PHY407 Final Project: Monte Carlo Simulation

Mark Buchanan: 1003520745

December 4th, 2019

1 Physics Background

1.1 Troposphere

The troposphere is the lowest level of the atmosphere. It ranges in height based on latitude. We will take it to range in height from sea level to 19 kilometres above sea level which corresponds to the height of the troposphere at the equator.

The temperature of the air decreases as we increase in elevation above sea level. We will approximate the changing temperature as a linear relation with height and decrease the temperature step-wise for every 1000 metres we go up. The temperature at sea level is taken to be $17°C$, and the temperature at the top of the troposphere (height of 19 kilometres) is taken to be $-51°C$. A temperature-elevation profile of the atmosphere including the troposphere can be seen in Figure 1.

The atmospheric composition includes various gasses and liquids. For the purpose of this lab, we will assume a dry atmosphere (gasses only). This is a fair approximation for desert regions. We will take the composition of the troposphere to be 79% Nitrogen (N_2) and 21% Oxygen (O_2).

Treating the troposphere as an ideal gas makes for simpler simulation and is still able to provide meaningful results. This means neglecting quantum effects of the particles and treating collisions as elastic.

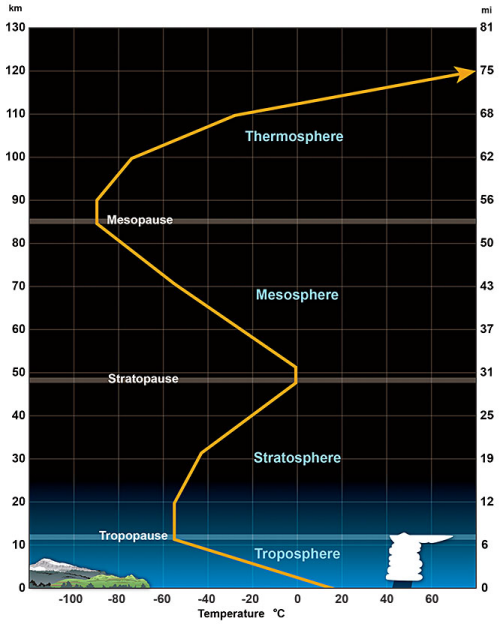


Figure 1: Atmospheric profile. Taken from NOAA source.

Information sourced from NOAA (1), NOAA (2) and UC Irvine Chemistry.

1.2 Boltzmann Distribution

The Boltzmann distribution shows up many places in statistical mechanics. Of particular use is the so called 'canonical ensemble' which involves a closed system in contact with a heat bath. The distribution is a classical approximation to systems that are in equilibrium with a given temperature T. It states that the number of particles N_i with a particular energy E_i follows

$$N_i = N \cdot exp\Big(\frac{E_i}{k_B T}\Big)$$

where N is the total number of particles in the system and k_B is the Boltzmann constant.

Information sourced from University of Kiel Materials Science.

1.3 Maxwell-Boltzmann Distribution

The Maxwell-Boltzmann distribution is a distribution of the speeds of a gas for a given temperature T. We assume that the ideal gas we're considering has independent speed components (ie. v_x independent of v_y and v_z and vice versa). The distribution for a total speed $v = \sqrt{v_x^2 + v_y^2 + v_z^2}$ is given by

$$f(v) = 4\pi v^2 \Big(\frac{m}{2\pi k_B T}\Big)^{\frac{3}{2}} exp\Big(\frac{-mv^2}{2k_B T}\Big)$$

where m is the mass of the molecule, k_B is the Boltzmann constant, and T is the temperature of the system.

The shape of the distribution changes with temperature. As temperature increases, the distribution shifts towards higher speeds and becomes more broad. This can be seen in Figure 2. The shape of the distribution also changes with molecule mass. Heavier molecules move slower than lighter ones, which consequently make heavier particles speed distribution more compact.

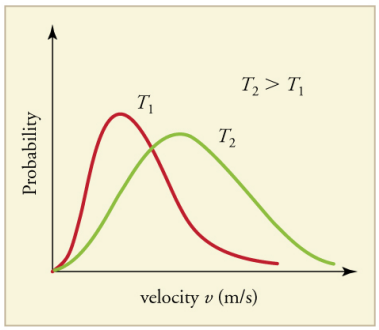


Figure 2: Maxwell-Boltzmann distributions. Taken from LibreTexts source.

Information sourced from Chemistry LibreTexts.

2 Computational Background

2.1 Metropolis Method for Monte Carlo

Monte Carlo simulation is most useful for problems of statistic mechanics, like simulating the molecules of the atmosphere. The Monte Carlo method works well in this domain since the foundation of statistical mechanics is random processes. The metropolis method involves generating random numbers and using them to move systems from one state to another. The probability of accepting a new state with energy E_j from an initial state with energy E_i is given by

$$P_a = \begin{cases} 1 & \text{if } E_j \leq E_j, \\ e^{-\beta(E_j - E_i)} & \text{if } E_j > E_i \end{cases} \tag{1}$$

where $\beta = \frac{1}{k_B T}$ with k_B being the Boltzmann constant and T is the temperature.

