Okeefe Niemann
5/5/2014
1281465
PHYS 115

Molecular Dynamics

1 Introduction

In this assignment, a program was created to show how an interacting classical system, after reaching equilibrium, can have its probabilities of certain configurations represented by the Boltzmann distribution. The Hamiltonian of such a system is

$$E = \sum_{i=1}^{N} \frac{1}{2} v_i^2 + \sum_{i<j} V(x_i - x_j) \tag{1}$$

where the mass is of unity. It then follows that a particle in equilibrium has its velocity represented by the following distribution (Maxwell-Boltzmann).

$$P(v_{equil}) = \frac{1}{\sqrt{2\pi T}} exp\left(-\frac{v^2}{2T}\right) \tag{2}$$

where k_B is also of unity and the temperature of the system is measured by

$$T =< v^2 >_{equil} \tag{3}$$

This is due to the average velocity being equal to zero, and thus the variance of the distribution relying on $< v^2 >$. Following the conventions of the anharmonic spring (taylor expanding the potential and taking the derivative for the force), we find the force on a one dimensional array of N particles one unit length apart from each other to be

$$f(t) = -(t + t^2 + t^3) \tag{4}$$

2 Program

Using periodic boundary conditions (i.e. the array of particles $1 \rightarrow N$ loop backs back on itself in a circular configuration), we begin the system with the following conditions:

$$x_n(t = 0) = 0$$
$$v_n(t = 0) = sin\left(\frac{2\pi(n + \delta)}{N}\right)$$

In these equations, n represents the position of the particle relative to the others, δ represents a shift of any magnitude in order to combat against the inherent symmetries produced by the equations of motion, and N corresponds to the total number of particles in the system. These initial conditions are chosen to begin the system in a state where all energy is kinetic. The following program is then implemented to track the frequency of velocities for N particles observed over any given amount of time steps.

```
#include <stdio.h>
#include <math.h>
#include "verlet.h"


//function represents the force on each particle during simulation
double f(double y)
{
        return -(y + y*y + y*y*y);
}
```

```
int main(void)
{
        int N, n, i, j, t, g, T_EQ, T_RUN, bins;
        double h, f(), v_sq, N_U;
        double  v_avg, vsq_avg, vavg_min, vavg_max, v_one;

        FILE* fout;
        fout = fopen("harmonicpositionvelocity.txt", "w");

        h = 0.02;
        T_EQ = 800;
        N_U = 1 / h;

        //User inputs parameters of program
        printf("How many molecules in the system? ");
        scanf("%d", &N);
        printf("How many timesteps? ");
        scanf("%d", &T_RUN);
        printf("Amongst how many bins should the velocities be distributed? ");
        scanf("%d", &bins);

        double x[N], v[N], vavg_array[T_RUN], bin_val[bins], v_bins[bins], width, gaussian[bins];
        width = 6.0 / bins;

        //gives each particle in the system an initial velocity/displacement
        for(n = 0; n < N; n++)
        {
                v[n] = sin(2*M_PI*((double)n + 1.3) / (double)N);
                x[n] = 0;
                //fprintf(fout, "%f   %f\n", x[n], v[n]);
        }

        //below loop puts the system into equilibrium
        for(t = 0; t <= T_EQ; t++)
        {
                for(j = 0; j < N_U; j++)
                {
                        verlet(N, h, x, v, f);
                }
        }

        //runs the system and begins to record velocities
        for(t = 0; t < T_RUN;  t++)
        {
                v_sq = 0;

                //updates particles' positions and velocities N_U times per second
                for(i = 0; i < N_U; i++)
                {
                        verlet(N, h, x, v, f);
                }

                //distributes all velocities into the bins
                CreateBins(N, bins, width, v, bin_val, v_bins);
```

```
                //computes <v^2> for each time step
                for(i = 0; i < N; i++)
                        v_sq += v[i] * v[i] / (double)N;

                //computes the average of all <v^2>
                vsq_avg += v_sq/T_RUN;
        }

        double Norm = 0, AREA_GAUSS = 0, AREA_BINS = 0;

        //finds the normalization constant to fit the outputted data to a normalized gaussian curve
        for(i = 0; i <= bins; i++)
                Norm += width * v_bins[i];


        fprintf(fout, "T = %f\nNormalization Constant = %f\n\n", vsq_avg, Norm);
        fprintf(fout, "     v        P(v)     Gaussian Fit\n");


        //Computes the gaussian fit for each bin based off <v^2> and prints normalized frequency data
        for(i = 0; i <= bins; i++)
        {
                gaussian[i] =  ((1 / sqrt(2 * M_PI * vsq_avg)) *
                                exp(-(bin_val[i] * bin_val[i]) / (2 * vsq_avg)));

                //calculates the area underneath each normalized function for every bin
                AREA_GAUSS += gaussian[i] * width;
                AREA_BINS += (v_bins[i] / Norm) * width;

                fprintf(fout, "%f   %f    %f\n", bin_val[i], v_bins[i] / Norm, gaussian[i]);
        }

return 0;
}
```

