental distribution doesn't improve significantly, leading to the belief that there is a minimum number of collisions per molecule required to achieve the Maxwellian profile. The only visible error is a spike at a Molecular speed of 1, however, as the collision count increased the disproportionate number of molecules with a speed of 1 decreases, signifying that unitary velocity is just a temporary bulk state. The total kinetic energy remained constant at 200 for the entirety of the simulation (see Figure 6), as expected; so, the bulk state (speed equal to 1) is not an error within the system of gas molecules.

Figure 4 and 5 depict the 1D Maxwell distributions for the x-component and y-component velocities respectively. Again a good agreement between the experimental and theoretical results is observed. Note that the 1D simulated distribution is more sensitive to a varying number of collisions, this makes sense seeing as the 1D results do not use an average (resultant) velocity as is the case with the 2D results.

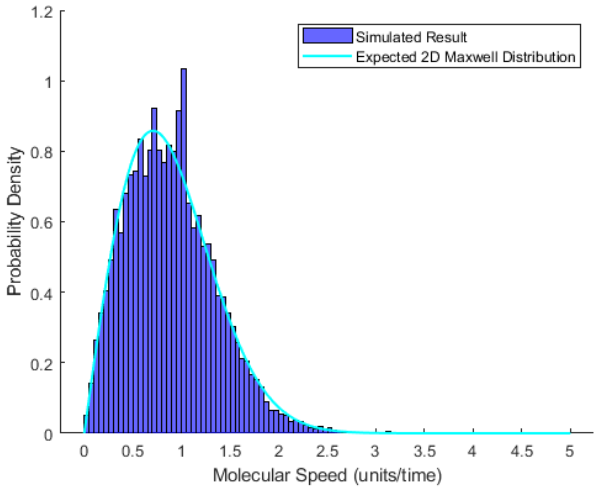


Figure 2: Comparison between the experimental and theoretical 2D speed distribution of 400 gas molecules for 20000 collisions

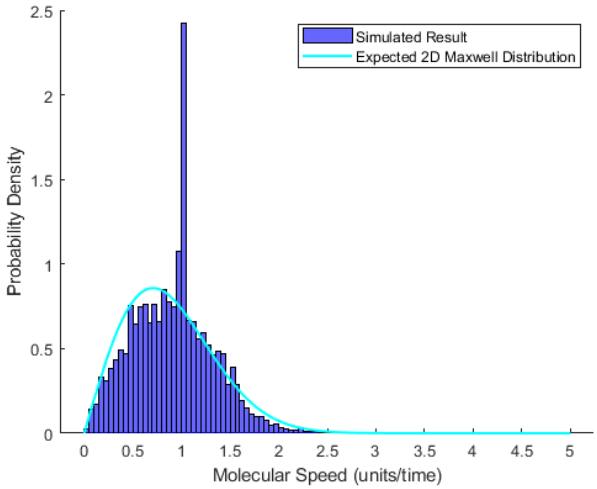


Figure 3: Comparison between the experimental and theoretical 2D speed distribution of 400 gas molecules for 4000 collisions

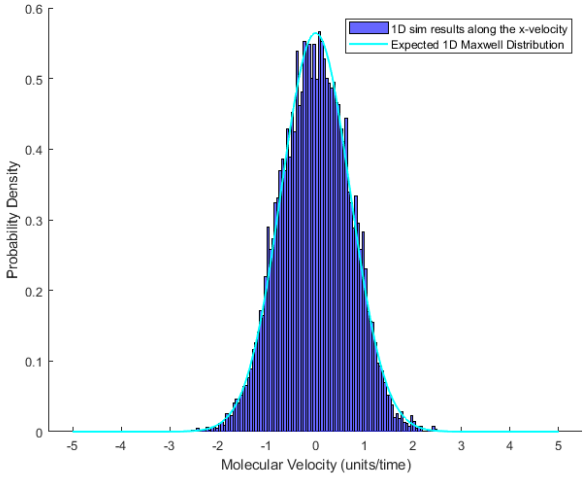


Figure 4: Comparison between the experimental and theoretical 1D x-velocity distribution of 400 gas molecules for 20000 collisions