Importantly, Markov chain simulations (which the metropolis method is part of) converge to Boltzmann distributions as long as the appropriate criteria are met (steps where don't change state are still counted, all states are accessible from all states). This is proven in Appendix D of the course textbook Computational Physics.

Information sourced from chapter 10.3 of the course textbook Computational Physics by Mark Newman.

2.2 Random Number Generation

The key to an accurate Monte Carlo simulation is a good random (or more precisely, pseudo-random) number generator (RNG). There are many ways to generate random numbers. For our purposes, we are interested in random numbers generated uniformly between two endpoints.

Mersenne Twister: This is the method used by `numpy`. The method utilizes Mersenne primes, which are prime numbers of the form $2^p - 1$ where p is a prime number. The most widely used Mersenne twister is MT19937 which utilizes the Mersenne prime $2^{19937} - 1$. The period of a Mersenne twister is the associated Mersenne prime. Without getting to in-depth, the way it works is as follows. The algorithm uses previous values of the generator which are represented by many-dimensional vectors (typically 624) that contain binary values and takes a specified number of one of the previous value's upper bits and concatenates it with a

related number of another previous value's lower bits. The result of this is then multiplied by a matrix which is called a twister which shifts the bits of the resultant vector based on the value of the least significant bit. This result is then added to another previous value in the sequence and we have generated a new random number.

Benefits of this algorithm is that it passes many of important statistical tests for randomness. It is of sufficient quality and efficiency to use for scientific purposes (non-cryptography). It also has a very long period and can be used with low memory requirements. Flaws are limited, but one flaw is that 'similar' seeds will generate similar sequences of random numbers before eventually diverging. Examples of 'similar' seeds are seed values that both have multiple trailing zeros.

Information is sourced from Matsumoto & Nishimura (1) and Matsumoto & Nishimura (2).

Linear Congruential RNG: A linear congruential random number generator is one way to generate numbers from a uniform distribution. It is one of the most easily implemented and famous RNGs. The generator is defined by

$$x_{i+1} = (ax_i + c)\%m$$

for given constants a, c, and m. We can make this a uniform distribution between $0 \leq x_i < 1$ by dividing the equation by m. It is important to choose 'good' values for the constants in the generator, otherwise the possible values it will generate will be diminished and less random. The values given in the textbook have been tested against statistical tests to judge randomness and performed better than other values.

Flaws of this method include that the maximal period is bounded above by m and also that consecutive numbers are correlated to each other. Additionally, the outputs of the generator itself can be contained in a relatively small number of hyperplanes, meaning that much of the possible outputs are artificially restricted. Generating random numbers in an n dimensional space will yield outputs in at most $\sqrt[n]{n!m}$ hyperplanes, where m is the constant we are using in the modulo.

Information sourced from chapter 10.1 of the course textbook Computational Physics by Mark Newman, Indiana University CS, George Marsaglia, and Art Owen (Chapter 3).

Inverse Congruential RNG: The inverse congruential RNG has a similar formulation as the linear congruential RNG. The generation is defined by

$$x_{i+1} = \begin{cases} (ax_i^{-1} + c)\%m & \text{if } x_i \neq 0, \\ c & \text{if } x_i = 0 \end{cases} \tag{2}$$

where x_i^{-1} is defined as $x_i x_i^{-1} \equiv 1\%m$ with m being a prime number. In general, this inverse is hard to find for large values of m. If m is small enough though, we can use Fermat's little theorem which states that $x^{m-1} = 1\%m$ for some integer x. From here, we get that the inverse of x mod m is $x^{-1} = x^{m-2}$. The set of parameters that maximizes period are those where the polynomial $x^2 - cx - a \in F_m$ is primitive (a polynomial ring in F_m). We can again get a uniform distribution by dividing by m. The maximal period is also bounded above by choice of m.

The biggest benefits of this method over the linear congruential RNG is that it does not have the same hyperplane problem, and that its correlations between values weaker. The biggest flaw of this RNG is computational time. The fastest known implementation is still eight times slower than the linear congruential RNG.

Information sourced from chapter 10.1 of the course textbook Computational Physics by Mark Newman, Indiana University CS, Anne Gille-Genest (section 4.2), Art Owen (Chapter 3), and Glyn Holton.