In order to have motion and interaction in this dynamical system, we need to have an underlying algorithm which takes into account the force on each particle. For this project, we will be using the position verlet method.

```
verlet(int N, double h, double x[], double v[], double f(double))
{
        int i;

        //half steps of x
        for(i = 0; i < N; i++)
        {
                x[i] += 0.5 * h * v[i];
        }

        //Because v[N] = v[0], need to run both v[0] and v[N-1] outside of the "for" loop
        v[0] += h * (f(x[0] - x[1]) - f(x[N-1] - x[0]));

        for(i = 1; i < (N - 1); i++)
        {
                v[i] += h * (f(x[i] - x[i+1]) - f(x[i-1] - x[i]));
        }

        v[N-1] += h * (f(x[N-1] - x[0]) - f(x[N-2] - x[N-1]));
```

```
        //Finds position x[i] with updates on velocity v[i]
        for(i = 0; i < N; i++)
        {
                x[i] += 0.5 * h * v[i];
        }
}
```

This algorithm updates the particles' positions a half step forward with their current velocities. From there, the new velocities can be determined by the anharmonic force as a function of the half-step positions. With these new velocities, we can then fully update the particles' positions one whole step. The above loop for the updated velocities excludes $n = 0, N$ as a result of the periodic boundaries. In the main body of the code, this algorithm updates positions and velocities $\frac{1}{h}$ times for every time step to accurately track the trajectories of the particles.

Because we want the system in equilibrium before recording the particles' velocities, we must run the position verlet T_EQ time steps. The value that T_EQ takes on should be large enough such that the velocities begin displaying statistical tendencies. From there we can begin placing the recorded velocities into bins to find the frequencies for ranges of v.

```
double CreateBins(int N, int bins, double width, double v[], double bin_val[], double v_bins[])
{
        int i, j;

        for(i = 0; i <= bins; i++)
        {
                bin_val[i] = width * i - 3;

                for(j = 0; j < N; j++)
                {
                        if((v[j] > bin_val[i]) && (v[j] < (bin_val[i] + width)))
                                v_bins[i] += 1;
                }
        }
}
```

This "CreateBins" function creates a given amount of bins beginning at $v = -3$ and incrementing at a value $width = \frac{6}{bins}$ until reaching $v = 3$. The velocities from the particles for the given time step are then tested to see if their values lay a given bin width Δv. If a velocity satisfies the conditional, the value of the bin increases by one. While the array of velocities is rewritten every time step, the values of the bins continue incrementing.

Before the loop corresponding to one time step ends, a variable v_sq calculates the $<v^2>$ for the time step. Another variable vsq_avg finally adds the value of $<v^2>$ normalized by the number of steps to itself. From this, we can find the average of all values of $<v^2>$ and determine the temperature T of the system.

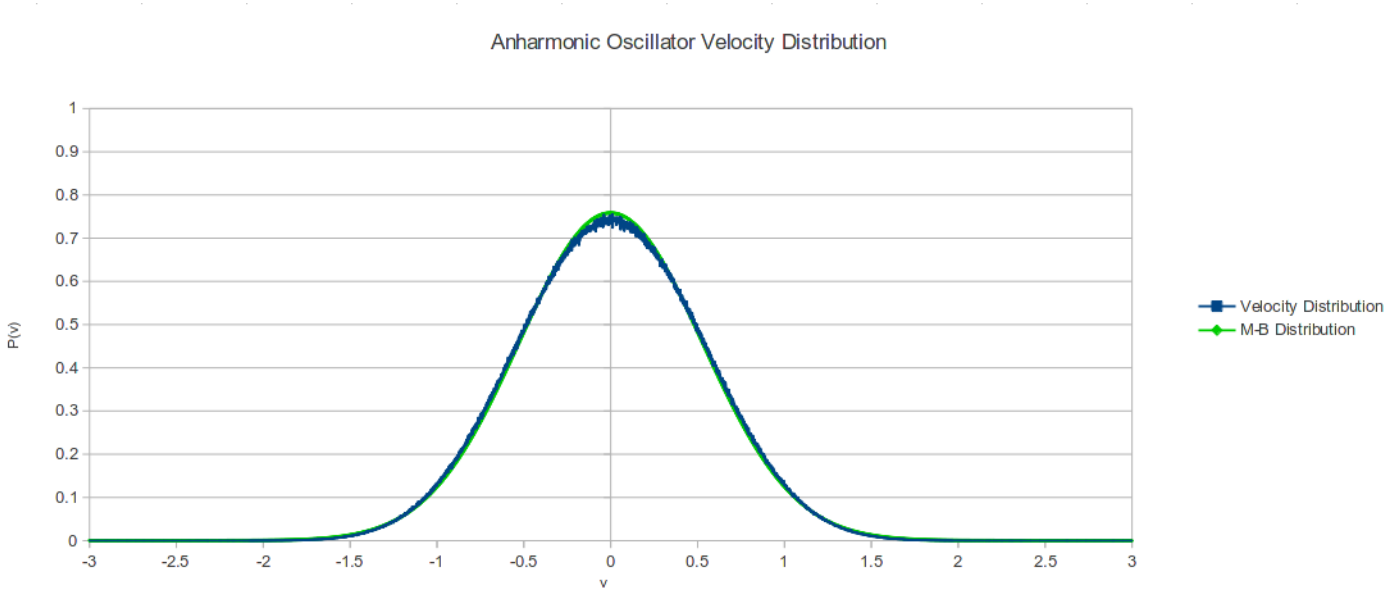
Now that T can be determined, we can apply it to equation (2), using the value of each velocity bin as our value for the v in the numerator of the exponential. We then compare our distribution of collected velocities with the Maxwell-Boltzmann constant, taking care to normalize our simulation values such that

