ABSTRACT

NUMERICAL SIMULATIONS OF COLD ATOM RATCHETS IN DISSIPATIVE OPTICAL LATTICES

by Anthony Paul Rapp

Brownian ratchets are interesting nanodevices capable of performing useful work by extracting energy from surrounding fluctuations, such that under certain conditions an increase in noise level can result in increased efficiency. Cold atoms confined in an optical lattice may serve as ideal candidates for investigating the basic physics behind Brownian ratchet efficiency dependence on environmental noise. In this thesis we implement detailed semi-classical Monte Carlo simulations to model the asymmetric diffusive motion of an $F_g=1/2 \leftrightarrow F_e=3/2$ atom in a 1D optical lattice in which one of the beams is phase-modulated in order to create a ratchet. We closely follow the treatment already shown by Martin Brown in his doctoral thesis [M. Brown, PhD thesis, University College London (2008)] and the referenced results therein. The results of simulations for ratchets using different types of driving modulations, such as biharmonic and multi-frequency, are presented as well as the case of a gating ratchet. All codes for our simulations are included.

NUMERICAL SIMULATIONS OF COLD ATOM RATCHETS IN DISSIPATIVE OPTICAL LATTICES

A Thesis

Submitted to the

Faculty of Miami University

in partial fulfillment of

the requirements for the degree of

Master of Science

by

Anthony Paul Rapp

Miami University

Oxford, Ohio

2019

Advisor: Dr. Samir Bali

Reader: Dr. Imran Mirza

Reader: Dr. Edward Carlo Samson

©2019 Anthony Paul Rapp

This Thesis titled

NUMERICAL SIMULATIONS OF COLD ATOM RATCHETS IN DISSIPATIVE OPTICAL LATTICES

by

Anthony Paul Rapp

has been approved for publication by

The College of Arts and Science

and

Department of Physics

_____

Dr. Samir Bali

_____

Dr. Imran Mirza

_____

Dr. Edward Carlo Samson

Contents

List of Figures

ACKNOWLEDGMENTS

I would first like to thank my advisor for giving me the opportunity to work on this project. Alex Staron and David Cubero were also essential in helping me understand how these simulations were going to be made, along with many of the initial steps into the process. I would also like to thank all the friends I've made within the department during my time at Miami. The in-class help and late-night data runs have been invaluable, and have helped shaped me into the physicist I am today. I would lastly like to thank my family for their continued support from the beginning of this road and beyond.

Chapter 1

Introduction

Brownian ratchets are "devices that rectify microscopic fluctuations, thus producing useful work out of a fluctuating environment. Though many biological, naturally-occurring Brownian ratchets have been identified and studied [3], their dynamics are still not well understood because of the complexity of biological systems. These ratchets are of particular interest because of the small scale they operate on. On a molecular level movement is not a simple matter, as collisions from every surrounding particle have significant impact. Artificial devices on this level struggle to achieve within orders of magnitude of the natural examples. One of the best efforts so far has seen 10^{18} nanomachines spend 12 hours to produce a single amide bond, while natural ribosome can produce 15-20 such bonds per second [4].

In this thesis I present simulation results for producing a Brownian ratchet using cold atoms in dissipative optical lattices. Cold atoms are exceptionally flexible for the control that experimenters have over them. By creating these simulations, we first predict where a cold atom ratchet might be very efficient, and second can help characterize results from experimental efforts.

This thesis will begin with a description of ratchets. We will cover the three necessary conditions to achieve ratcheting action, as well as laying out the conditions that will forbid a net motion. We will then describe the simulations that were run, beginning with a basic simulation of a non-ratcheting lattice. Each ratchet will then be described and results of the simulations shown. We will compare these results to established results.

In the appendices, we will go through many of the theoretical steps necessary for setting up the semi-classical simulations. I'll also describe some of the missteps that I had taken in developing the simulations, how those mistakes appear, and what is needed to correct them.

All of the simulations presented here can be found on GitHub at https://github.com/TrackPadMaster where the four simulations are each broken off into their own sections. Much of the code will be presented at the end of this thesis as well, but I strongly recommend downloading the files as they were used and commented.

These simulations and the efforts leading up to them have been presented at external conferences and workshops.

- "Noise-enabled ratchets in cold atom dissipative lattices", Poster, 20th Annual Southwest Quantum Information & Technology (SQuInT) workshop, Albuquerque, NM, Feb. 22-24, 2018
- "Numerical simulations of Brownian ratchets in dissipative optical lattices", Poster, 50th Annual Meeting of the APS Division of Atomic, Molecular, and Optical Physics (DAMOP), Milwaukee, WI, May 27-31, 2019

Chapter 2

What is a ratchet?

A ratchet is a device that extracts a net motion from no net force. This is not unlike the tool ratchet one might find in a garage. Motion is allowed in one direction (the preferred direction) while motion contrary is blocked (the opposed direction). However, motion on a molecular or atomic scale is not so simple as turning a crank. For a particle at this level, every surrounding particle can have a significant impact on motion. Even if a molecule had some means to propel itself, the Brownian motion and resulting collisions still dominate.

Yet directed motion is still possible at this level. Nature has provided such an example in the kinesin protein found in cells [3]. This is a protein that must compete with water molecules of comparable size to deliver packages within the cell in a surprisingly efficient manner. How might this same efficiency be achieved in an artificial device? How might this device even be created? Let's first look at a model that can describe the necessary characteristics to make a ratchet.

2.1 The Feynman-Smoluchowski Ratchet

Smoluchowski in 1912 described a nanodevice-based thought experiment to illustrate the difficulty in extracting useful work from Brownian noise. Feynman illustrated, as in Fig. 2.1, how this ratchet could still do work. The Feynman-Smoluchowski has seen an experimental realization [5], despite originally being only a thought experiment. We begin by submerging our device in a thermal bath. This device is small enough that each collision from a thermal particle results in a significant shift. As the particles collide with the paddles, the shaft turns about randomly as in a Brownian walk. This Brownian walk, if given enough time, will not produce any net motion though. It's movement is totally random, and its average position over a long enough time is just the starting position.

Asymmetry is then introduced to the shaft. By attaching a gear and pawl to the end opposite the paddles, we attempt to stop motion in the opposed

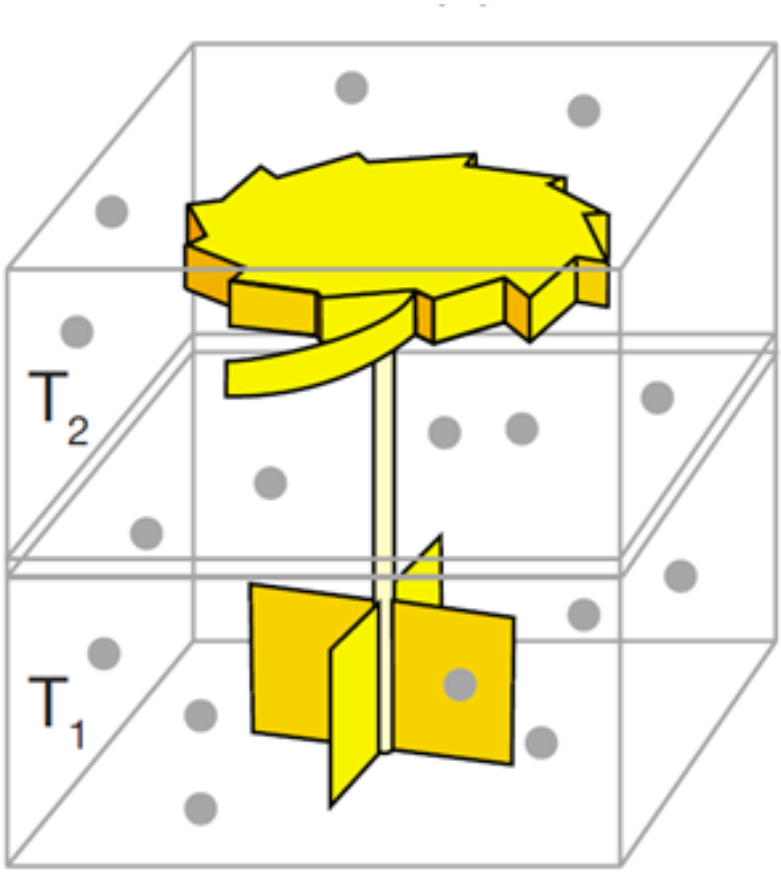

Figure 2.1: The Feynman-Smoluchowski ratchet, as illustrated in Hänggi [1]. The three components are shown: First, we have Brownian particles belaboring the paddles. Second, asymmetry is created by introducing the pawl. Lastly, there needs to be thermal non-equilibrium, where T_1 is higher than T_2.

direction, isolating out the preferred motion. This seemingly violates the second law of thermodynamics. In this ideal picture, we extract work from where there was no order. We have neglected to discuss the thermal bath also acting on the gear and pawl though. So long as both ends of the shaft are submerged in the same thermal bath, the pawl and gear will also constantly be jostled by the surrounding particles. When all facts are considered, we still are unable to achieve any net motion.

Were we to separate each end of the shaft, we could reduce or eliminate the collisions jostling about the pawl and gear. In Feynman's lectures [6], it's shown that by breaking the equilibrium of the device, placing the paddles in

a hot thermal bath and the gear in a cold bath, we are finally able to realize motion in our preferred direction. What has been created is a heat engine, where energy travels from the hot bath through the ratchet into the cold bath.

This picture demonstrates the three necessary conditions for any ratchet. There must be a source of Brownian noise to create the random motion. There must be a breaking of an existing symmetry to influence the direction of motion. Finally there must be a driving out of equilibrium so that work can actually be done.

2.2 Motion and symmetry

We want to describe the motion of a particle and how we might see a directed motion over a period of time. To do this we need to explore the dynamics of a particle and the different conditions that might allow or forbid a net motion. We'll begin by describing those dynamics and affecting conditions. We will keep this description isolated to one dimension for simplicity. Then we can discuss the transformations of time and space, and the different labels of periodic functions that we can assign to our system. Following that we establish how our forces and potentials of the system will appear, and the transformations that will forbid a net motion.

2.2.1 Langevin equation

Let's begin by describing the dynamics of a single atom classically. This atom has a damping from the environment γ as well as random fluctuations $\xi(t)$. This damping is linear and dependent on the velocity of the atom through the environment. Then we can describe the dynamics of the atom as

$$m\ddot{x} = -\gamma\dot{x} + \xi(t) \tag{2.1}$$

We require that the random noise acting on the atom is unbiased, so

$$\langle \xi(t) \rangle = 0 \tag{2.2}$$

The fluctuation-dissipation relation for Brownian noise will also dictate that

$$\langle \xi(t)\xi(t') \rangle = 2\gamma k_b T \delta(t - t') \tag{2.3}$$

Two features that we find in ratchets are a driving force $F(x,t)$ and a potential $V(x,t)$. We can include them into our equation of motion to realize a Langevin equation.

$$m\ddot{x} = -\frac{dV(x,t)}{dx} + F(x,t) - \gamma\dot{x} + \xi(t) \tag{2.4}$$

Depending on the degree of damping, three situations arise.

If the system is underdamped the friction term $\gamma\dot{x}$ is small and is not a dominant term for the equation. It's not necessarily small enough to neglect, but other terms will still have significant impact to the motion of the atom.

If the system is instead overdamped, $\gamma\dot{x}$ is large and friction dominates the equation. In this limit we can take $m = 0$. Inertial effects then do not affect the atom.

A system is Hamiltonian if there is no noise or dissipation in the system. That means that both $\gamma\dot{x} \approx 0$ and $\xi(t) \approx 0$.

2.3 Descriptions of symmetry

We wish to study the symmetries that exist within the lattice environment and the driving force acting on the atom. We must first be able to describe each. We begin with two possible inversions we can define for our system:

- Spatial inversion $x + x' \to -x + x', \quad t + t' \to t + t'$
- Temporal inversion $x + x' \to x + x', \quad t + t' \to -t + t'$

There are three specific cases of a function $f(y)$ with period Y that we can describe.

- $f(y)$ is symmetric (f_s) if $f(y + y') = f(-y + y')$
- $f(y)$ is anti-symmetric (f_{as}) if $f(y + y') = -f(-y + y')$
- $f(y)$ is shift-symmetric (f_{sh}) if $f(y + \frac{Y}{2}) = -f(y)$

Here y' is a constant about which we define the function. That is, if $f(y)$ is symmetric, it is symmetric about y'.

We can simplify our description of the ratchet environment by assuming that the potential is not changing with time $V(x,t) \equiv V(x)$ and the driving force is uniform across space $F(x,t) \equiv F(t)$. We require that both the potential and the driving force are periodic, where the potential has period X and the driving force period T. Now there are three possible cases where a net motion for the atom is forbidden:

- $\hat{S}_A$: $x \to -x + 2x'$, $t \to t + \frac{T}{2}$; for $\frac{dV}{dx}_{as}, F_{sh}$

For $\hat{S}_A$ to be true, the derivative of the potential is anti-symmetric and the driving force shift-symmetric. Since only the potential has spatial dependence and only the driving force has temporal dependence, we can treat each transformation separately. If both meet the necessary conditions, then net motion is forbidden

- $\hat{S}_B$: $x \to x$, $t \to -t + 2t'$; for $F_s, \gamma = 0$

$\hat{S}_B$ is only true when there is no damping acting on the atom. If the driving force acting on the atom is symmetric, then no matter what the shape of the potential, net motion will be forbidden.

- $\hat{S}_C$: $x \to x + \frac{X}{2}$, $t \to -t$; for $\frac{dV}{dx}_{sh}, F_{as}, m = 0$

$\hat{S}_C$ is a relevant transformation in the overdamped limit where $m = 0$. Net current is forbidden for a shift-symmetric potential and an anti-symmetric driving force.

2.4 Types of ratchets

When discussing cold atom ratchets there are two primary types to achieve the three conditions [7].

2.4.1 Flashing ratchet

The first type of ratchet is the flashing ratchet. This ratchet needs no external driving force, but instead drives itself through the shape of its potential. The Feynman-Smoluchowski ratchet is an example, because it doesn't rely on an outside force. It uses an asymmetric potential in the form of the gear and pawl. Likewise in cold atoms, we take what was a symmetric potential and break the symmetry.

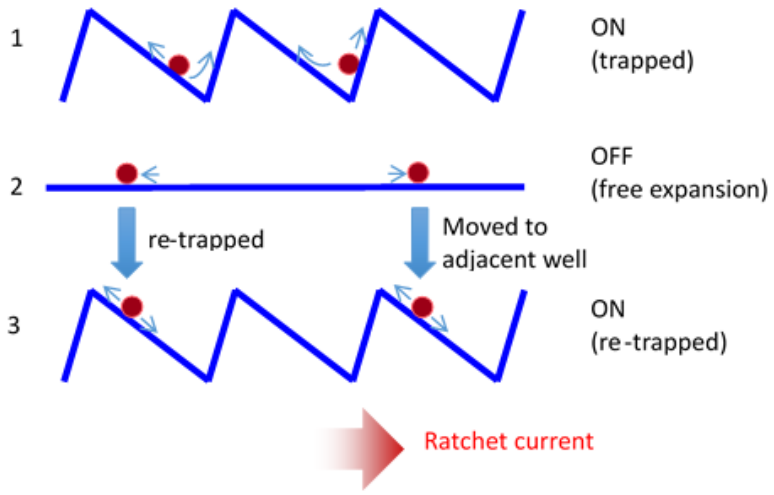

Figure 2.2: The flashing ratchet with a sawtooth potential. No ratcheting force is used in this regime. The symmetry is broken in space to achieve net motion.

One such broken symmetry is the sawtooth potential as in Fig. 2.2. For the atoms loaded into the potential, let's assume they have a random distribution of velocities. When the potential is turned off, they disperse and travel proportional to those velocities. The potential is turned back on and the atoms are retrapped. An atom moving in the preferred direction has a much smaller distance to cover than an atom moving in the opposed direction. Since these velocities are pulled from a random distribution, the likelihood of being able to move in the preferred direction is greater than to move in the opposed direction, and we get a net motion.

We are also able to see a current reversal, or change in the preferred direction, by altering how we shape the potential. If the sawtooth is instead reversed, so too is the preferred direction.