3 Questions

3.1 Boltzmann & Maxwell-Boltzmann Distributions

(a) Using the `numpy.random` package, create a program that simulates the troposphere using the metropolis method for Monte Carlo. Use the conditions described in the physics background for the troposphere. Take the energy of a molecule to be given by

$$E = mgh$$

where m is the mass of the molecule (not the atom), g is the gravitational constant acceleration, and h is the height above sea level. Simulate $N = 5000$ total molecules with the appropriate proportions of gasses. Use an appropriate step size and number of simulation iterations combination.

Make a histogram of the final heights of the nitrogen and oxygen molecules after the metropolis method has stopped significantly changing the system. Also make a histogram of the probability density function for the final heights of the nitrogen and oxygen.

How is the energy of the system increasing? Do the histograms make sense when considering pressure distribution of the atmosphere? Comment on the error and any difference in shape from the Boltzmann distribution.

(b) Modify the program to also calculate the speed of each of the components (v_x, v_y, and v_z) of the molecules and calculate the acceptance probability using the energy

$$K = \frac{1}{2}mv^2$$

where m is the mass of the respective molecules and v is the total speed. Use an appropriate step size and number of simulation iterations combination.

Make the same histograms as in part (a), except with the final total speed of each particle instead of height.

Comment on the error and any difference in shape from the Maxwell-Boltzmann distribution.

3.2 Random Number Generators

Modify the program from the previous question to perform the same simulations with the various implementations of RNGs discussed in the computational background. Discuss any significant changes, and possible reasons for which, compared to the Mersenne twister RNG used in the previous question.

4 Answers

4.1 Boltzmann & Maxwell-Boltzmann Distributions

$5,000$ molecules composed of 79% N_2 and 21% O_2 were simulated for $25,000$ iterations of the metropolis algorithm. The molecules were all initialized as travelling slowly (each component being less than 50 metres per second in magnitude) in the lower troposphere (first 250 metres). At each step, the height and the three components of the molecule's speed were randomly moved up or down a maximal of their respective step sizes. These new values were either accepted or rejected based on the acceptance characteristics described in the metropolis section of the computational method. The total potential and kinetic energies of all the molecules were calculated after each iteration of the metropolis algorithm.

Many different combinations of iteration length and step sizes were tried. Using a large number of iterations and small step sizes produces the best results. However, this also takes the most time to simulate. If testing the code, decrease the number of particles `N` and the number of simulations `simulations` in the code. I found that step sizes of 250 for height and 50 for speed was the best balance between time and accuracy.

The histograms of height frequency and the height PDF for the molecules can be seen in Figures 3a and 3c respectively. Similarly for the linear speed (square root of each coordinates speed squared) frequency and PDF in Figures 3b and 3d. Finally, the plots of the potential and kinetic energies of the whole system (all molecules) at each iteration of the simulation can be seen in Figures 3e and 3f respectively. Since we have changing temperature every kilometre, we really have 20 Boltzmann distributions that connect at the boundary of each layer (since the Boltzmann distribution is for a given temperature). As a result of this, it is not obvious how to overlay a plot of the theoretical Boltzmann distribution onto our distributions for N_2 and O_2 without a time-consuming ugly mess of code so one is not included.

As we can see from Figure 3e, the potential energy of the system of molecules steadily increases until it reaches an equilibrium point after around $10,000$ iterations of the simulation. This is caused by the way in which the simulation implicitly exchanges energy with the heat bath of the Earth. The Earth acts as a heat bath since it is much larger than the system it is in contact with (the lowest level of the troposphere) and has more energy. This gives us that the molecules that interact with the ground increase their energy over time until they reach an equilibrium. Information sourced from UC Irvine Chemistry.

From the height distribution, we can see the overall shape of the Boltzmann distribution. This is in line with the relationship of pressure with altitude, where there are more particles at the bottom of the atmosphere and less as you increase in height. There are some noteworthy features of the distribution in Figure 3c. We see that the PDF abruptly stops while still having a significant number of particles in this region. This is because of the artificial limit placed in the code where if a molecule tried to move above the 19 kilometre limit of the troposphere, it was given a new random change in height. Also, the distribution does not fall off as dramatically as expected. This is a result of the dynamic temperature regions. When the code was tested with an isothermal troposphere, the height distributions were more exponential in shape.

From the speed distribution, we can see the overall shape of the Maxwell-Boltzmann distribution. The magnitude of the speed seems to be off by a constant (too slow as is) but the important factor is the shape. We see in the PDF of speed in Figure 3d that the N_2 and O_2 have almost the same shape. We expect that heavier molecules have more compact speed distributions, but given the small relative difference in mass between N_2 and O_2, it makes sense that the shapes don't differ very much. Taking the height PDF into account in Figure 3c, we see that most particles are in the lower troposphere, which is also where the warmer temperatures are. This means that the warmer more energetic molecules dominate the speed distribution where warmer molecules generate broader speed distributions. In contrast to the potential energy, the kinetic energy of the system throughout the simulation stays relatively constant. This is visible in Figure 3f.