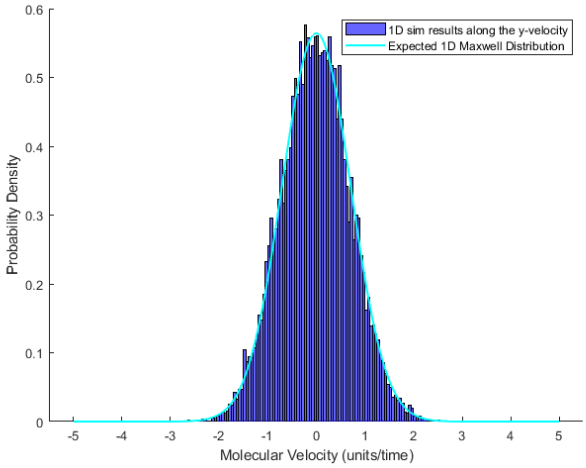


Figure 5: Comparison between the experimental and theoretical 1D y-velocity distribution of 400 gas molecules for 20000 collisions

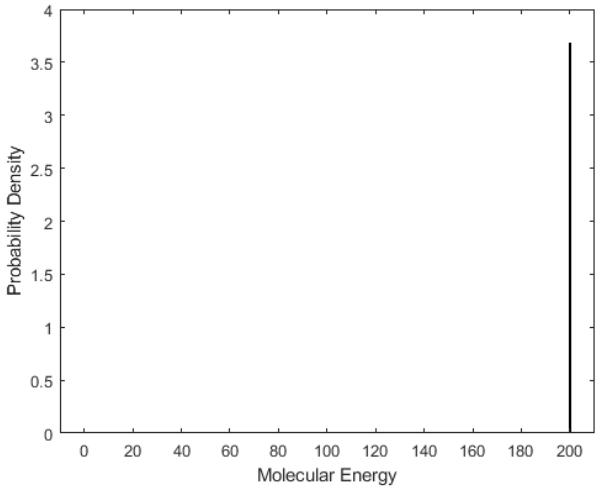


Figure 6: Total energy distribution of 400 gas molecules for 20000 collisions

4 Appendix

Listing 1: 2D MD simulation written in Matlab

```
clc
clear all

% Input parameters
mass = 1;
diameter = 0.1;
k = 1; % Boltzmann constant
T_0 = 1/2; % Temperature
num_mol = 400;
dim = sqrt(num_mol); % Dimension of the box
num_cols = 20000;

left = 0;
right = dim+1;
bottom = 0;
top = dim+1;
wall = [left right bottom top];
time_of_closest_approach_wall = zeros(length(wall),1);

% Arrays for collision Recordings
collisions_between_molecules = zeros(0.5*num_mol*(num_mol-1),3); % Particles are in pairs
collisions_with_wall = zeros(num_mol,3);

% Initialize arrays consisting for attributes of molecule
mol_position = zeros(num_mol,2);
mol_velocity = zeros(num_mol,2);
position_plot = zeros(num_mol,2,num_cols);
velocity_plot = zeros(num_mol,2,num_cols);
C = zeros(num_mol,1,num_cols);
Energy = zeros(num_cols,1);

%Time parameters
t = 0;
Time = zeros(num_cols,1);

count = 0;
% Seed random number generator
for i = 1:dim
      for j = 1:dim

      count = count+1;

            mol_position(count,1) = i;
            mol_position(count,2) = j;

            theta = 2*pi*randi([0 360],1,1)/360;
            mol_velocity(count,1) = cos(theta);
            mol_velocity(count,2) = sin(theta);

      end
```

```
end

position_plot(:,:,1) = mol_position(:,:);
velocity_plot(:,:,1) = mol_velocity(:,:);

% Plotting the initial frame
figure(1);
P = plot(position_plot(:,1,1),position_plot(:,2,1),'o','Color','blue',...
   'MarkerFaceColor','blue','MarkerSize',5);
xlim([0 dim+1]);
ylim([0 dim+1]);
time_frames = title(sprintf('2D MD Simualtion - Runtime: %0.2f sec', Time(1)));
drawnow update

% Algorithm implementaion follows see Haile Section 3.2.1

count = 0;
% Collisions between molecules
for i = 1:(num_mol-1)
      for j = (i+1):num_mol

            count = count+1;

            % Calculating the relative position and velocity
            dr = mol_position(i,:) - mol_position(j,:);
            dv = mol_velocity(i,:) - mol_velocity(j,:);

      dot_product = dot(dv,dr);
            situationB = dot_product^2 - norm(dv)^2*(norm(dr)^2-diameter^2);

      if (dot_product >= 0)
         % There is no collision
         collisions_between_molecules(count,1) = i;
         collisions_between_molecules(count,2) = j;
         collisions_between_molecules(count,3) = NaN;
         continue
      end

      if (situationB < 0)
         % There is no collision
         collisions_between_molecules(count,1) = i;
         collisions_between_molecules(count,2) = j;
         collisions_between_molecules(count,3) = NaN;
         continue
      end

      % Checking if the distance of closest approach is greater than diameter
            if (dot_product < 0 && situationB >= 0)
         collisions_between_molecules(count,1) = i;
         collisions_between_molecules(count,2) = j;
                  % Solving for time of closest approach Eq 3.24 from Hailee.
                  collisions_between_molecules(count,3) = min(t+...
            (-dot_product+sqrt(situationB))/norm(dv)^2, ...
            t+(-dot_product-sqrt(situationB))/norm(dv)^2);
            end
```

```
      end
end


count = 0;
% Collisions with wall
for i = 1:num_mol
      for j = 1:length(wall)

            count = count+1;

            collisions_with_wall(count,1) = i;
            collisions_with_wall(count,2) = j;


            if j == 1 || j == 2 % Left & Right walls