Realize that by actively turning on and off the potential we put energy into the system. Waste energy is realized as an atom is likely to gain energy by jumping to the next well in the preferred direction. For example, begin with an atom oscillating at the bottom of a well. The potential is turned off, the atom moves in the preferred direction and the potential returns. When the potential returns the atom is much more likely to be at a higher potential than a lower one. Now with a restored potential the atom is oscillating with much more energy. A non-equilibrium must exist here to continue cooling the atoms to keep them trapped in the wells.

2.4.2 Rocking ratchet

The rocking ratchet doesn't rely on an asymmetric potential but instead requires an asymmetric driving force acting on the atoms. A biharmonic, or more complex, driving force is required. One example is shown in Fig 2.3.

We begin with the potential in a tilted position with atoms loaded into the wells (Step 2 in Fig. 2.3). A very sudden tilt in the preferred direction is applied, causing some atoms to "spill" into the next well (Step 3). From here, the potential is very slowly tilted back to its starting orientation (Step 4). This very slow tilting is not enough to push atoms in the opposed direction. The potential is periodic in this tilting, so the net force over a full cycle is zero.

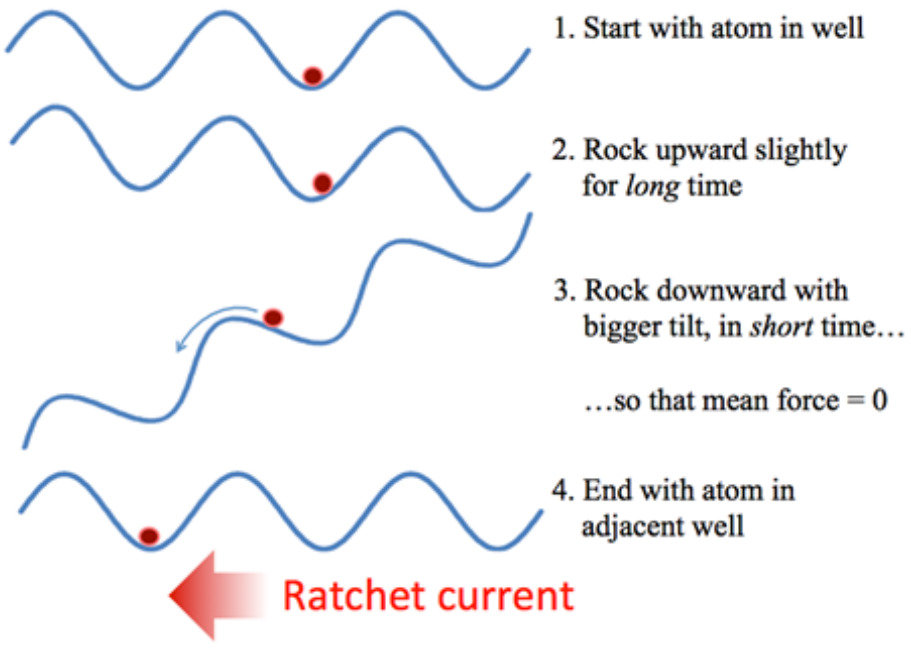

Figure 2.3: The rocking ratchet with a symmetric potential. A ratcheting force is necessarily added to achieve net motion over a full period.

We can determine the preferred direction for the rocking ratchet by simply slightly changing the driving force. Should the sharp tilt occur instead in the opposite direction, the current of atoms will reverse.

2.4.3 Gating ratchet

The gating ratchet, sometimes known as the information ratchet, is a combination of the flashing and rocking ratchets [7]. The potential being flashed does not necessarily have to be asymmetric. The gating ratchet attempts to synchronize the driving force and the off-state of the potential so that the atoms move unobstructed in the preferred direction. When the force pushes in the opposed direction, the potential rises again to stop the motion of the atoms. However, these two features are operated independently, and frequencies and amplitudes of each can be tuned and explored.

2.5 Current reversals

We have recognized in the examples how we might reverse the direction of the net motion. We describe this as a current reversal, and it's a characteristic of most ratchets. Through a change of parameters for a ratchet, we might see the direction of net motion in that ratchet reverse directions.

2.6 Cold atom ratchets

Cold atoms began as a field of study before becoming a tool in exploring quantum optics phenomena. Cold atoms in optical lattices are easy to work with because of the complete control afforded to the experimenter. There are two parameters of the lattice that are extremely easy to manipulate, detuning Δ and intensity I_o. These can be used to scale the potential well depth $U_o \propto I_o/\Delta$ and the scattering rate $\Gamma' \propto I_o/\Delta^2$. We can study cold atom ratchets as functions of these two parameters.

Chapter 3

Simulations

This chapter begins with a brief summary of the treatments of the atom-light interaction, followed by the results of the semi-classical approach. We start by showing the results of a Monte-Carlo simulation of the lattice without ratcheting to build confidence in the simulation initially before expanding the simulation to ratcheting action. We demonstrate three different ratchets acting on the cold atoms. We closely follow the treatment in Ref. [2] from the University College London group which is a pioneer in the field of cold atom ratchets.

3.1 Treatments of the atom-light interaction

There are three general views one can take when simulating atoms interacting with light. Though all three share some common beginnings analytically, the final equations, behaviors, and computation time do have quite a bit of variability. Rather than solve through each method individually and describe how they would then be plugged into the simulation, I'm only going to give an overview of the method and the advantages of each.

3.1.1 Classical

The classical atom and classical environment is the simplest to create and describe. This treatment sees the atom as having no internal states. Further, the position and momentum of the atom are treated classically. Atoms in this regime still feel a force from a potential, but that potential is taken from the semi-classical approach, and since the classical atom has no internal states, only a single potential is considered. From the semi-classical derivation, it's assumed that the atom would spend most of its time sitting in wells in one of the ground states, so the classical view is not immediately absurd. The starting conditions of the atom are randomly sampled, so it will still undergo diffusion while moving through the potential, not simply going through the same trajectory in every iteration of the simulation.

In conditions where jumps between the opposing wells of the bipotential is suppressed, this could provide a relatively accurate picture of the atoms within the experimental optical lattice. One major fault with this model is that the temperature of the atoms after spending some time in the lattice won't be accurate. Unable to go through any Sisyphus cooling [8], the classically simulated atom has no means to broadly change the temperature it started with, as temperature is directly related to velocity on an atom-by-atom basis. There is no ongoing Brownian noise acting on the atom either. Since the atom will never spontaneously emit, the random kicks of momentum are not present in the classical treatment.

The classical treatment is a very straightforward model to create though, and compared to other methods, is very quick to run through computationally. It is even easier to visualize than the semi-classical atom, hence it was used to first illustrate types of ratchets. For short trajectories it could be used as a proof-of-concept model to showcase how a ratchet using cold atoms is realized. However, in longer spans, there is waste energy that manifests as heat which a real experiment would remove by Sisyphus cooling. It is also extremely important to pick accurate initial conditions for the atom, since having too much or too little momentum cannot be rectified. Atoms with too little momentum will never be able to escape from the wells they are trapped in, while atoms with too much will largely ignore the potential and continually fly off in one direction.

3.1.2 Fully Quantum

The fully quantum approach demands that the atom has quantized internal states in a quantized field. The momentum and position of the atom are treated as quantum variables. The dynamics of the atom are determined by a master equation. This is the most representative of the evolution of the atom, but also the most computationally intensive. The ratcheting force is also not so easily introduced as in the semi-classical case. The equations are lastly not so easy to visualize like the semi-classical treatment. We don't get the picture of the bipotential so frequently used to describe the behavior of the atom. Examples of the derivation and how it is implemented into a Monte-Carlo simulation can be found in [9, 10].

3.1.3 Semi-classical

The semi-classical treatment lets the internal states of the atom stay quantum, but treat the light field classically. We therefore use the Bloch equations, calculated using a classical electric field. Next, we take a Wigner transform of the density matrix of the atom, which is time intensive for the physicist, but the solution is not complicated for a computer. That solution is a Fokker-Planck equation, which can identify changes in momentum owing to forces and diffusions acting on the atom. This is not entirely different from the classical simulation in implementing, and it is also simple to add in new driving forces, as we would to represent the ratchet.

One major difference from the classical treatment is that the semi-classical view still retains the bipotential (or more for higher-level atoms) and can account for Sisyphus cooling. For having those internal states, we've seen a few more equations must be added to the process past just the equations of motion, as it must be determined in a time step if the atom is going to change states or not. This means that the semi-classical simulation is slightly more intensive than the classical simulation but is a reasonable compromise compared to the fully quantum approach. Some other examples of the derivations can be found in [8, 11, 12].

By recreating average temperatures of atoms in different optical lattices that are comparable to experimental results, we establish trust that this semi-classical approach acts as a faithful recreation of the experiment.

3.2 Steps of integration

Once the approach has been selected, the next step is to create the Monte-Carlo simulation that will create different trajectories for the atom. The key feature of the Monte-Carlo simulation is that each trajectory is produced randomly from the governing equations. For every step of the integration, there are a number of possible paths that the atom might take. Every path has its own probability of occurring, which is calculated in each step. A random number generator (RNG) is then compared against the probabilities and a path is selected. In the simulations performed and presented here, the RNG of the Ubuntu operating system was used at each step for each process of the simulation. More discussion is included within the codes, but one should first test the RNG of the system used to simulate. Each process should see its own unique string of numbers or else the trajectories of atoms may not be unique and independent.

Let's begin by stating the equations derived in Appendix A that are used for these simulations. We can express a change in the momentum Δp over a time step dt in two different cases. The atom sits in a bipotential like shown in Fig. 3.1. If in the time step dt the atom stayed in the same ground state we have

$$\Delta p_{\pm} = -\frac{dU_{\pm}}{dz}dt + \sqrt{2D_{\pm\pm}dt}N(0,1) \tag{3.1}$$

where $U_{\pm}$ is the bipotential acting on the atomic ground states. On the other hand, if in that time step the atom changed states, we have

$$\Delta p_{\pm} = -\frac{dU_{\pm}}{dz}dt + \sqrt{\frac{2D_{\pm\mp}}{\gamma_{\pm\mp}}}N(0,1) \tag{3.2}$$

In both equations the terms selected are determined by the internal state that the atom begins in, whether the $+\frac{1}{2}$ or $-\frac{1}{2}$ state. $N(0,1)$ is a sample from a random distribution with mean 0 and variance 1. The diffusion terms are

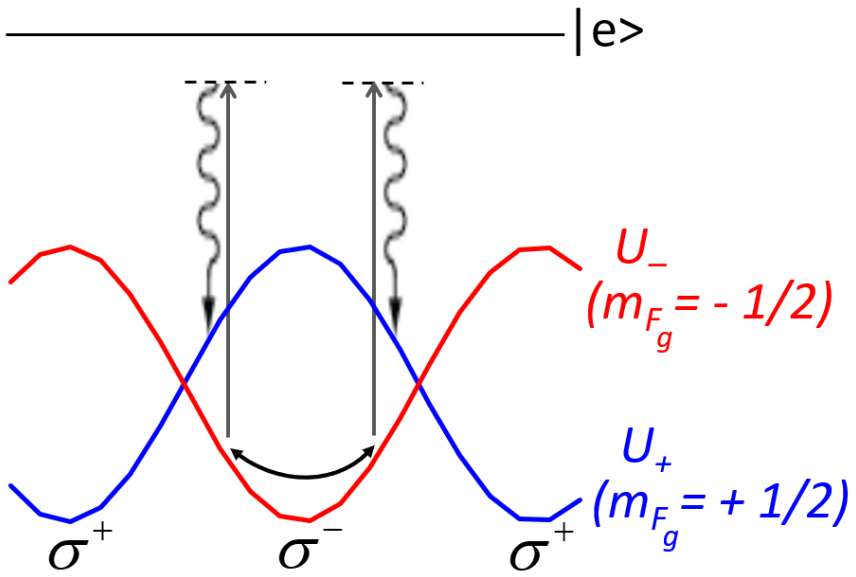

Figure 3.1: A small sample of the one-dimensional bipotential. As the atom climbs the sides of the well, it is more likely to become excited and may spontaneously emit to the other ground state.

defined as

$$\begin{aligned} D_{\pm\pm} &= \frac{\hbar^2 k^2 \Gamma'}{90}(35 \pm 7\cos 2kz) \\ D_{\pm\mp} &= \frac{\hbar^2 k^2 \Gamma'}{90}(6 \mp \cos 2kz) \end{aligned} \tag{3.3}$$

where the left subscript determines the initial atomic state ($\left|g_{+1/2}\right\rangle$ or $\left|g_{-1/2}\right\rangle$) and the right subscript the final atomic state. Γ' is the spontaneous noise scattering rate, dependent on intensity and detuning, given as

$$\Gamma' = \Gamma s_o \tag{3.4}$$

where s_o is the saturation parameter.

$$s_o = \frac{\Omega^2/4}{\Delta^2 + (\frac{\Gamma}{2})^2} \tag{3.5}$$

Ω_o is the Rabi frequency. The probability of the atom changing state is given by

$$\begin{aligned} \gamma_{+-} &= \frac{2}{9}\Gamma' \cos^2 kz \\ \gamma_{-+} &= \frac{2}{9}\Gamma' \sin^2 kz \end{aligned} \tag{3.6}$$

The force from the potential is given by

$$\frac{dU_{\pm}}{dz} = \frac{d}{dz}\Big(\frac{U_o}{2}(-2 \pm \cos 2kz)\Big) = \mp U_o k \sin 2kz \tag{3.7}$$

These equations describe the probability that the atom in a particular state was pumped by the lattice field and underwent spontaneous emission into the other ground state. The probability that this sequence occurred in any one time step is $\gamma_{\pm\mp}dt$.

One issue that arises in running the simulation is the momentum contribution from the diffusion term, specifically when the atom changes states. If γ is sufficiently small, this can cause sudden unrealistically drastic changes to the momentum. Different groups have taken different approaches. Some have simply neglected the term because of its rare occurrence and small impact, others place a cap on the highest velocity allowed for the atoms.

In our simulation we will exclusively use Eq. 3.1. Since the occurrence of a jump is set to roughly once every hundred time steps, the frequency of needing Eq. 3.2 is small. It also does not typically have a large impact on momentum, on rarely giving the atom that unrealistic kick. Allowing the atom to go through the same diffusion in every time step was for simplifying and optimizing the program. A comparison was made using the equations as written above to where diffusion is eliminated during a jump, and no noticeable difference was found.

3.3 Recreating the Lattice

The first goal of the simulation was to accurately recreate the average temperature of atoms after they have thermalized in the optical lattice. There are two separate phenomena acting on the atom to create the graph of figure 3.2 that are characteristic of the optical lattice.

The first is related to the linear slope extending into higher potential well depths. Atoms in the lattice lose energy through Sisyphus cooling until the atoms are no longer able to pass the cross-over between wells. Any atom with this little energy will only gain energy if it is pumped to another well. The energy required to pass the cross-over of wells is determined by how deep the potential wells are. Then smaller potentials can cool atoms to lower temperatures before the atoms become trapped. Experimentally, when loading atoms into the optical lattice, shallow wells suffer in loading because of their inability to trap hotter atoms

The second process is what stops atoms from reaching condensate temperatures in a red-detuned optical lattice: Spontaneous emissions. As the wells become more and more shallow and the difference between the two potentials decreases, the opportunity to jump from one potential to the other increases, which is a spontaneous-emission process. Each spontaneous emission gives the atom some random kick of momentum, which adds to the average momentum

of the atom. In fact, as the wells become more and more shallow, the spontaneous emissions tend to dominate the behavior of the atom, which manifests as a sharp increase in the average temperature.

There is actually an instructive intermediate step here for those creating simulations of atoms in optical lattices. If the momentum kick from a spontaneous emission is removed, you can observe a steady decrease in temperature as the potential decreases further and further as in figure 3.2. Similarly in debugging, not allowing the atoms enough time to find an equilibrium temperature in the lattice will also produce a similar graph. Likewise, by removing the force from the potentials and leaving only the random momentum kicks, only the sharp slope near the low potential should be seen, and the atoms will reach some limit temperature as potential increases. Refer to App. B for illustrative examples.