The nature of a Monte Carlo simulation makes quantifying error challenging. To combat error, I used a large number of particles, small step sizes, and a long simulation time. In addition, I ran the simulations with various different seeds to ensure consistent results before choosing a seed that provided representative results. To combat mistakes in the code, I first simulated using an isothermal troposphere and only considering one molecule (N_2) before increasing the complexity. Calculations were done using numpy arrays and limiting operations like appending to decrease computation time. Calculations were done in order to minimize numerical errors (specifically when calculating the energies, it was calculated such that similar order of magnitude values were calculated with eachother before they were combined).

Finally, I wanted to use step sizes that took into account the different masses and abundances of N_2 versus O_2 (and consequently their varied interactions) but the only way I could think of doing this would be to use the equipartition theorem. The problem with this is that it can be derived from the Maxwell-Boltzmann distribution, so I decided against variable step sizes as I thought it would be less meaningful to get a Maxwell-Boltzmann distribution using the distribution itself. I was also unsure about how to calculate the probability of acceptance when moving to a higher energy state due to the temperature changes across regions. I tested using the temperature of the region the molecule started in and finished in and ended up getting results that had no discernible difference at all, so I decided to use the temperature of the region the molecule was traversing into.

4.2 Random Number Generators

All of the random number generators were tested over multiple runs of the simulations and the plots shown are the ones that I deemed to be representative of the average case of the simulation under the respective RNG.

Linear Congruential RNG: For this RNG, I used two parameter sets. The first was the set described in the section 10.1 of the course textbook that passes statistical tests with $a = 1664525$, $c = 1013904223$, and $m = 4294967296$. This RNG produced expected results which agree with the distributions of the Mersenne twister RNG. The set of plots for this parameter set can be seen in Figure 4. This is somewhat surprising that this simple implementation of an RNG is comparable to a much more sophisticated Mersenne twister, at least at this scale of a simulation.

The second parameter set was a truncated version of the first, with $a = 167$, $c = 1013$, and $m = 4297$. This was done to test the importance of choosing a good parameter set. The first thing to notice is that the maximal period of m in this is much smaller than the first parameter set. This is an issue, but there is a larger issue with this naive choice of parameters. The set of plots for this parameter set can be seen in Figure 5. The most noticeable change is visible in the height PDF. The most likely heights are those near the top and bottom of the troposphere. This can be explained by a biased RNG, specifically biased towards small numbers. We have that there are many more particles near the bottom of the troposphere which (RNG more likely to suggest downward move). We also have that any time the RNG produces a larger value to increase the energy of the molecule, it is more likely that the next value to decide acceptance is small, meaning that the move is more likely to accept this increased energy move. This implementation illustrates the importance of choosing a parameter set that produces unbiased outputs.

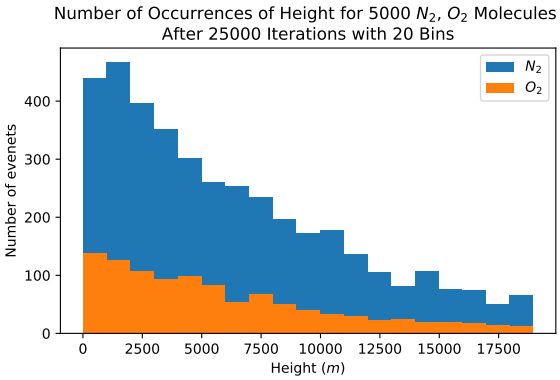
Inverse Congruential RNG: My field theory is a little lacking so I was unable to find a primitive polynomial of the form $x^2 - cx - a \in F_m$ which would provide a maximal periodicity of m for the RNG. I chose the largest prime m such that Python would not give an `inf` or `nan` in the output of the RNG, which happened to be $m = 137$. I brute forced some choices of a, c to find the maximal period I could. I settled on $a = 57$ and $c = 96$ which gave a measly period of 18 iterations.

This is obviously not good enough for a simulation of this scale, but given the restrictions of the method that the period is bounded above by choice of m, and using the simple method of finding inverses of x mod m to generate values described in the computational background, this is not very far away from the maximal hypothetical period in absolute terms. Other more sophisticated methods of finding inverses allow larger values of m and thus larger periods. However, these methods are considerably more computationally intensive and would make a simulation of this scale (on the hardware I have available) unrealistic.