                  collisions_with_wall(count,3) = t + (wall(j) - mol_position(i,1))...
            /mol_velocity(i,1) - (diameter/2)/abs(mol_velocity(i,1));
            else % Bottom & Top walls
                  collisions_with_wall(count,3) = t + (wall(j) - mol_position(i,2))...
            /mol_velocity(i,2) - (diameter/2)/abs(mol_velocity(i,2));
            end
      end
end
% Wall was not hit, fill zeros with NaN
collisions_with_wall(collisions_with_wall<=0) = NaN;

for col_frame = 2:num_cols % Initial collision frame is static, so iterate from second onward

      % Finding the next minimum collision time
      [next_molecules_collision_time, mols_col_index] = min(collisions_between_molecules(:,3));
      [next_wall_collision_time, wall_hit_index] = min(collisions_with_wall(:,3));

   % Case where collision between molecules occurs sooner
   if (next_molecules_collision_time < next_wall_collision_time)
      a = 2; % Flag for collision between molecules happening first
            dt = next_molecules_collision_time - t;
            molecule1 = collisions_between_molecules(mols_col_index,1);
            molecule2 = collisions_between_molecules(mols_col_index,2);
            t = t + dt;

      % Update position and velocity
            mol_position = mol_position + mol_velocity*dt;
            r_rel_norm = (mol_position(molecule2,:) - mol_position(molecule1,:))...
         /norm(mol_position(molecule2,:) - mol_position(molecule1,:));
            v_1 = mol_velocity(molecule1,:)-dot((mol_velocity(molecule1,:)...
         -mol_velocity(molecule2,:)),r_rel_norm)*r_rel_norm; % Eq. 3.17
            v_2 = mol_velocity(molecule2,:)+dot((mol_velocity(molecule1,:)...
         -mol_velocity(molecule2,:)),r_rel_norm)*r_rel_norm; % Eq. 3.18
            mol_velocity(molecule1,:) = v_1;
            mol_velocity(molecule2,:) = v_2;

      % Once the collision is handled clear the time entry to move forward
            collisions_between_molecules(mols_col_index,3) = NaN;
   else
```

```
            a = 1; % Flag for collision between a molecule and a wall hapening first
            dt = next_wall_collision_time - t;
            molecule1 = collisions_with_wall(wall_hit_index,1);
            wall_col = collisions_with_wall(wall_hit_index,2);
            t = t + dt;

      % Update position and velocity
            mol_position = mol_position + mol_velocity*dt;

            if wall_col == 1 || wall_col == 2 % Left & Right walls
                  mol_velocity(molecule1,1) = -mol_velocity(molecule1,1);
      else % Bottom & Top walls
                  mol_velocity(molecule1,2) = -mol_velocity(molecule1,2);
            end

            collisions_with_wall(wall_hit_index,3) = NaN; %Collision was handled
   end

   % Loop through collision type flag to update particle collision list
   % or wall collision list
      for count = 1:a

      if count == 1
                  [mol_col_flag, ~] = find(collisions_between_molecules == molecule1);
                  [wall_col_flag, ~] = find(collisions_with_wall == molecule1);
      else
                  [mol_col_flag,~] = find(collisions_between_molecules == molecule2);
                  [wall_col_flag, ~] = find(collisions_with_wall == molecule2);
      end

      for i = 1:num_mol-1

                  n = collisions_between_molecules(mol_col_flag(i),1);
                  m = collisions_between_molecules(mol_col_flag(i),2);

                  % Redo calculations to compute new entries for the list of
         % collision times Step 11 of Haile algorithm
                  dr = mol_position(n,:) - mol_position(m,:);
                  dv = mol_velocity(n,:) - mol_velocity(m,:);
                  dot_product = dot(dv,dr);