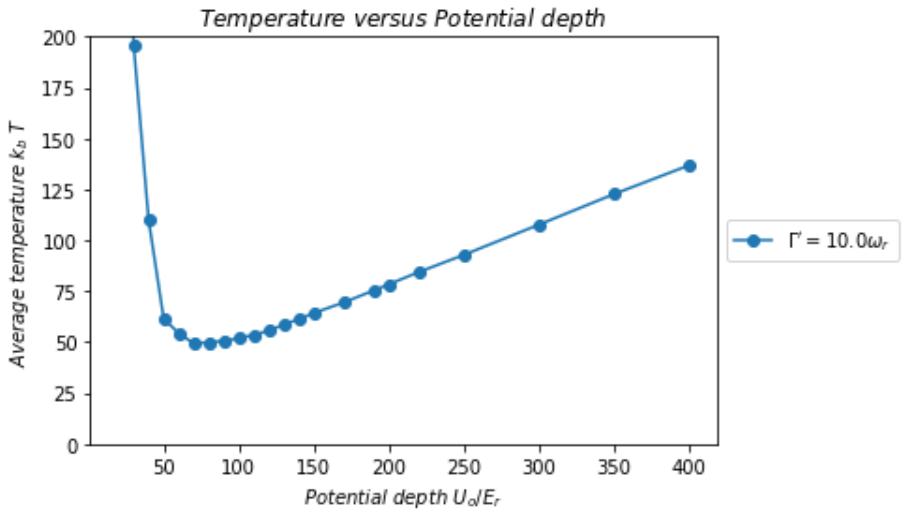

Figure 3.2: Average temperature of 10,000 atoms for different potential well depths. Here the scattering rate scaled by recoil frequency $\Gamma'/\omega_r = 10$ and the atoms had sufficient time to reach equilibrium within the lattice.

3.4 Ratchet simulations

Once the lattice simulation has been established and tested, we can add in a driving force to make the ratchet. Each ratchet is now described and simulation results shown.

3.4.1 Biharmonic Driving

The biharmonic driving ratchet is a form of the rocking ratchet. Rather than try to systematically tilt an optics table with any level of precision, the phase of one of the lattice beams can be shifted so that the wells of the lattice shift back and forth.

$$F_d(t) = F_o\Big(A\cos\omega_d t + B\cos(2\omega_d t + \phi)\Big) \tag{3.8}$$

where F_o is the amplitude of the driving force, t is the time elapsed, ϕ is a chosen phase difference between the two terms, and ω_d is the driving frequency given by

$$\omega_d = 2\omega_r\sqrt{U_o} \tag{3.9}$$

where ω_r is the recoil frequency and U_o is the potential depth in units of recoil energy.

Since we can choose the phase of the driving force, the driving force can be symmetrical or not. Further, since a current-reversal of the ratchet can be caused by any variable, the phase is an excellent choice to demonstrate this phenomenon because of how easy it is to implement. As we will see, other variables do indeed display a current reversal at certain thresholds, but they are often limited in the experiment. For example, laser intensity is not a limitless value, nor is detuning freely selectable without consequence for the equipment. Phase is freely selectable though, as it is selected by a function generator that drives the modulator. Examples of the driving force from Eq. 3.8 for different ϕ are shown in Fig. 3.3.

Biharmonic symmetry analysis

We have already seen that the force from the potential wells is symmetric, anti-symmetric, and shift-symmetric, thus capable of making the system comply with any of our conditions $\hat{S}$ in Sec. 2.2.1 which may forbid net motion. Therefore we look to the driving force to break the symmetry of the system.

The damping on the system does not dominate the motion of the atom because the noise acting on the atom is not negligible, so we cannot assume $m = 0$. Thus we can eliminate $\hat{S}_C$. The next condition to check is if the driving force is shift-symmetric.

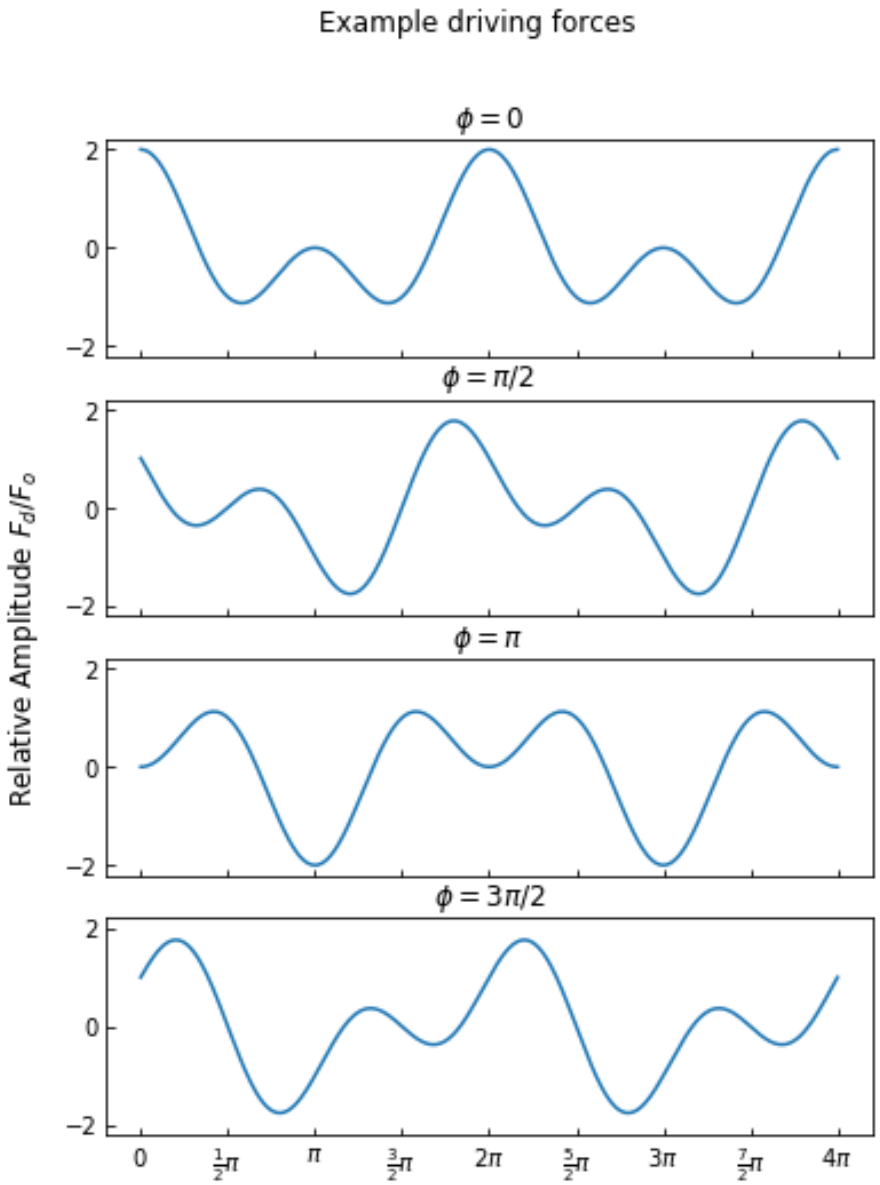

Figure 3.3: This shows what the driving force looks like for four different phase differences. Notice that the driving force can be symmetrical or anti-symmetrical just by a choice of ϕ. Here $A = 1$, $B = 1$, and $\omega_d = 1$.

$$\begin{aligned} F(t+\tfrac{T}{2}) &= F(t+\tfrac{\pi}{\omega_d}) \\ &= F_o\big(A\cos(\omega_d t+\pi)+B\cos(2\omega_d t+2\pi+\phi)\big) \\ &= F_o\big(-A\cos\omega_d t+B\cos(2\omega_d t+\phi)\big) \\ &\neq -F(t) \text{ for any } \phi \end{aligned} \tag{3.10}$$

Since the driving force is not shift-symmetric, condition $\hat{S}_A$ does not apply.

Finally we ask if the driving force is ever symmetric. For $\phi = n\pi$ for any integer n, the driving force is symmetric. Condition $\hat{S}_B$ is satisfied here, but only for those specific values of ϕ. We expect to see a net motion otherwise.

Biharmonic results

A typical result for the biharmonic simulation is shown in Fig. 3.4. Notice first that the average velocities exhibit a current reversal over ϕ as we see that the average velocity flips direction. We might initially expect that the maximum current would be found when the driving force is anti-symmetric and no current when $\phi = n\pi$, but the fact that the cross-over of current for each curve does not occur exactly at π implies this might not be true. In fact, for each curve, that cross-over occurs at a unique place for each. The symmetry analysis of biharmonic driving had forbidden motion, but it critically neglected to consider the scattering the atom would undergo. We can see in Fig. 3.4 that the scattering rate will not only affect the amplitude of the atomic current, but also the phase of the current as a function of ϕ. We will fit all of our data using the function

$$I_{fit}(\phi) = I\sin(\phi-\phi_o) \tag{3.11}$$

where I_{fit} is the resulting curve, I is the amplitude, and ϕ_o is the phase offset of the atomic current. For these simulations and those following later in the chapter, we scale the velocity by the recoil velocity $v_r = 5.9113$ mm/s and the scale the scattering rate by the recoil frequency $\omega_r = 2\pi \cdot 3.7179$ kHz, taken from [13].

Many similar data sets were produced, using a wide range of driving force amplitudes F_o and noise scattering rate Γ'. Each data set was then fitted with the function $I_{fit} = I\sin(\phi-\phi_o)$, where I is the fitted current of the atoms and ϕ_o is the phase of the fit. Work has been done but not presented here on different values for A, B, and U_o, which is more important for experimenters to compare to data. Codes to simulate and graph the ratchets can be found on GitHub following the link in the introduction.

We begin by exploring trends for more efficient atomic current I. We see in Figs. 3.5 and 3.6 that the highest atomic current is found in instances where there is low scattering and high force. However, as force increases we begin to see a leveling off in current in Fig. 3.6. Both Figs. 3.5 and 3.6 show that too small of a force struggles to produce any current, regardless of scattering.

We can also analyze the phase shift ϕ_o of the fitted curves against other parameters as shown in Figs. 3.7 and 3.8. In the case of no dissipation, we expect that for any $\phi = n\pi$ the driving force is symmetrical and will produce

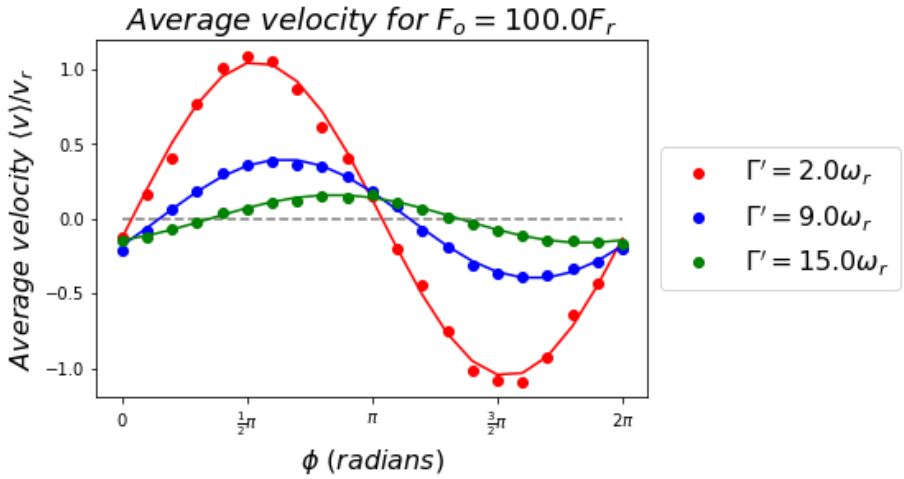

Figure 3.4: Average velocity of 10,000 atoms scanned over phase difference ϕ. The potential depth $U_o/E_r = 200$, driving force $F_o/F_r = 100$, $A = 1$, $B = 1$, and $\omega_d/\omega_v = 1$. Different values of Γ' is listed in the legend. The fit is created with $I_{fit} = T\sin(\phi - \phi_o)$, where I_{fit} is the best-fitting curve. $F_r = \hbar k\omega_r$.

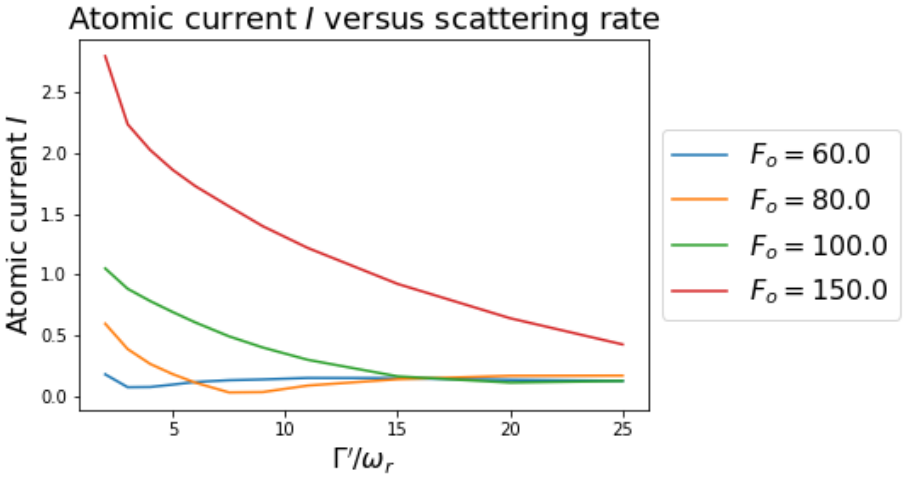

Figure 3.5: Atomic current as compared to Γ'/ω_r. We see much higher currents for low Γ', with a steady decrease for any force at high Γ'.

no net motion. When dissipation is introduced, this behavior changes because the dissipation breaks the symmetry of the bipotential. We would then expect that in cases where the driving force from the ratchet dominates, the value for ϕ_o should be very close to zero as seen in Fig. 3.7. In the opposing case of high

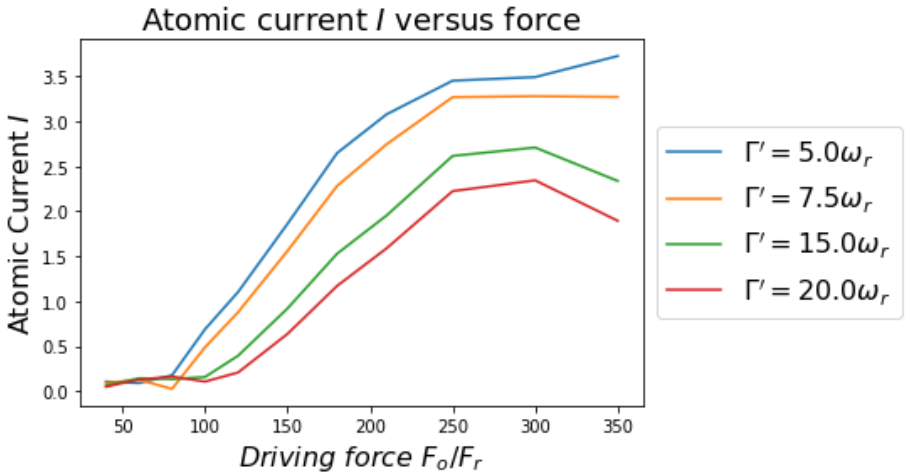

Figure 3.6: Atomic current as compared to the magnitude of the driving force. There is not a continuing increase in current for larger forces, we see the trend begin to reverse for sufficiently high force.

dissipation, whenever the driving force is small or when the scattering rate is large we see a phase away from $\phi_o = 0$, as can be seen in Figs. 3.7 and 3.8. Notice in Fig. 3.7 that for large force and high scattering, both factors mix so that there is more of a transition of the dominant behavior, rather than a sudden change. This is especially apparent in the difference of the transition between $F_o = 100F_r$ and $F_o = 80F_r$, where phase quickly jumps towards π which signifies a current reversal. We also see this current reversal in Fig. 3.8 for all curves when F_o is small.

Understanding different current reversals for variables is a testable milestone between simulation and experiment and provides insight into the efficiency limits of a particular parameter. Remember that a current reversal for non-biased ratchets can occur from the changes of any one parameter. In Figs. 3.9 and 3.10 we demonstrate current reversals as functions of the driving force amplitude and the scattering rate. These have been described over a wide range of values in Figs. 3.7 and 3.8, but these plots exemplify how they appear from a typical data set. In 3.9 we see reversal between $F_o = 100F_r$ and $F_o = 80F_r$, as already noted in Fig. 3.7. This reversal is again seen in 3.10 but as a result of increasing scattering.