$$\Delta v \sum_{i=1}^{bins} h_i = 1$$

3 Results

3.1 Anharmonic Oscillator

The results for the anharmonic oscillator yielded the following graph:



$N = 20$; T_RUN=10^6; $bins = 5000$

As expected, equation(3) yielded a Gaussian distribution almost identical to that of the Maxwell-Boltzmann distribution. The temperature of the system was found to be

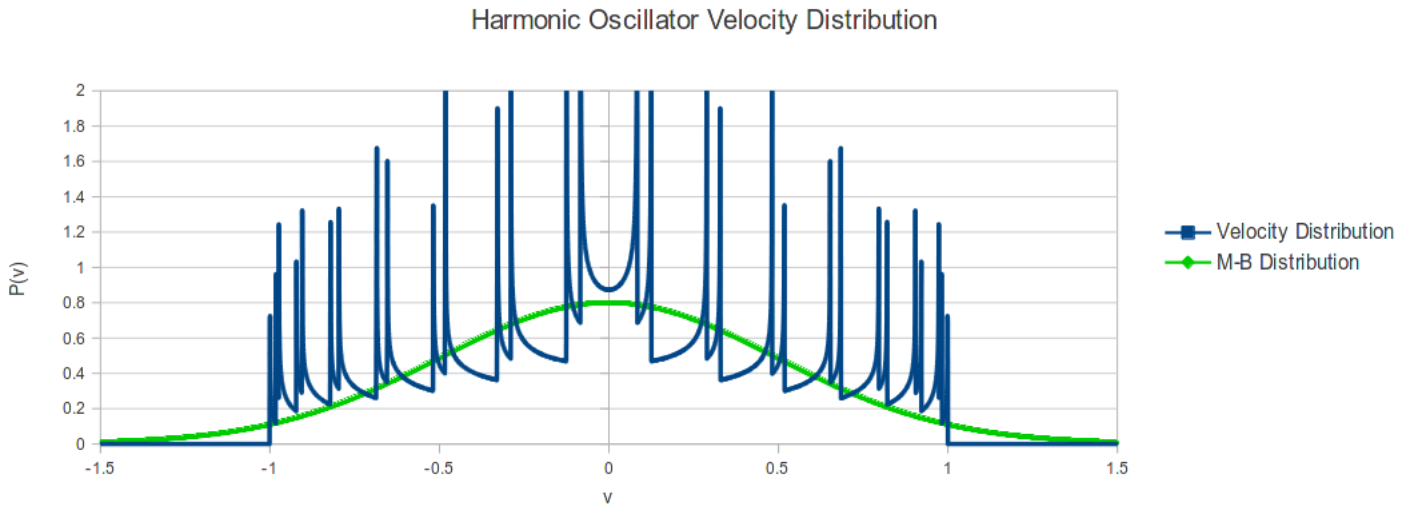
$$T = 0.271153$$

3.2 Harmonic Oscillator

Equation (3), if trunkated after the first term, produces a simulation which can be likened to a harmonic oscillator. In this configuration, all the modes of the particles oscillate at independent frequencies given by

$$\omega_m = 2sin\left(\frac{m\pi}{N}\right)$$

Observing how the velocity distribution compared to the Maxwell-Boltzmann distribution, I ran the simulation and plotted the results.



$N = 15$; T_RUN=10^6; $bins = 10000$

The visual representation of the data confirms not only that the velocity distribution neglects to follow the Maxwell-Boltzmann distribution, but proves by the sharp increase in certain velocities that the modes of oscillation are independent from each other. We can see this by the 15 symmetric "spikes" on the graph corresponding to the maximum

velocities when each of the N particles' energy is entirely kinetic. One can also find that the slope of the bins in between the maximum velocities correspond to the particles' accelerations as they oscillate in their different modes. The average temperature of the system was found to be

$$T = 0.250000$$