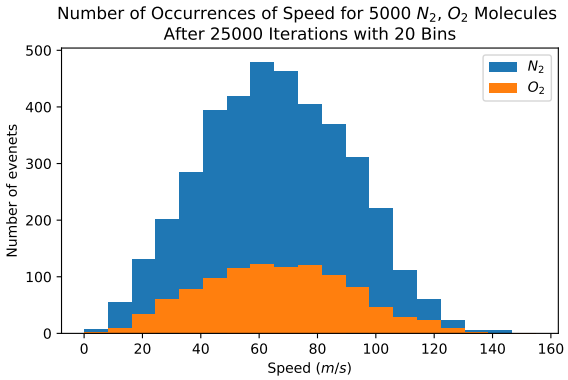
We can see from the set of plots in Figure 6 that this implementation of the RNG faces a more extreme version of the problem faced by the naive implementation of the linear congruential RNG in Figure 5. This RNG is heavily biased towards small outputs, and consequently results in higher speeds and heights closer to the top and bottom of the troposphere. The bias is strong enough such that it shows its effect in the speed distributions as well, whereas the naive linear congruential RNG's bias was only really noticeable in the height distribution.

5 Conclusion & Figures:

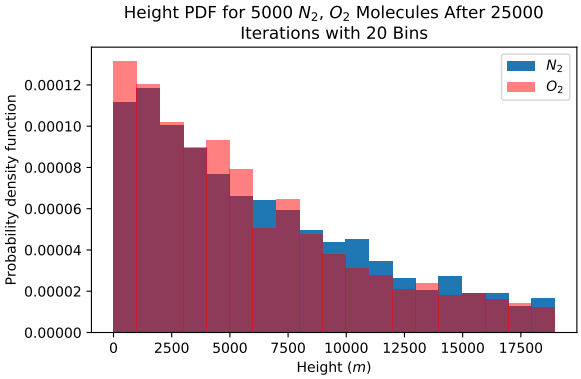
Since the problems in statistical mechanics are inherently random, the RNGs used to simulate these processes must be as close to truly random as possible. The consequences of a biased RNG makes for simulations with very large errors to the point where they lose their usefulness in predictive ability.



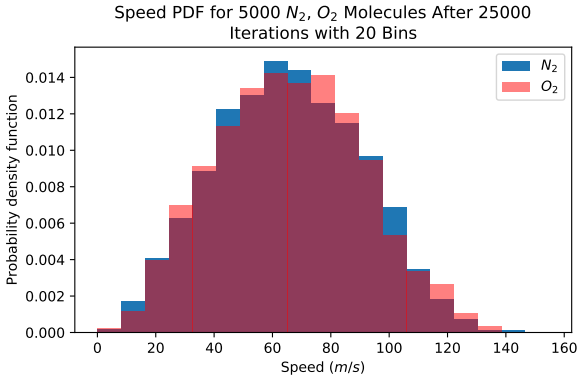
(a) Final molecule height frequency histogram



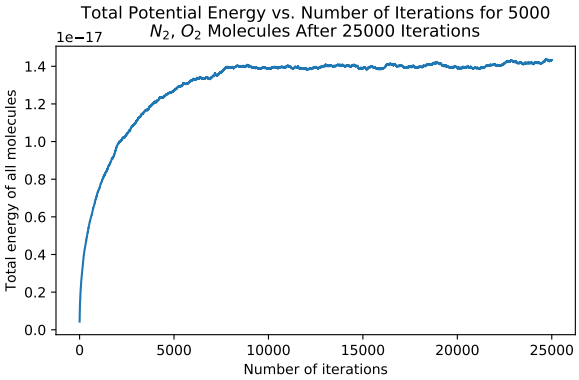
(b) Final molecule linear speed frequency histogram



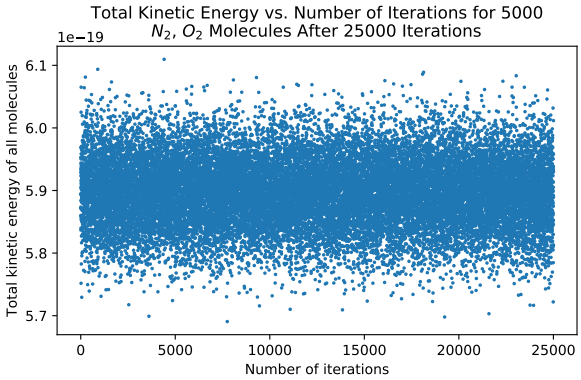
(c) Final molecule height probability distribution function