Besides current reversals, we can also measure the effectiveness of a ratchet by studying the Peclet number for each simulation. Measuring the Peclet number is useful because it allows us to compare different experiments, rather than being limited to looking at other instances of the biharmonic driving. Note that when calculating the Peclet number, one must choose a representative unit length. The length selected here is the average traveled length of the atoms. This way, the Peclet number gives a description of the rate of diffusion versus

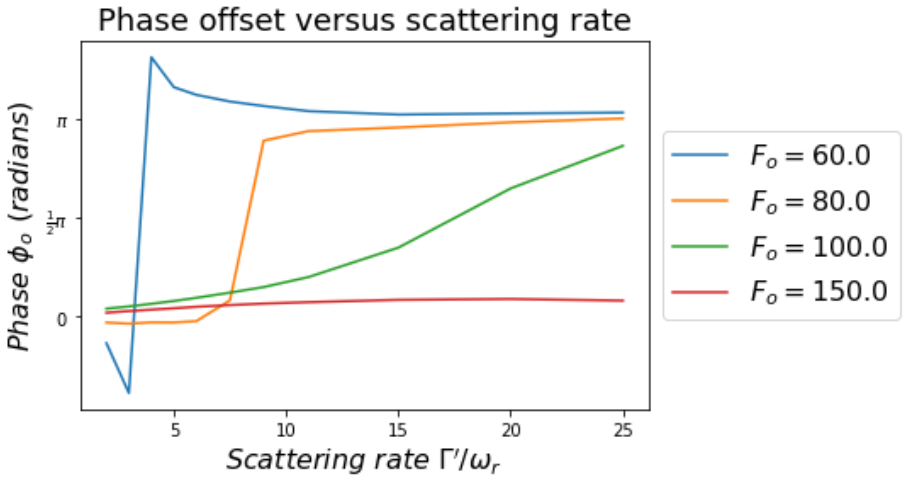

Figure 3.7: Phase offset of the atomic current versus scattering rate. Phase approaching π represents a current reversal of the ratchet. All simulation parameters otherwise follow Fig. 3.4.

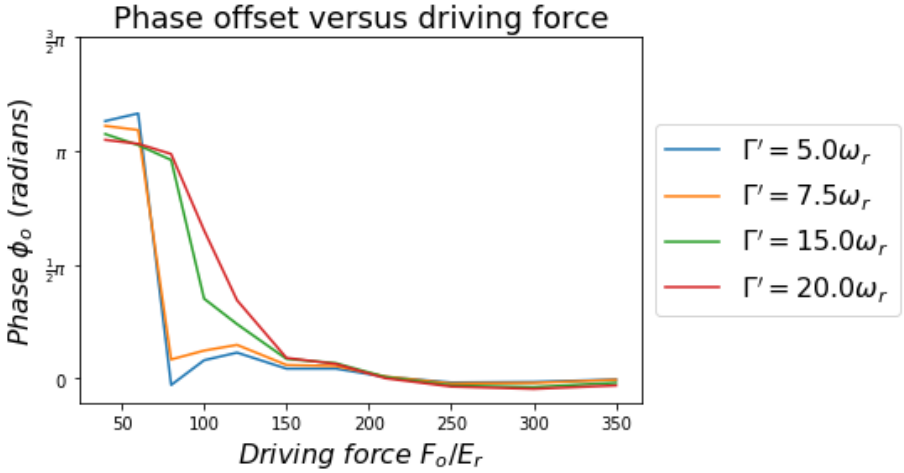

Figure 3.8: Phase offset of the atomic current versus driving force of amplitude F_o. This plot illustrates a clear distinction where the current reversal occurs for different scattering rates. It also shows that for any scattering rate, there is a sufficiently small driving force that will produce a current reversal.

traveling a particular distance. Some preliminary results suggest that the maximum Peclet value seen from a biharmonic driving is 0.05, though issues with small numbers in the simulation cause errors in extracting Peclet numbers for

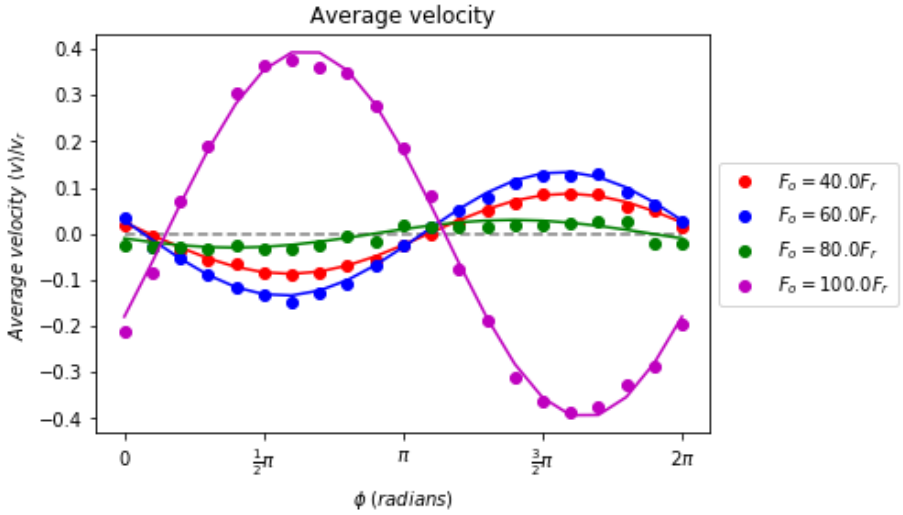

Figure 3.9: Current reversal in the biharomnic ratchet shown as a function of the driving force. Here $\Gamma' = 9\omega_r$.

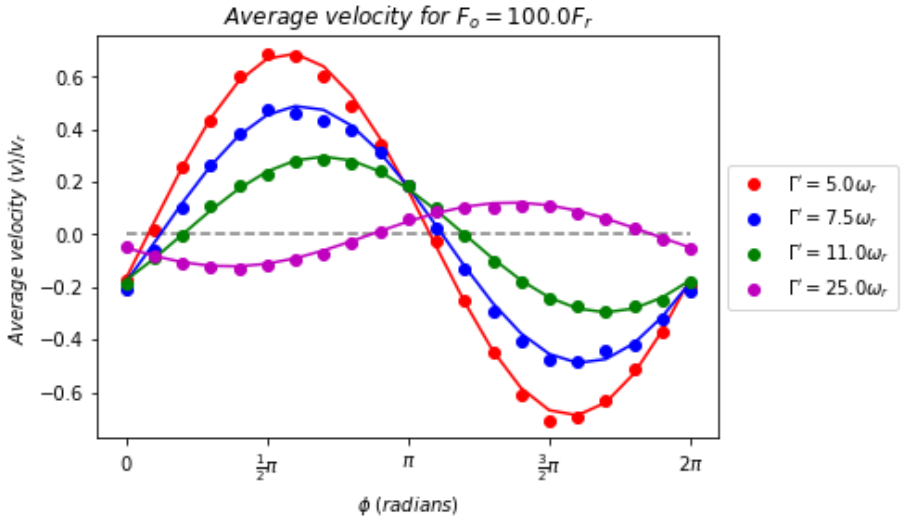

Figure 3.10: Current reversal in the biharmonic ratchet shown as a function of the scattering rate.

certain parameters. This is an ongoing issue within the simulations and is a point for future work to fix.

3.4.2 Multi-Frequency Driving

The multi-frequency driving regime is very similar to the biharmonic albeit more complex. This regime uses two independent frequencies to alter the frequency of one lattice beam.

$$\omega_{shift} = c\sin(\omega_2 t + \delta)\big(a\sin\omega_1 t + b\sin 2\omega_1 t\big) \tag{3.12}$$

This results in an effective force acting on an atom.

$$\begin{aligned} F(t) = -F_o\Big[&\omega_2\cos(\omega_2 t+\phi)\big(A\sin\omega_1 t + B\sin 2\omega_1 t\big) \\ &+ \omega_1\sin(\omega_2 t+\phi)\big(A\cos\omega_1 t + 2B\cos 2\omega_1 t\big)\Big] \end{aligned} \tag{3.13}$$

The ratio between the two driving frequencies is a new parameter to now be explored, which we express as

$$\omega_2/\omega_1 = p/q \tag{3.14}$$

where p and q are positive integers. It is redundant to test equivalent fractions, so p and q will be relatively prime. One example of the driving force is shown in Fig. 3.11. The shape and period of the driving force can vary greatly by choice of p and q. We will explore the conditions that make this force symmetric, anti-symmetric, or shift-symmetric.

Multi-frequency symmetry analysis

Again the potential of the optical lattice can satisfy any of the conditions which forbid net motion. We look exclusively at the driving froce. The damping on the system is still not great enough to approximate that $m = 0$. We can then discard $\hat{S}_C$.

We then start by asking if the multi-frequency force is shift-symmetric. The full period of the multi-frequency driving is defined $T = T_2 p = T_1 q$, where T is the total period, $T_2 = \frac{2\pi}{\omega_2}$ and $T_1 = \frac{2\pi}{\omega_1}$ are the periods relative to the driving frequences ω_2 and ω_1. When the shift transformation is applied we have four terms to compare, ignoring constants.

$$\begin{aligned} \cos(\omega_2 t+\phi)\sin\omega_1 t\cos p\pi\cos nq\pi &= -\cos(\omega_2 t+\phi)\sin\omega_1 t \\ \sin(\omega_2 t+\phi)\cos\omega_1 t\cos p\pi\cos nq\pi &= -\sin(\omega_2 t+\phi)\cos\omega_1 t \end{aligned} \tag{3.15}$$

where $n = 1, 2$. The values of p and q satisfy these equations when p is odd and q is even. Notice in Fig. 3.12 that the values satisfying these conditions are indeed with nearly no net motion.

We should also now check if $\hat{S}_B$ is ever satisfied, which will ask if the driving force is symmetric. Since we have already shown where motion is forbidden, we will only explore where it might still occur, namely when q is odd. Then for the driving force to be symmetric, we require that

$$\begin{aligned} \cos\big((\omega_1+\omega_2)t' + \phi\big) &= 0 \\ \cos\big((\omega_1-\omega_2)t' - \phi\big) &= 0 \end{aligned} \tag{3.16}$$

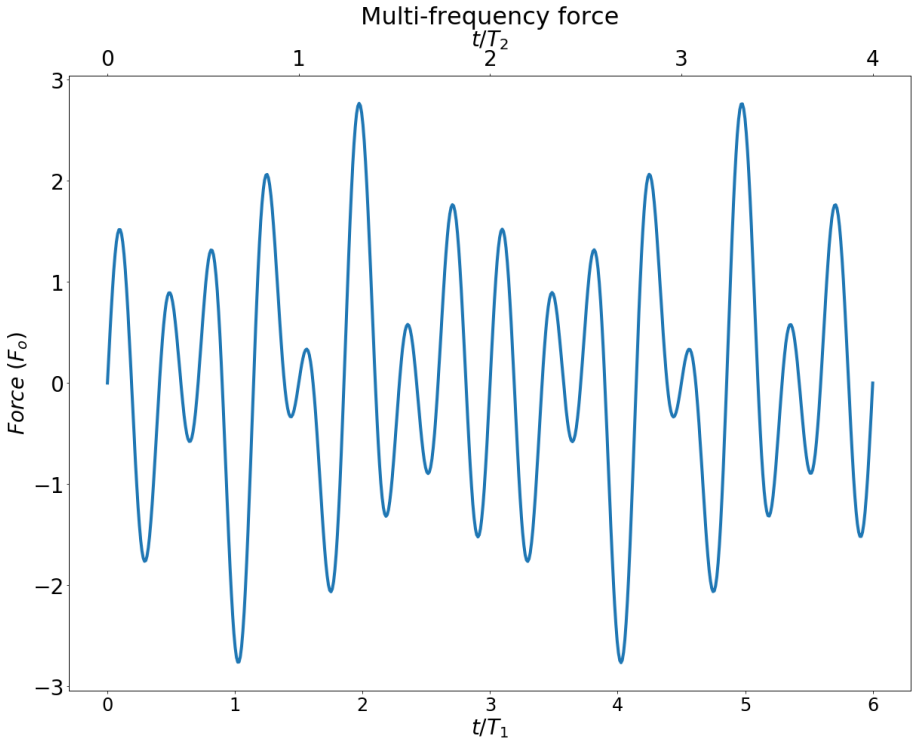

Figure 3.11: Example of a driving force from the multi-frequency regime. In this case $p/q = 2/3$, $A = 1$, $B = 1$, $\phi = 0$, and $T = T_1 q = T_2 p$ are the periods of the force and the two frequencies, respectively.

and we also require

$$\begin{aligned} \cos\left((\omega_1 + \omega_2)t' + \phi + \omega_1 t'\right) &= 0 \\ \cos\left((\omega_1 - \omega_2)t' - \phi + \omega_1 t'\right) &= 0 \end{aligned} \tag{3.17}$$

which is satisfied whenever

$$q\phi = (n + \tfrac{1}{2})\pi \tag{3.18}$$

which defines when the driving force is symmetric. With these two conditions in mind, we can look at the results.

Multi-frequency simulation results

We generate average velocities and fit them in a very similar manner as in the bi-harmonic ratchet. We will instead use the fitting function $I_{fit} = I\sin(q\phi - \phi_o)$.

We may compare more p/q ratios by extracting only the maximum velocity achieved. Notice that of all the values, those with an even q are very nearly

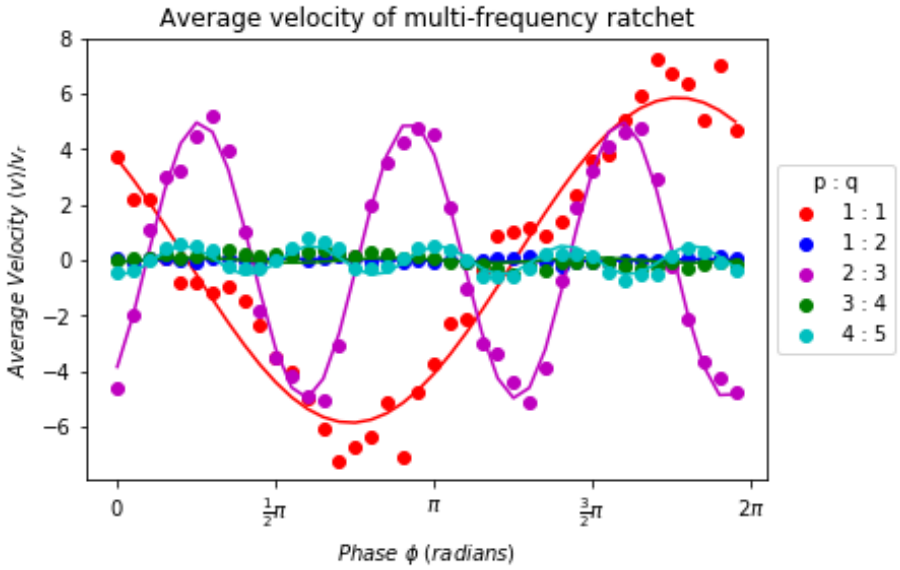

Figure 3.12: Multi-frequency driven ratchet comparing average velocities for different phase constants. The fitted lines are the best fit using a single-frequency sine curve. This data was taken over 10,000 atoms using $U_o/E_r = 200$, $\Gamma'/\omega_r = 10$, $F_o/F_r = 20$, $A = 1$, $B = 1$, and $\omega_1/\omega_v = 0.74$.

zero, where the driving force is symmetric and directed motion should be forbidden. Again, the symmetry analysis we've performed does not account for the scattering present in the simulation, so there will still be some driving, more clearly seen in Fig. 3.13 Finally we study the quasiperiodic limit of the multi-frequency driving. The quasiperiodic limit is the increasing value of pq in the multi-frequnecy regime. In this limit, derived in [2], motion is forbidden regardless of choice of p and q. We study this by comparing the maximum velocity achieved against the product pq, rather than the ratio of the two. As pq gets larger, we approach a quasiperiodic limit of the driving force. The atom cannot mechanically react to the driving force, and we return to no (or little) net motion. Figures 3.15 and 3.16 confirms this, as the maximum velocity continually decreases as pq gets larger.