                  situationB = dot_product^2 - norm(dv)^2*(norm(dr)^2-diameter^2);

         if (dot_product < 0 && situationB >= 0)
                        % Resolving for the time of closest approach
                        temp_t = min(t+(-dot_product+sqrt(situationB))/norm(dv)^2,...
                                    t+(-dot_product-sqrt(situationB))/norm(dv)^2);
                        collisions_between_molecules(mol_col_flag(i),3) = temp_t;
         end
      end

            for i = 1:length(wall)

                  n = collisions_with_wall(wall_col_flag(i),1);
                  m = collisions_with_wall(wall_col_flag(i),2);
```

```
                    if m == 1 || m == 2 % Left & Right walls
                          time_of_closest_approach_wall(m) = (wall(m)-mol_position(n,1))...
                /mol_velocity(n,1)-(diameter/2)/abs(mol_velocity(n,1));
            else % Bottom & Top walls
                          time_of_closest_approach_wall(m) = (wall(m)-mol_position(n,2))...
                /mol_velocity(n,2)-(diameter/2)/abs(mol_velocity(n,2));
            end

                    if time_of_closest_approach_wall(m) <= 1e-12
                          time_of_closest_approach_wall(m) = NaN;
                    end
                    collisions_with_wall(wall_col_flag(i),3) = t + ...
               time_of_closest_approach_wall(m);
               end


        end

        position_plot(:,:,col_frame) = mol_position(:,:);
        velocity_plot(:,:,col_frame) = mol_velocity(:,:);
        C(:,1,col_frame) = sqrt(sum(mol_velocity.^2,2));
        Time(col_frame) = t;

    % Update collision frame
        set(P, 'XData', position_plot(:,1,col_frame));
        set(P, 'YData', position_plot(:,2,col_frame));
        set(time_frames, 'String', ...
        sprintf('2D MD Simualtion - Runtime: %0.2f s, Collisions: %d', Time(col_frame), col_frame));
        drawnow update

        % Store current total kinetic energy
    Energy(col_frame) = (1/2)*mass*sum(sum(mol_velocity.^2,2));
end

% Plotting the 2D Maxwellian distribution
figure(2)
hold on
edges = [0 0:0.05:5 5];
exp_distribution = histogram(C,edges,'Normalization','pdf');
exp_distribution.FaceColor = "b";
X_C = @(y) (mass/(k*T_0))*y*exp(-mass*(y^2)/(2*k*T_0));
fplot(X_C, [0 5],"-c",'Linewidth',1.5)
xlabel('Molecular Speed (units/time)')
ylabel('Probability Density')
legend('Simulated Result', 'Expected 2D Maxwell Distribution')
hold off

% Plotting the energy distribution
figure(3)
exp_distribution = histogram(Energy,'Normalization','pdf');
exp_distribution.FaceColor = "b";
xlabel('Molecular Energy')
ylabel('Probability Density')

% Plotting the 1D Maxwellian distributions
```

```
271
272 figure(4)
273 hold on
274 edges = [-5 -5:0.05:5 5];
275 exp_distribution_x = histogram(velocity_plot(:,1,:),edges,'Normalization','pdf');
276 exp_distribution_x.FaceColor = "b";
277 f_C = @(y) sqrt(mass/(2*pi*k*T_0))*exp(-mass*(y^2)/(2*k*T_0));
278 fplot(f_C, [-5 5],"-c",'Linewidth',1.5)
279 xlabel('Molecular Velocity (units/time)')
280 ylabel('Probability Density')
281 legend('1D sim results along the x-velocity', 'Expected 1D Maxwell Distribution')
282 hold off
283
284 figure(5)
285 hold on
286 edges = [-5 -5:0.05:5 5];
287 exp_distribution_y = histogram(velocity_plot(:,2,:),edges,'Normalization','pdf');
288 exp_distribution_y.FaceColor = "b";
289 f_C = @(y) sqrt(mass/(2*pi*k*T_0))*exp(-mass*(y^2)/(2*k*T_0));
290 fplot(f_C, [-5 5],"-c",'Linewidth',1.5)
291 xlabel('Molecular Velocity (units/time)')
292 ylabel('Probability Density')
293 legend('1D sim results along the y-velocity','Expected 1D Maxwell Distribution')
294 hold off
```

References

[1] J. M. Haile, *Molecular dynamics simulation: Elementary methods*. Wiley, 1997.