(d) Final molecule linear speed probability distribution function

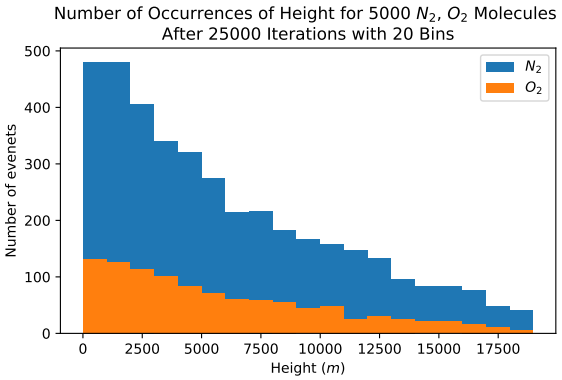


(e) Total potential energy of all the molecules throughout simulation

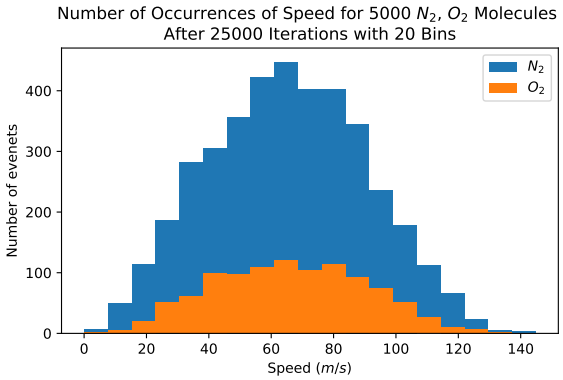


(f) Total kinetic energy of all the molecules throughout simulation

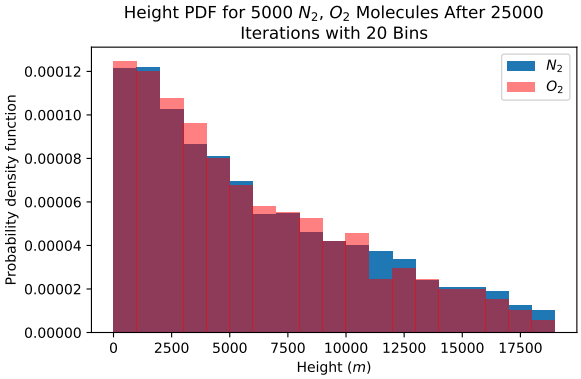
Figure 3: Final height and speed distributions as well as energy plots throughout Monte Carlo simulation using numpy's Mersenne Twister.



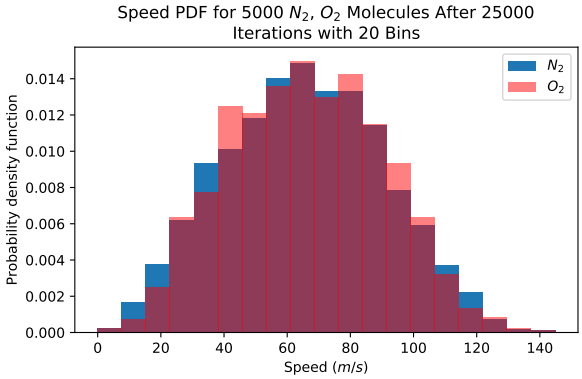
(a) Final molecule height frequency histogram



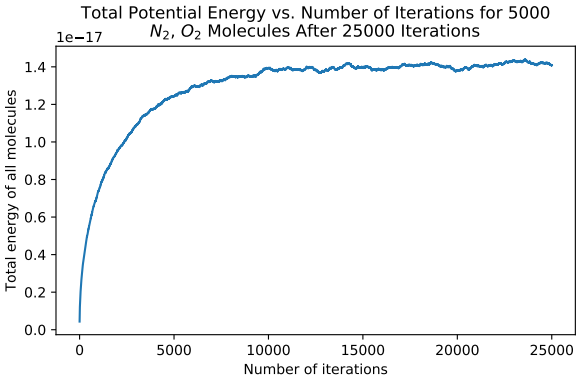
(b) Final molecule linear speed frequency histogram



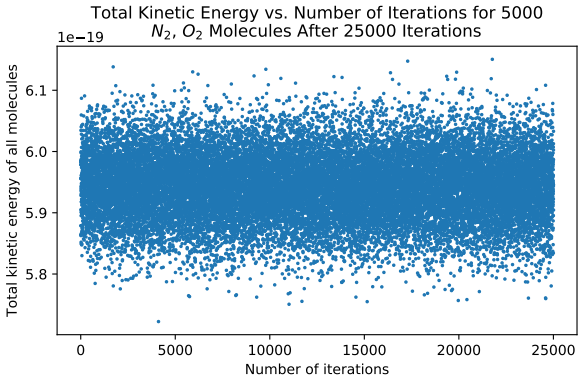
(c) Final molecule height probability distribution function