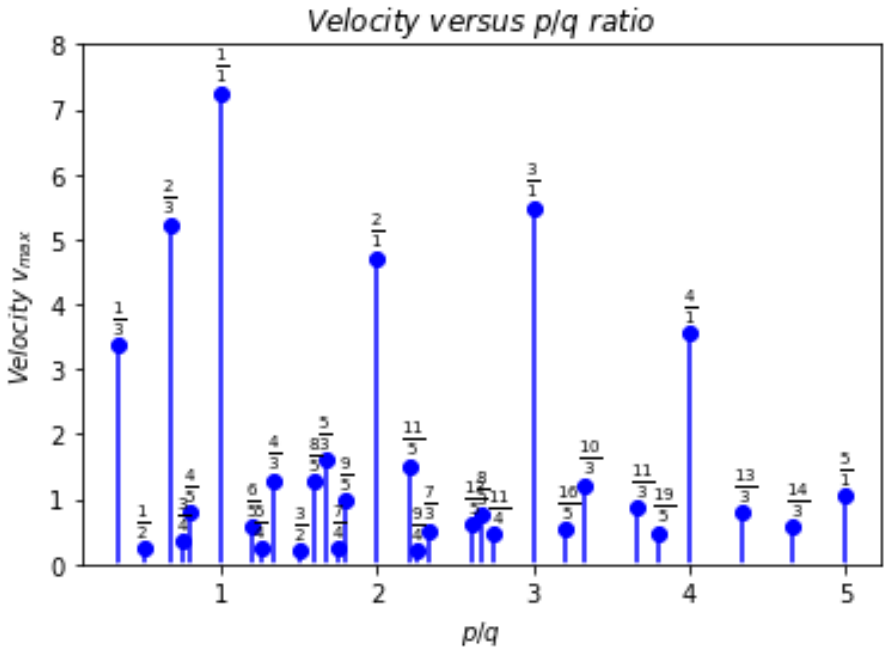

Figure 3.13: Multi-frequency driven ratchets for varying driving ratios ω_1/ω_2. The labels above each point p/q identify the frequency ratio. All other parameters match Fig. 3.12.

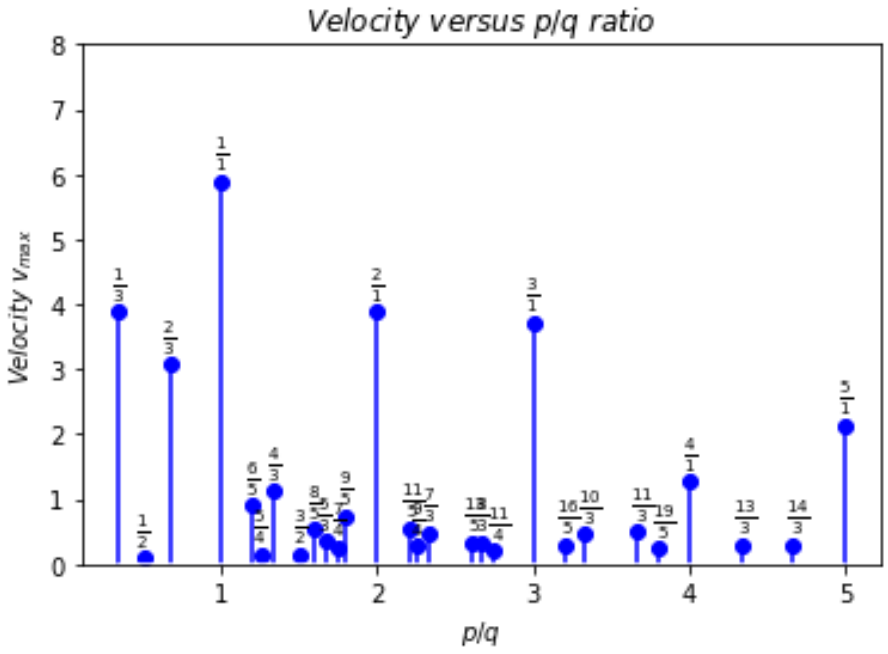

Figure 3.14: Mutli-frequency ratchet, matching all parameters of 3.12 except that $B = 0.3$. The high-velocity ratios are common between the two though the velocities achieved differ.

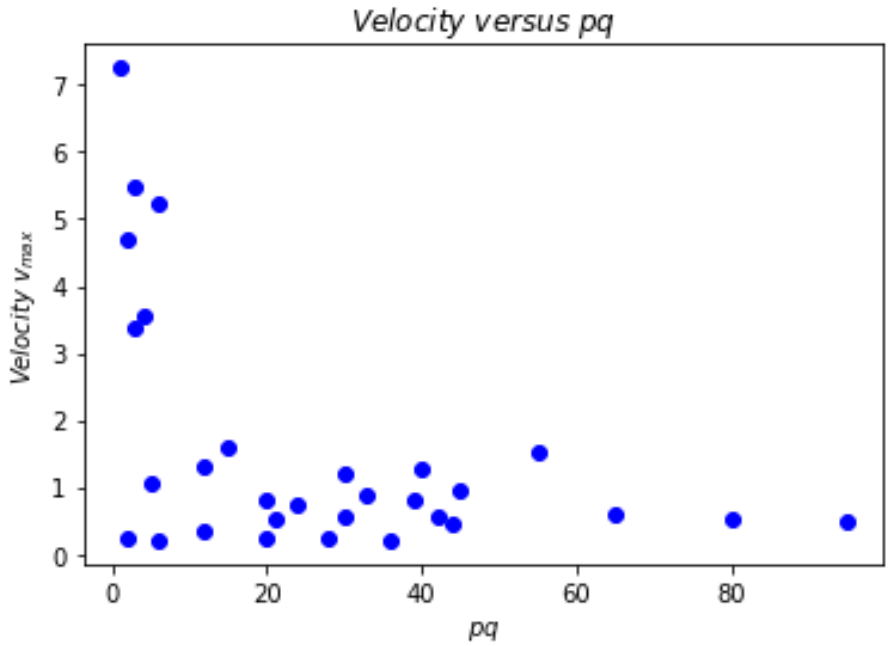

Figure 3.15: This plot shows the maximum velocity achieved by a ratchet versus pq. As pq gets larger, we reach a quasiperiodic limit where net motion disappears even if the conditions would not have forbid it. This plot uses the same parameters as Fig. 3.12.

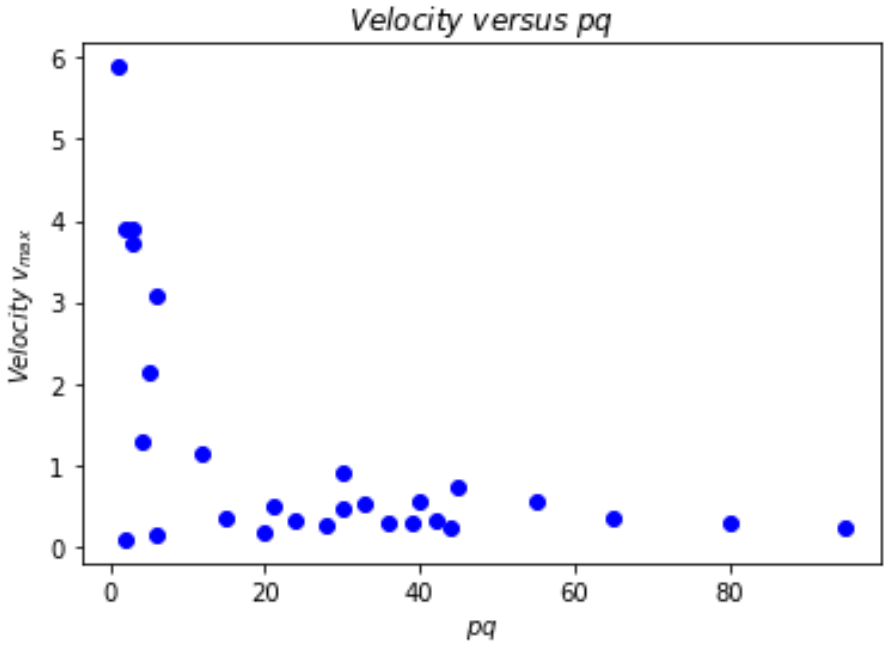

Figure 3.16: Multi-frequency ratchet matching parameters of Fig. 3.12 but with $B = 0.3$. We still see the quasi-periodic limit where maximum velocity vanishes for large pq.

3.4.3 Gating

The gating ratchet, sometimes known as the information ratchet, combines aspects of the flashing and rocking ratchet. There is a single frequency modulating the phase of one of the lattice beams as

$$F(t) = B\cos(\omega_2 t + \phi) \tag{3.19}$$

while a second frequency modulates the intensity of both lattice beams so the potential becomes

$$V(x,t) = V(x)(1 + A\cos\omega_1 t) \tag{3.20}$$

This change in intensity will also affect the scattering rate of the atom. The potential well depth and scattering rate now gain a time dependence in the form

$$U_o(t) = U_o\big[1 + A\cos\omega_1 t\big] \tag{3.21}$$

$$\Gamma'(t) = \Gamma'\big[1 + A\cos\omega_1 t\big] \tag{3.22}$$

The gating ratchet begins by applying a substantial force from one direction. In the ideal case, the majority of the atoms are now moving in the direction of that force. Now the potential begins to drop, removing the primary obstacle from the movement of the atoms. As the force returns in the other direction, the potential rises, preventing back motion. As in the multi-frequency regime, we can study the potential ratios between the frequencies of the occurrences and look for efficient transport.

Gating symmetry analysis

Unlike the previous analyses, the bipotential is no longer unchanging and has gained a time dependence. Then our three conditions which forbid net motion no longer apply and we must determine when motion is forbidden from scratch. We begin again from the motion of equation

$$m\ddot{x} = -\frac{dV(x)}{dx}(1 + A\cos\omega_1 t) + B\cos(\omega_2 t + \phi) - \gamma\dot{x} + \xi(t) \tag{3.23}$$

The condition we look for is any transformation in x or t that changes the sign of p. $\xi(t)$ is an unbiased force, so it does not contribute to our analysis. We begin with a transformation $x \to -x + x'$ and $t \to t + t'$ to get

$$-m\ddot{x} = \frac{dV(-x+x')}{dx}\big(1+A\cos(\omega_1 t + \omega_1 t')\big)+B\cos(\omega_2 t + \omega_2 t' + \phi)-\gamma\dot{x} \tag{3.24}$$

Following the same notation from the multi-frequency regime where $p/q = \omega_2/\omega_1$, net motion will be forbidden when q is even and p is odd. We take these conditions and apply a second transformation $x \to x + x'$ and $t \to -t + t'$. We must also eliminate damping or we will find no other conditions.

$$m\ddot{x} = -\frac{dV(x+x')}{dx}\big(1 + A\cos(\omega_1 t + \omega_1 t')\big) + B\cos(\omega_2 t + \omega_2 t' + \phi)\gamma\dot{x} \tag{3.25}$$

This can be evaluated to find that motion will be also be forbidden for low to no damping when

$$q\phi = n\pi \tag{3.26}$$

for any integer n. Remember that this second condition doesn't apply since we do have dissipation, but will provide estimates where we should find nearly no net motion.

Gating results

We now look at the results of the gating ratchet simulation. We can see from Fig. 3.17 the two cases presented here with even q show almost no net motion. The behavior of the motion as it depends on ϕ again depends p and q.

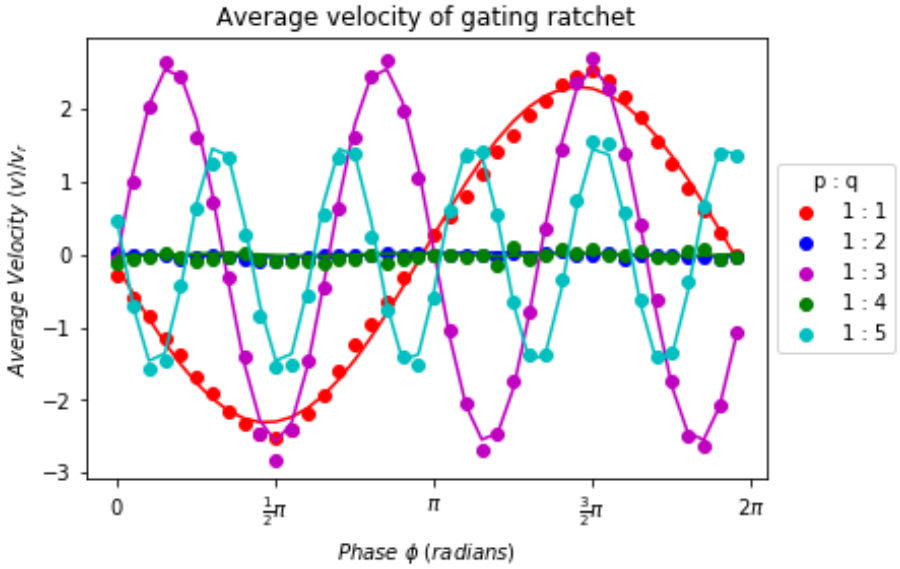

Figure 3.17: Five examples of the gating ratchet average velocity versus ϕ. Notice that when q is even, net motion is nearly zero as we would expect with no dissipation. The periodicity of the velocity is determined by the p value. Here $U_o/E_r = 200$, $\Gamma'/\omega_r = 10$, $\omega_1 = 0.7\omega_v$, $A = .5$, $B = 40$, and 10,000 atoms were simulated.

We can see the maximum velocities achieved for different ratios in Fig. 3.18. Notice again that all cases with even q have a very small maximum velocity.

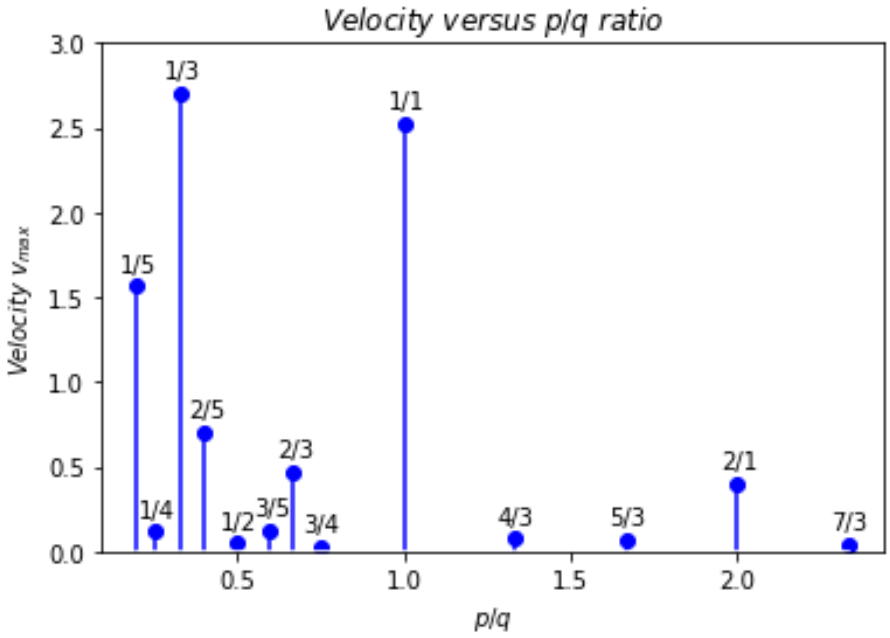

Figure 3.18: Maximum velocity for different ratios p/q in the gating ratchet. This plot shows the results of pulling the highest velocity from a parameter set such as those in Fig. 3.17, with which it shares the same parameters. We see that current is only reasonably generated when q is odd.

3.5 Discussion

My work closely follows the results presented in Martin Brown's 2008 Ph.D thesis. The ongoing goal of this project has been to predict high efficiencies in the parameter space of the ratchet to guide experimental efforts and provide a theoretical framework. Since much of Brown's work had parallel experimental results, recreating the plots presented in his thesis served as a benchmark for the success of our own simulations. Each simulation has its own particular space to explore, and each will be discussed for the current and future work.

3.5.1 Lattice

Recreation of the lattice is the simplest method of proving that the simulation was acting as expected. For achieving the temperature, the relation $k_B T = p^2/m$ was used, which is the kinetic energy to temperature relation in one dimension. Rather than taking the final momentum at the end of the simulation, the momentum was averaged over the entire simulation for each atom. This may skew the data slightly towards the initial conditions, but should be negligible if the atom spends enough time in the lattice.

"Enough time" is still a relative value for the simulation, though experiments typically use a few to tens of milliseconds. In these simulations, giving the atom enough time for on average one hundred absorption-spontaneous emission cycles resulted in a realistic lattice. This still has a dependence on initial conditions though, and more time should be allowed for atoms that begin at higher temperatures. In this simulation, time step and number of steps both must be adjusted to ensure that the time spent in the lattice is appropriate.

In regards to the time step, we have a limit upon the largest step we can take to still be accurate. We should first expect that our time step is smaller than the time to jump between wells.

$$dt << \frac{9}{2\Gamma'} \tag{3.27}$$

Following that, we should also ensure that the dynamics of the atom aren't being truncated with time. The time step should also be smaller than the time to get from a well to a peak, while we assume that an average momentum of the atom is $p \approx 10\hbar k$

$$dt << \frac{\lambda/2}{10\hbar k/M} \tag{3.28}$$

which is not as restrictive as the first time step condition. For this reason, Eq. 3.27 is the determining equation and each simulation's time step is scaled by $1/\Gamma'$. Typical operating time steps chosen are $dt = (0.01 \sim 0.05)/\Gamma'$, though steps as large as 0.1 have been used with no noticeable difference.

The only real limit on how small the time step can be is how patient the researcher is. The total time spent in the lattice still needs to be sufficient to reach an equilibrium, which demands more time steps if the steps are smaller. This can rapidly increase computation time as the time step is decreased.

One last consideration to be made is the choice to eliminate diffusion from changing wells and replace it with a same-well diffusion as in Eq. 3.1. The problem is that the cross-well diffusion term can suddenly kick the atom with very unrealistic momentums. Some groups eliminate these atoms [**?**], others ignore the term [2]. When the simulation was first built, the term was still included. The term did not seem to make any real impact on the average temperatures found. Furthermore, it was reasoned that since the occurrence of a jump was every hundred or so steps ($dt\Gamma' = 0.01$) the impact of using the same-well diffusion in every step would be minimal. This decreases computation time and when tested seems to have no impact on results.

With the parameters settled, the simulation is run and the results compared against the plot in Brown's thesis, to good agreement as seen in Fig. 3.19. The results should be within an order of magnitude of experimental results, but our assumptions of an $F_g = \frac{1}{2} \leftrightarrow F_e = \frac{3}{2}$ atom instead of the real $F_g = 3 \leftrightarrow F_e = 4$ and our restriction to one dimension do have impact on the final values.

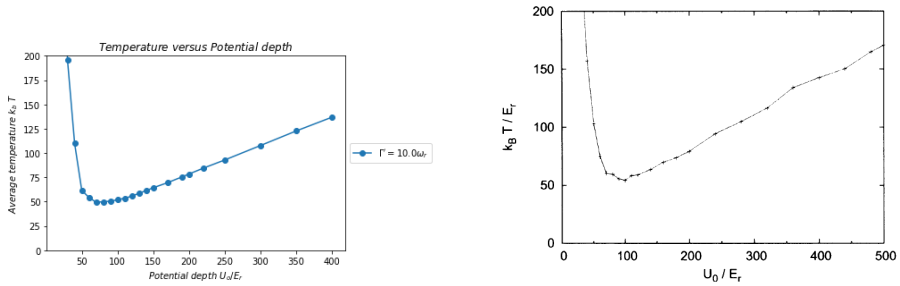

Figure 3.19: A comparison of the lattice temperatures from our simulation (left) versus the simulation presented by Brown [2] (right).

3.5.2 Biharmonic

The biharmonic ratchet simulation is derived from the lattice simulation. All relevant details were still in effect, and the driving force was introduced as an extra term in the change of momentum for each time step. The primary difference between the simulations lie in what data is being collected. Since we are looking for the change in position of each atom, we were no longer interested in finding $\langle p^2 \rangle$, but instead compared the initial and final positions of the atoms. This also meant that using the intermediate steps of the simulation was no longer possible, so more atoms had to be sampled relative to the lattice simulation to achieve accurate results. The fact that both simulations used the same number of atoms was an unnecessary increase for the lattice. Comparisons are being made to the results Brown published in [14].

Since we don't have experiment on the ratchets running yet, the comparison used for confidence in the simulation is to Brown's results and the experimental work his group has performed. We can begin by comparing the velocity versus phase ϕ. Fig. 4.4 in Brown's thesis shows an example plot, but a more comprehensive comparison can be made by comparing current amplitudes and phases as in Fig. 3.20. In both simulations, we see the characteristic feature of a ratchet that a current reversal can exist as a function of any parameter of the system. However, efficiency of the transport may suffer by changing these variables.

A clearer comparison of matching the current reversals is comparing the fitted phase of the directed motion. Comparing our simulation results to Brown we find mostly good agreement, where the edges of the parameter space tested match, while there is a very slight discrepancy around $F_o = 80F_r$. The fitted phase is not quite matching. Around $F_o = 80F_r$ is a current reversal for different values of Γ'. In our simulation, the current reversal is very sudden, whereas for slightly higher F_o the change is more gradual. This gradual shift occurs in Brown's work right at $F_o = 80F_r$. No adjustments to integration methods or size of time steps seem to change this difference. Overall, this difference is small where most other results agree. The last thing to compare is the maximum

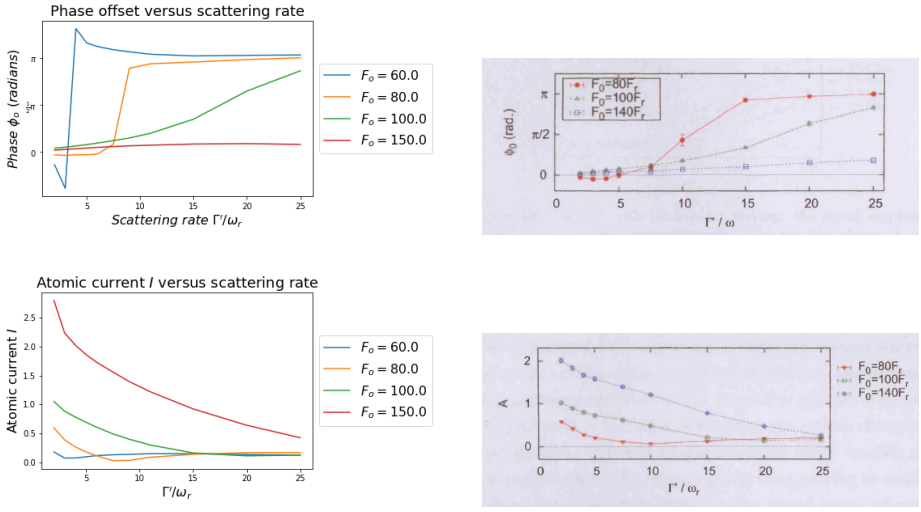

Figure 3.20: A side-by-side comparison of the biharmonic ratchet simulations presented here (left) and Brown's results [2] (right). Both the current amplitude and current phase are shown.

velocities achieved for different sets of parameters. All of these values have been in very good agreement despite that small difference found in the phase.

3.5.3 Multi-Frequency

The multi-frequency ratchet was also derived directly from the lattice simulation, recycling much of the code from the biharmonic driving. One large difference in how the simulation was run is the number of ϕ values was doubled because the resulting shape of the data is very complex. Even in the simpler p/q ratios you can see that a single sine wave doesn't accurately fit to the data, but a consistently fitting function isn't possible for the more complex ratios. For this reason the analysis plots over all data use the maximum velocity from a data set over the amplitude of a fitted curve. A consequence of this is that the values plotted in the analysis graphs (Figs. 3.15 and 3.17) could be higher than expected due to outliers.

The results are compared again to Brown in [15]. There are two different data sets here, one where $A = 1$ and $B = 1$, and a second $A = 1$ and $B = 0.3$. Both have been presented here, but comparisons will only be made to the first set in Fig. 3.21. While we do see the same p/q ratios having a non-zero current, those currents are not quite matching. Besides the overestimation of current, we still see a few ratios giving much higher currents than was expected. A better fitting program is necessary here, as Fig. 3.12 hints that a single sine wave is not a very accurate fit. If a more accurate fitting was used instead of a maximum velocity, this difference would not be as extreme.

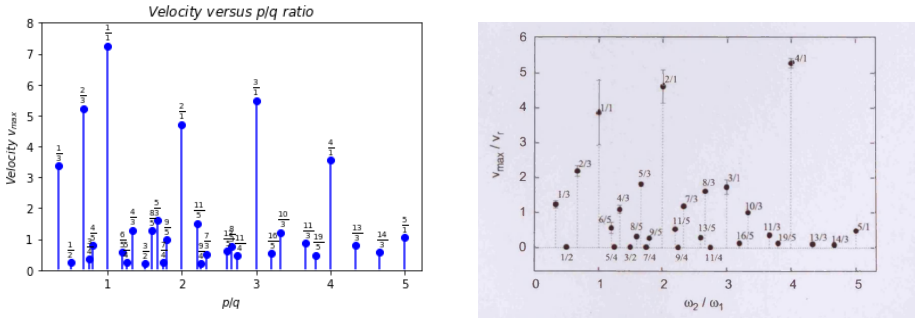

Figure 3.21: A comparison of our multi-frequency driving (left) versus the results from Brown [2] (right).

3.5.4 Gating

The gating ratchet was studied in much the same way that the multi-frequency ratchet was. Both had complex trends of the atom current relative to the biharmonic driving. We see a relatively good agreement with the results presented in Brown's thesis and the relating papers [16] in Fig. 3.22. Indeed the effective ratios show similar maximum velocities, though our simulations appear to state a higher maximum velocity. One possible reason is that I consider the maximum velocity achieved, rather than fitting a curve and using the amplitude as Brown had done [16]. There could also be small differences in the simulation itself, such as the time step and integration method.

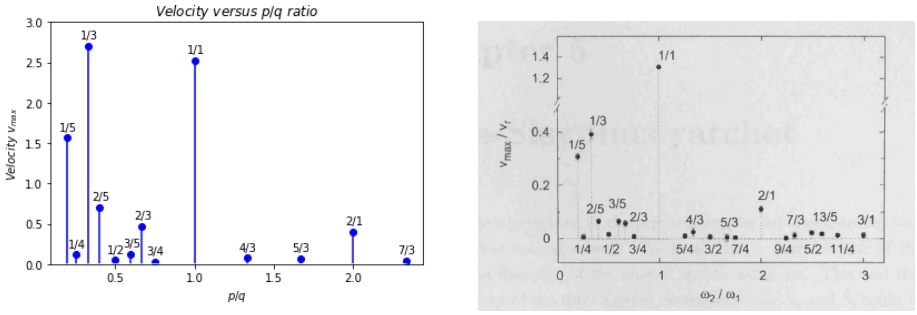

Figure 3.22: Side-by-side comparison of the gating ratchet results presented in this thesis (left) versus Brown [2] (right).

Chapter 4

Conclusion

This thesis began as a motivation for why to study Brownian ratchets, specifically made from cold atoms in optical lattices. The simulations presented in this thesis act as a starting point for the group's effort towards experimentally creating these ratchets, where the simulations serve first as a guide to relevant results, and end as a theoretical understanding of the dynamics inside the lattice. For this reason, the simulations were designed for flexibility so that any experimental values can be easily measured in the simulation for early estimates.

Our simulations were confined to the case of a $F_g = 1/2 \leftrightarrow F_e = 3/2$ atom in a one-dimensional optical lattice. The first simulation made was only to describe the optical lattice with no ratchet. Results were agreeable with the expectations within a lattice, and these results can be easily compared to any temperature-measuring method for optical lattices. Once results from this simulation were found to match the established simulations, the ratcheting force is introduced into the simulation just as was described in Chp. 2. One benefit of a semi-classical model is that forces acting on the atom can be simply added as extra terms to the change of momentum in a time step. For different regimes of the rocking ratchet (e.g. biharmonic and multi-frequency) no change needs to be made to the main body of the simulation. The ratcheting force was adjusted for each, and with minimal changes to account for regime-specific variables, the simulation was ready.

The biharmonic driving was the first tested. We first showed a symmetry analysis for a simplified scenario where there was no dissipation. This describes a general trend for the ratchet dynamics before the simulations are run. We expect the results of the simulation to differ slightly though because dissipation cannot be neglected. The results were then compared to results from the Renzoni group [2, 14, 17] and found to be in good agreement, save for small differences in where current reversals occur. The transport rates match closely, giving confidence to the accuracy of the simulations created here. The parameter space was explored a bit more than the results presented in Brown's thesis, but ultimately the characteristics of the ratchet are unchanging, and further exploration of the parameter space doesn't reveal new physics.

The multi-frequency and gating ratchets were compared and presented in much the same way. Results were still similar to the established results [2, 15], and there is confidence that these simulations were accurate as well. Current reversal here was not as well studied, as it is more complicated in these regimes and we already have shown through the biharmonic driving that the ratchets will flip transport as a function of any parameter. The parameter space is more rich here, as there is a second frequency to adjust that wasn't present in the biharmonic driving. The ratio between these two characteristic ratchet frequencies ω_2/ω_1 became a point of focus. The estimated motion-forbidden conditions were indeed shown to have nearly no net motion, with the expectation that dissipation introduces asymmetry. The gating ratchet was studied in a very similar manner to the multi-frequency, with the focus again on ω_2/ω_1. The results proved agreeable with [2, 16].

Future Work

We've seen that for any of these ratchets, there's a multitude of variables to work with. There is a complex interaction between multiple terms. The work presented is intended to display general trends in ratchet behavior, not to match experimental results. Further progress requires experiments on cold atom ratchets to be initiated. Our simulation results have matched [2] but of course the ultimate goal is to perform ratchet experiments and model experimental observations. Preliminary results for a cold atom ratchet have been made in [18], though these results are concerned with a Brillouin ratchet, and doesn't follow any regime described in these simulations.

The comparison of experiment with simulation may also help determine the limitations that this particular simulation has had. We have made a number of assumptions that could all be improved upon in later iteration of the code. First, the structure of the lattice is limited to one dimension. While there are experimental one-dimensional lattices, most cutting edge experiments at present are being conducted in three-dimensional optical lattices. The simulation could be expanded to a three-dimensional lattice structure, whether tetrahedral or inverted-umbrella [19], and other ratcheting regimes [20] may also be interesting to investigate. It is important to go beyond the $F_g = 1/2 \leftrightarrow F_e = 3/2$ model atom towards a more realistic $F_g = 3 \leftrightarrow F_e = 4$ rubidium atom or $F_g = 4 \leftrightarrow F_e = 5$ for cesium. Finally, it would be of interest to make the simulation fully quantum so as to investigate possible quantum walks in optical lattices.

Besides modeling just position and momentum of the atom, we'd like to extend our simulations to predict what we might observe using frequently employed methods of observation such as pump-probe spectroscopy and photon correlation measurements.

Appendices

Appendix A

Semi-classical derivations

A.1 Roadmap

When the ratchets were first described, we took a very classical picture with them. The atoms didn't have different states, there was a single periodic potential, and we certainly weren't very concerned with the temperature of those atoms. Reality is not so simple though. Truthfully the atom has multiple internal states (ground and excited), there is not just a single potential pattern, and the temperature and velocity of the atom changes over time. We still need to observe the motion that our simulated atom goes through.

When we describe this motion there are three forces acting on the atom. The first is the force from the potential the atom experiences from the light field. The second is a random force from each spontaneous emission event. After every absorption, the atom may spontaneously emit in a random direction, generating Brownian noise. The third is the driving force which may be introduced to create a ratchet.

The first force arises from the gradient in the spatially-varying light shifts of the atom confined in the optical lattice. The light shift depends on the intesnity of the light field, the light polarization, and the strength of coupling of the light with the internal state of the atom. A $F_g = \frac{1}{2} \leftrightarrow F_e = \frac{3}{2}$ atom is the simplest system that captures the general behavior of the experimental atom moving in the optical lattice.