(d) Final molecule linear speed probability distribution function

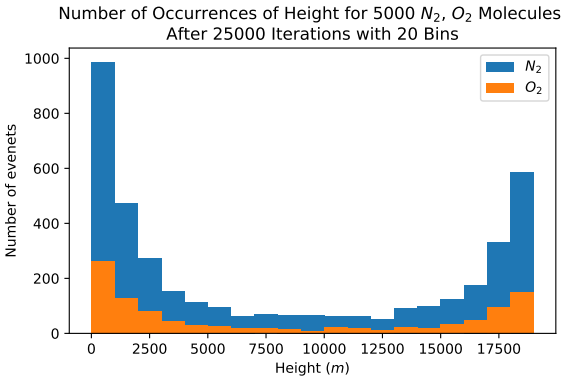


(e) Total potential energy of all the molecules throughout simulation

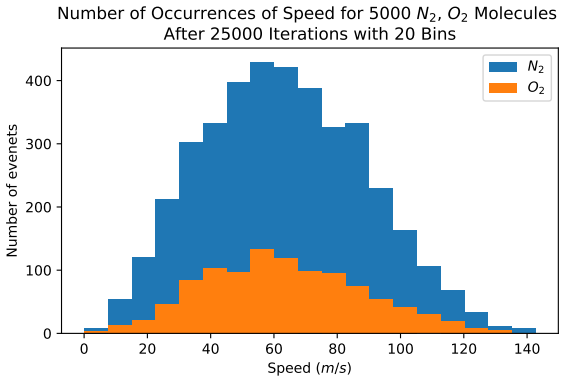


(f) Total kinetic energy of all the molecules throughout simulation

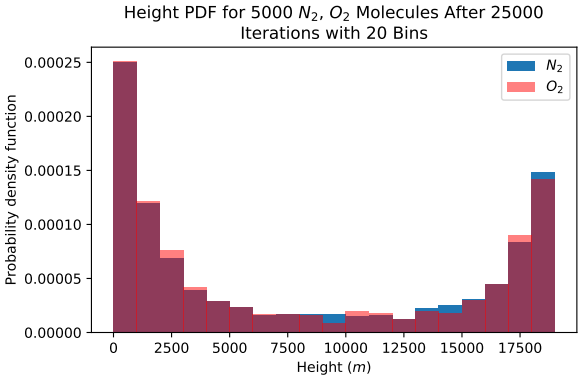
Figure 4: Final height and speed distributions as well as energy plots throughout Monte Carlo simulation using linear congruential RNG with good parameter set from textbook.



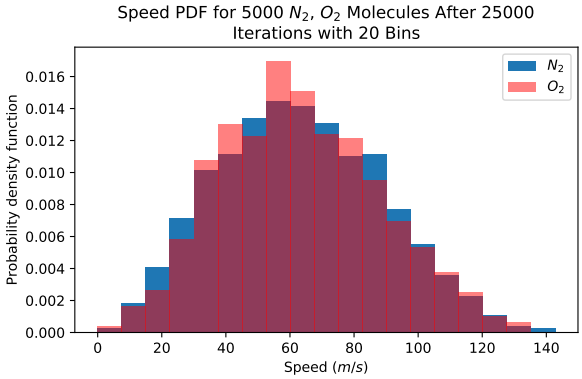
(a) Final molecule height frequency histogram



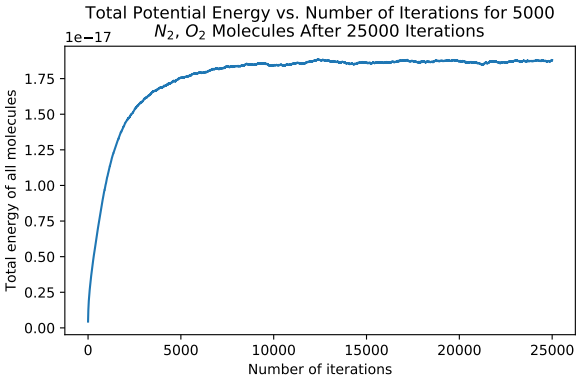
(b) Final molecule linear speed frequency histogram



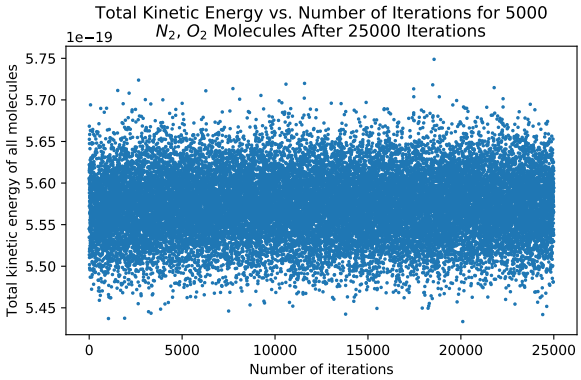
(c) Final molecule height probability distribution function