The second force, from photon scattering, is also dependent on light field intensity and the internal states of the atom. Every time the atom absorbs a photon from the field, there are two options for emission: spontaneous or stimulated. In the case of a stimulated emission, the photon is re-emitted along the same path as it was absorbed, causing no net change to the momentum of the atom. In the spontaneous case though, a photon can be re-emitted in any direction. We need to calculate the probability of an absorption, followed by spontaneous emission, which depends on the atom's location in the field as well as the state that the atom is in. At that point we revisit the populations in

states and polarization gradient that we will have found for the potential. We will use the Fokker-Planck equation to determine atomic diffusion in space and momentum.

A.2 Optical Bloch Equations

We will follow a very similar approach to Cohen-Tannoudji [8] for the inital steps starting from the Bloch equations. We'll follow Brown [2] when the Wigner transform is introduced, though a more thorough explanation of those steps can be found in [11]

A.2.1 Field configuration for a 1D Lattice

The structure of the lattice is determined by the orientation and number of laser beams. A three-dimensional lattice, such as the one in our experiment, requires a minimum of four beams which create a structure of wells like a crystal. However, this is not only difficult to visualize, it's also much more complicated to implement in a simulation. Therefore, we choose to model a one-dimensional lattice as shown in Fig. A.1. The one-dimensional lattice we use is a lin⊥lin configuration but a lattice could also be made consisting of circularly-polarized beams as explored in [21]

The one-dimensional optical lattice is created by the overlap of two counter-propagating beams with orthogonal linear polarizations, known as the lin⊥lin configuration. Both beams are of equal intensity E_o and frequency ω, which is detuned from the transition of the atom by $\Delta \equiv \omega_o - \omega$. The two beams also have a phase difference ϕ between them. The resulting electric field along the z-axis is then:

$$\begin{aligned}\vec{E}(z,t) = E_o\hat{\epsilon}_x e^{i(kz-\omega_L t)} + E_o\hat{\epsilon}_y e^{i(-kz-\omega_L t)}e^{i\phi} + c.c. \\ = \epsilon^+(z)e^{-i\omega t} + \epsilon^-(z)e^{i\omega t}\end{aligned} \tag{A.1}$$

where $k = 2\pi/\lambda$ is the wave number and ϵ represents polarization. Because of the presence of the Zeeman substates, we choose to express the electric field in terms of left- and right-circularly polarized beams instead of orthogonal linear polarizations. Following [8] we define the circular polarizations as:

$$\begin{aligned}\hat{\epsilon}_+ &= \frac{-(\hat{e}_x + i\hat{e}_y)}{\sqrt{2}} \\ \hat{\epsilon}_- &= \frac{\hat{e}_x - i\hat{e}_y}{\sqrt{2}}\end{aligned} \tag{A.2}$$

$$\vec{E}(z,t) = \frac{E_o}{\sqrt{2}}\Big[\hat{e}_+(A_{++}e^{i\omega t} + A_{+-}e^{-i\omega t}) + \hat{e}_-(A_{-+}e^{i\omega t} + A_{--}e^{-i\omega t}\Big] \tag{A.3}$$

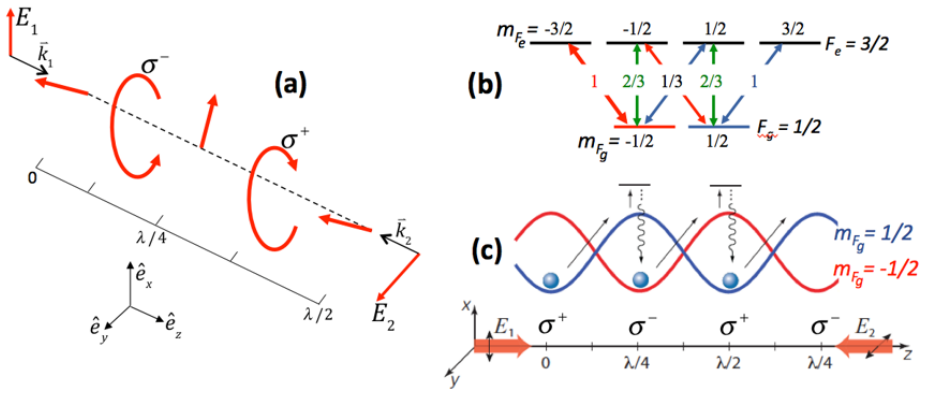

Figure A.1: (a) Shows the resulting polarization gradient in the lin⊥lin one-dimensional lattice. (b) This is the structure of the $F_g = \frac{1}{2} \leftrightarrow F_e = \frac{3}{2}$ atom with Clebsch-Gordon coefficients depicted on each transition. (c) The two ground states of the atom experience different light-shifts dependent on polarization.

where we have described these terms A as

$$\begin{aligned} A_{++} &= 2i \sin kz = -e^{-ikz} + ie^{ikz}e^{-i\phi} \\ A_{+-} &= -2 \cos kz = -e^{ikz} + ie^{-ikz}e^{i\phi} \\ A_{-+} &= 2 \cos kz = e^{-ikz} + ie^{ikz}e^{-i\phi} \\ A_{--} &= 2i \sin kz = e^{ikz} + ie^{-ikz}e^{i\phi} \end{aligned} \tag{A.4}$$

Note that here we have made a choice of the phase offset between the two lattice beams. Any choice of the phase offset will ultimately describe the same physics, but we have chosen $ie^{-i\phi} = 1$ to simplify the terms as much as possible. Other possible lattice structures are described in detail inf [19].

A.2.2 $F_g = \frac{1}{2} \leftrightarrow F_e = \frac{3}{2}$ Atom in 1D Lattice

The model atom $F_g = \frac{1}{2} \leftrightarrow F_e = \frac{3}{2}$ is depicted in Fig. **??**, along with the Clebsch-Gordon coefficients for each dipole-allowed transition.

Now we will follow the figure in labeling the states 1 through 6, where 1 and 2 are ground states $F_g = -1/2$ and $F_g = +1/2$ respectively, and 3 through 6 are excited. We can represent the wave function of the atom as a combination of all six of these states.

$$\psi(\vec{x}, t) = \sum a_n(t) \left|\phi_n(\vec{x})\right\rangle \tag{A.5}$$

Here $|\phi_n\rangle$ represents each internal state of the atom and a_n describes the probability amplitude for that state. We'll use the Schrodinger equation to solve for each a_n. The Hamiltonian used here consists of two terms; first the term

$\hat{H}_o$ which describes the energy of the state and second V is the energy of the induced dipole in the electric field of the optical lattice.

$$i\hbar\frac{\partial\left|\psi\right\rangle}{\partial t} = \hat{H}_o\left|\psi\right\rangle + \hat{V}\left|\psi\right\rangle \tag{A.6}$$

Then the Schrodinger equation becomes

$$i\hbar\sum_n \dot{a}_n\left|\phi_n\right\rangle = \sum_n E_n a_n\left|\phi_n\right\rangle + \sum_n V a_n\left|\phi_n\right\rangle \tag{A.7}$$

Where E_n is the bare energy of the n-th state. Now we will start to isolate a single term.

$$\left\langle\phi_m\right| i\hbar\sum_n \dot{a}_n\left|\phi_n\right\rangle = \left\langle\phi_m\right|\sum_n E_n a_n\left|\phi_n\right\rangle + \left\langle\phi_m\right|\sum_n V a_n\left|\phi_n\right\rangle \tag{A.8}$$

The internal states of the atom are orthogonal. We also define $V_{mn} = \left\langle\phi_m\right| V\left|\phi_n\right\rangle$ is the induced dipole interaction between $\left\langle\phi_m\right|$ and $\left|phi_n\right\rangle$.

$$i\hbar\dot{a}_m = E_m a_m + \sum_n V_{mn} a_n \tag{A.9}$$

So for one such state we would find

$$i\hbar\dot{a}_1 = E_1 a_1 + V_{11}a_1 + V_{12}a_2 + V_{13}a_3 + V_{14}a_4 + V_{15}a_5 + V_{16}a_6 \tag{A.10}$$

From this equation many terms can be eliminated. V_{11} is eliminated because there is no dipole interaction between a state and itself. V_{12} and V_{16} are both eliminated because they are forbidden transitions. Lastly in this particular setup V_{14} is eliminated because it is not a driven transition. In other lattice configurations, this term might survive otherwise.

In a similar fashion, we derive an equation for each term as

$$\begin{aligned}
i\hbar\dot{a}_1 &= E_1 a_1 + a_3 V_{13} + a_5 V_{15} \\
i\hbar\dot{a}_2 &= E_2 a_2 + a_4 V_{24} + a_6 V_{26} \\
i\hbar\dot{a}_3 &= E_3 a_3 + a_1 V_{31} \\
i\hbar\dot{a}_4 &= E_4 a_4 + a_2 V_{42} \\
i\hbar\dot{a}_5 &= E_5 a_5 + a_1 V_{51} \\
i\hbar\dot{a}_6 &= E_6 a_6 + a_2 V_{62}
\end{aligned} \tag{A.11}$$

This simulation assumes that there is no magnetic field present in the optical lattice, and stray magnetic fields in our experiment have been reduced to less than 10 Milligauss. We therefore assume that the energies of the both ground states to be zero and the energies of the excited Zeeman sub-states are $E_i = \hbar\omega_o$,

where $i = 3, 4, 5, 6$.

$$\begin{aligned}
\dot{a}_1 &= -\frac{i}{\hbar}(a_3 V_{13} + a_5 V_{15}) \\
\dot{a}_2 &= -\frac{i}{\hbar}(a_4 V_{24} + a_6 V_{26}) \\
\dot{a}_3 &= -i\omega_o a_3 - \frac{i}{\hbar} a_3 V_{31} \\
\dot{a}_4 &= -i\omega_o a_4 - \frac{i}{\hbar} a_4 V_{42} \\
\dot{a}_5 &= -i\omega_o a_5 - \frac{i}{\hbar} a_5 V_{51} \\
\dot{a}_6 &= -i\omega_o a_6 - \frac{i}{\hbar} a_6 V_{62}
\end{aligned} \tag{A.12}$$

A.2.3 Rotating-Wave Approximation and the $\dot{c}$-Equations

We want to separate less interesting phenomena occurring at the optical frequency from more interesting phenomena that occurs at slower frequencies. We replace the $\dot{a}$-equations from Eq. A.12 with

$$\begin{aligned}
a_i &= c_i e^{-i\omega t}, \quad i = 3, 4, 5, 6 \\
a_i &= c_i, \qquad\qquad\quad i = 1, 2
\end{aligned} \tag{A.13}$$

Where we have only replaced the terms corresponding with excited states. Now we solve for the $\dot{c}$-equations.

$$\begin{aligned}
\dot{c}_1 &= -\frac{i}{\hbar}(V_{13} c_3 e^{-i\omega t} + V_{15} c_5 e^{-i\omega t}) \\
\dot{c}_2 &= -\frac{i}{\hbar}(V_{24} c_4 e^{-i\omega t} + V_{26} c_6 e^{-i\omega t}) \\
\dot{c}_3 &= -i\Delta c_3 - \frac{i}{\hbar} V_{31} c_1 e^{i\omega t} \\
\dot{c}_4 &= -i\Delta c_4 - \frac{i}{\hbar} V_{42} c_2 e^{i\omega t} \\
\dot{c}_5 &= -i\Delta c_5 - \frac{i}{\hbar} V_{51} c_1 e^{i\omega t} \\
\dot{c}_6 &= -i\Delta c_6 - \frac{i}{\hbar} V_{62} c_2 e^{i\omega t}
\end{aligned} \tag{A.14}$$

A.2.4 Dipole Interaction Terms

The dipole interaction considers the coupling of the electric field with the induced dipole of the atom. Up until now, the dipole interaction terms have been very complicated, but the introduction of the $\dot{c}$-equations is an ideal time to evaluate these terms. Let's revisit the definition and expand upon it.

$$V_{mn} = \langle \phi_m | \, V \, | \phi_n \rangle = (-e\vec{r}_{mn}) \cdot \big(\vec{E}(z)\big) \tag{A.15}$$

We have given $(-e\vec{r}_{mn})$ as the induced dipole moment. But the dipole interaction is not an isolated term in the $\dot{c}$-equations like it previously was. We'll consider the terms along with the exponential terms they've picked up. We can work with a single term as an example.

$$V_{24}e^{-i\omega t} = \frac{E_o}{\sqrt{2}}(-e\vec{r}_{24}) \cdot \begin{bmatrix} (\hat{e}_+)(A_{++}e^{i\omega t} + A_{+-}e^{-i\omega t}) \\ +(\hat{e}_-)(A_{-+}e^{i\omega t} + A_{--}e^{-i\omega t}) \end{bmatrix} e^{-i\omega t} \tag{A.16}$$

We are following the notation for the electric field from Eq. **??** for simplicity. The $|2\rangle \leftrightarrow |4\rangle$ transition is a σ^- transition, so the σ^+ terms are eliminated as $\vec{r}_{24} \cdot (\hat{e}_+) = 0$. Then the exponential is pulled into the brackets as

$$V_{24}e^{-i\omega t} = \frac{E_o}{\sqrt{2}}(e\vec{r}_{24}) \cdot \left[(\hat{e}_-)(A_{-+} + A_{--}e^{-2i\omega t}\right] \tag{A.17}$$

Now we have two terms, where one is quickly oscillating. The Rotating-Wave Approximation (RWA) assumes that the atom can't mechanically react to such a quick oscillation, and so that term can be neglected. After replacing A_{-+} we are left with a far simpler dipole term

$$V_{24}e^{-i\omega t} = \frac{\hbar\Omega}{\sqrt{6}}\cos kz \tag{A.18}$$

where Ω represents the Rabi frequency given by

$$\Omega \equiv \frac{-2erE_o}{\hbar} \tag{A.19}$$

Performing similar derivations gives results for some of the other dipole interaction terms.

$$\begin{aligned} V_{13}e^{-i\omega t} &= \frac{\hbar\Omega}{\sqrt{2}}cos(kz) \quad & V_{24}e^{-i\omega t} &= \frac{\hbar\Omega}{\sqrt{6}}sin(kz) \\ V_{13}e^{i\omega t} &= \frac{\hbar\Omega}{\sqrt{2}}sin(kz) & V_{24}e^{i\omega t} &= \frac{\hbar\Omega}{\sqrt{6}}cos(kz) \\ V_{15}e^{-i\omega t} &= \frac{\hbar\Omega}{\sqrt{6}}sin(kz) & V_{26}e^{-i\omega t} &= \frac{\hbar\Omega}{\sqrt{2}}cos(kz) \\ V_{15}e^{i\omega t} &= \frac{\hbar\Omega}{\sqrt{6}}cos(kz) & V_{26}e^{i\omega t} &= \frac{\hbar\Omega}{\sqrt{2}}sin(kz) \end{aligned} \tag{A.20}$$

For a single transition, there are eight possible unique corresponding terms, not all included here. We will only need four such terms, but that is still rather complex given the four driven transitions. Recognizing that we will need the interaction terms in not only the $\dot{c}$-equations but also their complex conjugates, let's solve a second example.

$$\begin{aligned} V^*_{42}e^{i\omega t} &= \frac{E_o}{\sqrt{2}}(-e\vec{r}_{42}) \cdot \begin{bmatrix} (\hat{e}_+)(A^*_{++}e^{-i\omega t} + A^*_{+-}e^{i\omega t}) \\ +(\hat{e}_-)(A^*_{-+}e^{-i\omega t} + A^*_{--}e^{i\omega t}) \end{bmatrix} e^{i\omega t} \\ &= \frac{\hbar\Omega}{\sqrt{6}}\cos kz \end{aligned} \tag{A.21}$$

We have gone through the same process of elimination and using the RWA to find the same result as our first interaction example. We may also notice here that our final terms are real, so

$$(V_{mn}^* e^{\pm i\omega t})^* = V_{mn} e^{\mp i\omega t} \tag{A.22}$$

Each transition follows a similar path to our two examples, so that we only will need to describe two terms for each transition knowing that we can use Eq. A.22 and the following equation for any needed term.

$$V_{mn}^* e^{\pm i\omega t} = V_{nm} e^{\mp i\omega t} \tag{A.23}$$

A.2.5 Coherences and Populations

The density matrix is an object that can describe how the populations and coherences of the internal states of the atom change with time. Specifically, we will use it to derive probabilities of the atom being in a particular state and the likelihood of changing to a different state. We can define each element of the density matrix as

$$\rho_{mn} = c_m^* c_n \tag{A.24}$$

We now recast the $\dot{c}$-equations to describe the time-evolution of the density matrix.