(d) Final molecule linear speed probability distribution function

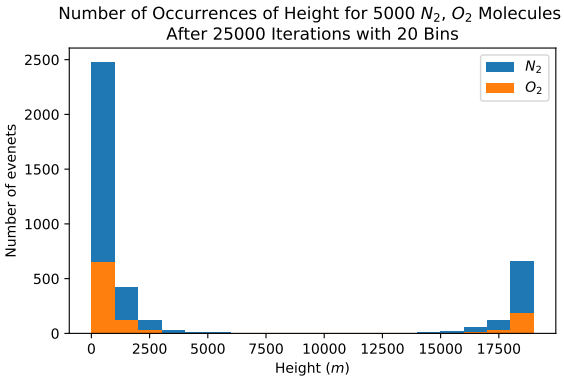


(e) Total potential energy of all the molecules throughout simulation

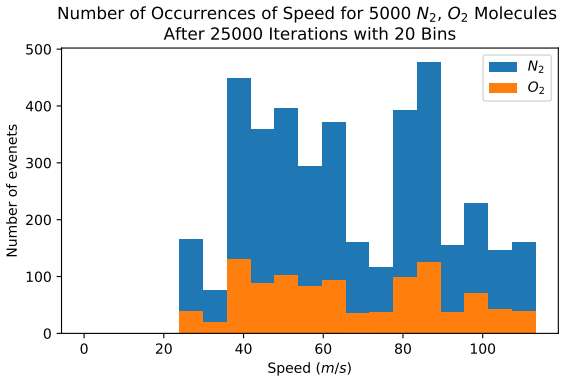


(f) Total kinetic energy of all the molecules throughout simulation

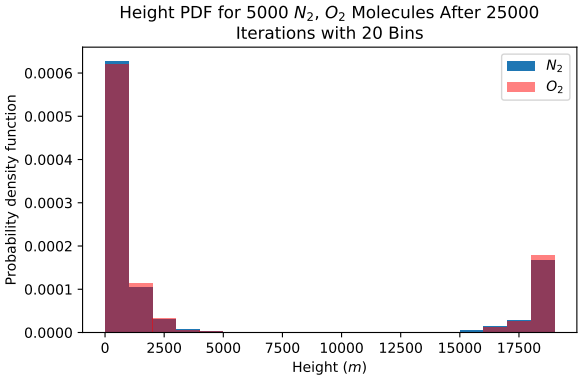
Figure 5: Final height and speed distributions as well as energy plots throughout Monte Carlo simulation using linear congruential RNG with naive parameter set.



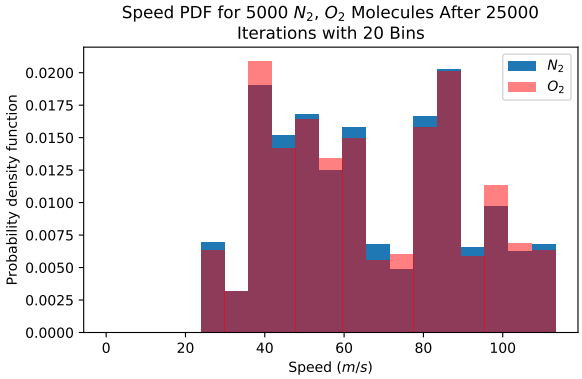
(a) Final molecule height frequency histogram



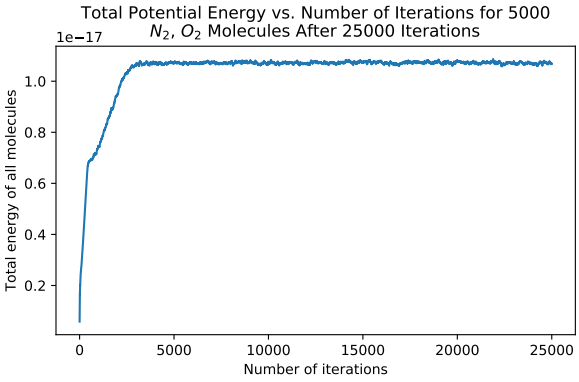
(b) Final molecule linear speed frequency histogram



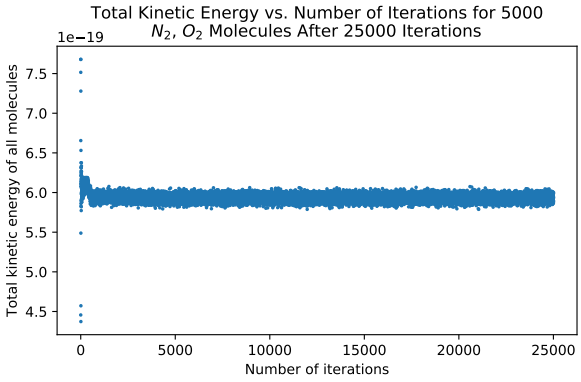
(c) Final molecule height probability distribution function



(d) Final molecule linear speed probability distribution function



(e) Total potential energy of all the molecules throughout simulation



(f) Total kinetic energy of all the molecules throughout simulation

Figure 6: Final height and speed distributions as well as energy plots throughout Monte Carlo simulation using inverse congruential RNG.