$$\dot{\rho}_{mn} = \frac{d}{dt}(c_m^* c_n) = \dot{c}_m^* c_n + c_m^* \dot{c}_n \tag{A.25}$$

We will derive one such element as an example.

$$\begin{aligned}
\dot{\rho}_{24} &= \dot{c}_2^* c_4 + c_2^* \dot{c}_4 \\
&= \frac{i}{\hbar}\big(V_{24}^* e^{i\omega t} c_4^* + V_{26}^* e^{i\omega t} c_6^*\big) c_4 + c_2^*\big(- i\Delta c_4 - \frac{i}{\hbar}(V_{42} e^{i\omega t}) c_2\big) \\
&= \frac{i}{\hbar}\Big(\frac{\hbar\Omega}{\sqrt{6}} \cos kz \rho_{44} + \frac{\hbar\Omega}{\sqrt{2}} \sin kz \rho_{64}\Big) + \Big(- i\Delta \rho_{24} - \frac{i}{\hbar}\frac{\hbar\Omega}{\sqrt{6}} \cos kz \rho_{22}\Big) \\
&= -i\Delta\rho_{24} + \frac{i\Omega}{\sqrt{2}} \sin kz \rho_{64} - \frac{i\Omega}{\sqrt{6}} \cos kz (\rho_{22} - \rho_{44})
\end{aligned} \tag{A.26}$$

We can also derive one of the ground state population terms.

$$\begin{aligned}
\dot{\rho}_{11} &= \dot{c}_1^* c_1 + c_1^* \dot{c}_1 \\
&= \frac{i}{\hbar}\big(V_{13}^* e^{i\omega t} c_3^* + V_{15}^* e^{i\omega t} c_5^*\big) c_1 - c_1^* \frac{i}{\hbar}\big(V_{13} e^{-i\omega t} c_3 + V_{15} e^{-i\omega t} c_5\big) \\
&= \frac{i}{\hbar}\left(\frac{\hbar\Omega}{\sqrt{2}} \cos kz \rho_{31} + \frac{\hbar\Omega}{\sqrt{6}} \sin kz \rho_{51}\right) - \frac{i}{\hbar}\left(\frac{\hbar\Omega}{\sqrt{2}} \cos kz \rho_{13} + \frac{\hbar\Omega}{\sqrt{6}} \sin kz \rho_{15}\right) \\
&= \frac{i\Omega}{\sqrt{2}} \cos kz [\rho_{31} - \rho_{13}] + \frac{i\Omega}{\sqrt{6}} \sin kz [\rho_{51} - \rho_{15}]
\end{aligned} \tag{A.27}$$

A.2.6 Inclusion of spontaneous emission and relaxation

So far we have only considered the driven transitions in the atom and not considered any of the spontaneous emission that may also occur. We can ad hoc introduce those terms into the elements of the density matrix. The spontaneous emission terms are scaled by the scattering rate Γ and the squared Clebsch-Gordon coefficients for that partiuclar transition.

$$\begin{aligned}\dot{\rho}_{11} &= \frac{i\Omega}{\sqrt{2}}cos(kz)[\rho_{31}-\rho_{13}] + \frac{i\Omega}{\sqrt{6}}sin(kz)[\rho_{51}-\rho_{15}] + \Gamma\big(\rho_{33}+\tfrac{2}{3}\rho_{44}+\tfrac{1}{3}\rho_{55}\big)\\ \dot{\rho}_{22} &= \frac{i\Omega}{\sqrt{2}}sin(kz)[\rho_{62}-\rho_{26}] + \frac{i\Omega}{\sqrt{6}}cos(kz)[\rho_{42}-\rho_{24}] + \Gamma\big(\rho_{66}+\tfrac{2}{3}\rho_{55}+\tfrac{1}{3}\rho_{44}\big)\end{aligned} \tag{A.28}$$

We also add the relaxation term to the off-diagonal elements of the density matrix.

$$\begin{aligned}\dot{\rho}_{13} &= \Big[-i\Delta-\tfrac{\Gamma}{2}\Big]\rho_{13} + \frac{i\Omega}{\sqrt{6}}sin(kz)\rho_{53} - \frac{i\Omega}{\sqrt{2}}cos(kz)\big(\rho_{11}-\rho_{33}\big)\\ \dot{\rho}_{15} &= \Big[-i\Delta-\tfrac{\Gamma}{2}\Big]\rho_{15} + \frac{i\Omega}{\sqrt{2}}cos(kz)\rho_{35} - \frac{i\Omega}{\sqrt{6}}sin(kz)\big(\rho_{11}-\rho_{55}\big)\\ \dot{\rho}_{24} &= \Big[-i\Delta-\tfrac{\Gamma}{2}\Big]\rho_{24} + \frac{i\Omega}{\sqrt{2}}sin(kz)\rho_{64} - \frac{i\Omega}{\sqrt{6}}cos(kz)\big(\rho_{22}-\rho_{44}\big)\\ \dot{\rho}_{26} &= \Big[-i\Delta-\tfrac{\Gamma}{2}\Big]\rho_{26} + \frac{i\Omega}{\sqrt{6}}cos(kz)\rho_{46} - \frac{i\Omega}{\sqrt{2}}sin(kz)\big(\rho_{22}-\rho_{66}\big)\end{aligned} \tag{A.29}$$

A.2.7 Adiabatic elimination of excited states

This optical lattice is in a weak-excitation regime, the time spent in the ground states is significantly greater than the time spent in the excited states. The ground populations of the density matrix dominate as compared to the excited populations and coherences, so we are able to eliminate the non-ground terms. We will also look towards the steady-state solution, because it will be able to describe the long-term behavior of the atom in the lattice. Since we've determined that most time is spent sitting in the well in the ground state, steady-state is a good approximation of an extended simulation in the lattice. Then we can also eliminate all derivatives from the equations.

$$\begin{aligned}\rho_{13ss} &= -\frac{i\Omega cos(kz)}{\big(i\Delta+\frac{\Gamma}{2}\big)\sqrt{2}}\rho_{11}\\ \rho_{15ss} &= -\frac{i\Omega sin(kz)}{\big(i\Delta+\frac{\Gamma}{2}\big)\sqrt{6}}\rho_{11}\\ \rho_{24ss} &= -\frac{i\Omega cos(kz)}{\big(i\Delta+\frac{\Gamma}{2}\big)\sqrt{6}}\rho_{22}\\ \rho_{26ss} &= -\frac{i\Omega sin(kz)}{\big(i\Delta+\frac{\Gamma}{2}\big)\sqrt{2}}\rho_{22}\end{aligned} \tag{A.30}$$

A.2.8 Steady-state solutions for coherences and populations

Now using these terms reevaluate the population terms. Despite the fact that we've just eliminated the excited population terms from the coherences, the ground state populations don't have one dominating that allows the elimination of the others. Instead, we'll take these steady-state solutions from the coherences and the populations.

$$\begin{aligned}
\rho_{33ss} &= s_o cos^2(kz)\rho_{11} \\
\rho_{44ss} &= \frac{s_o}{3} cos^2(kz)\rho_{22} \\
\rho_{55ss} &= \frac{s_o}{3} sin^2(kz)\rho_{11} \\
\rho_{66ss} &= s_o sin^2(kz)\rho_{22}
\end{aligned} \tag{A.31}$$

Finally we return to the equations for the populations of the ground states. By plugging in A.31 and A.30 to the equations from A.28 we end upon the two ground populations.

$$\begin{aligned}
\rho_{11ss} &= cos^2(kz) \\
\rho_{22ss} &= sin^2(kz)
\end{aligned} \tag{A.32}$$

Rather than an empirical solution to a very complicated system of equations, we have very easy-to-visualize equations that are only dependent on position.

A.3 Wigner transform and the Fokker-Planck equation

Our density matrix still lacks the time-evolution needed for the Monte Carlo simulation. A Fokker-Planck equation can describe this evolution. We will use a Wigner distribution to represent the atom as it is often used for a semi-classical probability distribution for internal atomic states. We derive the Wigner distribution through the Wigner transform given by

$$\begin{aligned}
W(z,p,t) &= \int_{-\infty} \infty dz' \langle z + \frac{z'}{2} | \rho(t) | z - \frac{z'}{2} \rangle e^{-ipz'/\hbar} \\
W(z,p,t) &= \int_{-\infty} \infty dp' \langle p + \frac{p'}{2} | \rho(t) | p - \frac{p'}{2} \rangle e^{-ip'z/\hbar}
\end{aligned} \tag{A.33}$$

$$\begin{aligned}
\left(\frac{\partial}{\partial t} + \frac{p}{m}\frac{\partial}{\partial z} - \frac{\partial U_\pm}{\partial z}\frac{\partial}{\partial p} - \frac{\hbar^2 k^2 \Gamma'}{90}(35 \pm 7\cos 2kz)\frac{\partial^2}{\partial p^2} \right) W_{\pm\pm} = \\
\mp \frac{\Gamma'}{9}(1 + \cos 2kz) W_{++} \pm \frac{\Gamma'}{9}(1 - \cos 2kz) W_{--} + \\
\frac{\hbar^2 k^2 \Gamma'}{90}(6 \mp \cos 2kz)\frac{\partial^2 W_{\mp\mp}}{\partial p^2}
\end{aligned} \tag{A.34}$$

We now have an equation that describes the motion of the atom that is state-dependent. The terms on top regard the diffusive motion from an atom staying in the same well, while the very bottom terms are the diffusive motion of an atom changing wells. The terms in the middle portion describe transitions between the ground states.

A.4 Groundwork for Monte-Carlo simulation

We can now pick pieces from the Wigner distribution for use in the Monte Carlo simulation. In each step of the simulation, we will have updates on the momentum, position, and internal state of the atom. We start by establishing what the potentials are that the atom is trapped in. The bipotential here is created because the different ground states of the atom experience a different light-shift based on the polarization of the light. This light-shift is given as

$$\Lambda(z) = \frac{1}{3}(2 + \cos 2kz)\rho_{11} + \frac{1}{3}(2 - \cos 2kz)\rho_{22} \tag{A.35}$$

Then each potential is given by

$$\begin{aligned} U_{\pm}(z) &= \left\langle g_{\pm 1/2} \right| \hbar\delta' \Lambda \left| g_{\pm 1/2} \right\rangle \\ &= \frac{U_o}{2}(-2 \pm \cos 2kz) \end{aligned} \tag{A.36}$$

Now knowing the potential the atom is in, we can ask about the evolution of the internal state. At any point in the simulation, the atom has a probability of changing states as given by

$$\begin{aligned} \gamma_{+-}(z) &= \frac{2}{9}\Gamma' \cos^2 kz \\ \gamma_{-+}(z) &= \frac{2}{9}\Gamma' \sin^2 kz \end{aligned} \tag{A.37}$$

Then for any single step, the probability of jumping from one well to another is just $\gamma +_{\pm\mp} dt$. This is a total probability that the atom was pumped by a lattice beam and spontaneously emitted into the opposing ground state.

The momentum will be adjusted with each step then from two sources, the potential and the diffusion. If The atom stays in the same well, the diffusion is

$$\Delta p = -\frac{dU_{\pm}}{dz}dt + \sqrt{2D_{\pm\pm}dt} \tag{A.38}$$

or if the atom changes wells during a time step

$$\Delta p = -\frac{dU_{\pm}}{dz}dt + \sqrt{\frac{2D_{\pm\mp}}{\gamma_{\pm\mp}}} \tag{A.39}$$

Finally we integrate momentum to update the position of the atom as

$$\Delta z = \frac{\Delta p}{M}dt \tag{A.40}$$

With these equations, we now have a step-by-step evolution of the internal and external variables of the atom.

Appendix B

Bad lattice simulations

Though not good for accurate results, there are characteristics that arise in a simulation because of various mistakes in how the lattice was implemented. For those so inclined, this can satiate the curious "what if?" for different cases like extremely rapid spontaneous emissions or the true zero-dissipation circumstance that our symmetry analysis had depended on. In this section I'd like to cover what some of these cases look like so that anyone trying to run or create their own simulations might identify their bugs more quickly than I.

The first example we'll see is when the spontaneous emission is zero. That means that there is no contribution from diffusion in the momentum change of the atom. This may also be a case where the diffusion is an order of magnitude too low or some similar case. Notice that unlike the proper graph shown in Chp. 3, there is no slope upwards for shallow potential wells. This would seem to imply that we could reach condensate temperatures in a dissipative lattice with no evaporation. There is no immediate solution to this issue besides looking through every line and variable.

A very similar graph is made when the atoms don't have enough time in the lattice to undergo those spontaneous emissions. The atoms don't reach an equilibrium in this case. Depending on the initial conditions passed to the atom, this may not manifest the same in every simulation. If the atom begins with enough momentum it may appear as if the simulation is running accurately, though the atoms don't have enough time to reach an equilibrium.

The final circumstance is if the spontaneous emissions have been overstated in the simulation. This particular example boosts the diffusion from spontaneous emission by an order of magnitude. Even a small change like this has an extremely significant difference. No part of the original graph shows any resemblance to what we have produced in Fig. B.3.

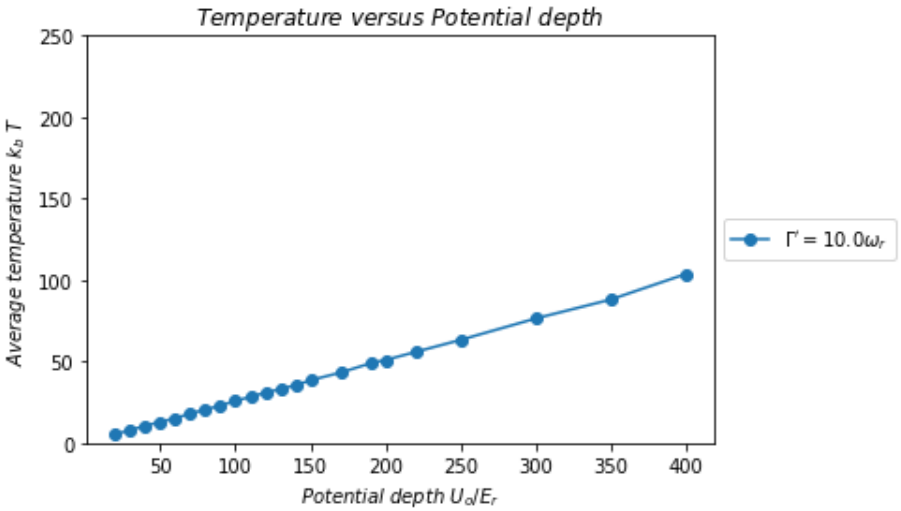

Figure B.1: The lattice simulation but where kicks from spontaneous emission have been artificially changed to zero.

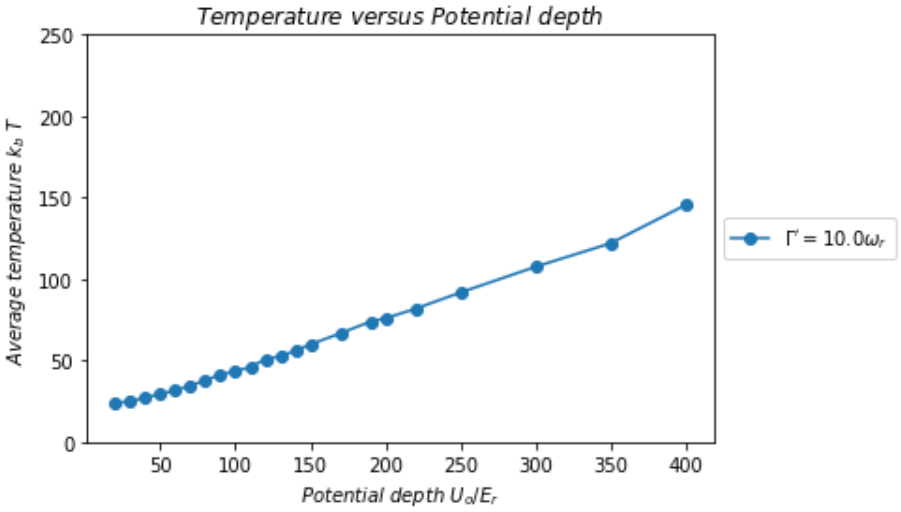

Figure B.2: The same lattice simulation as presented in Chp. 3 but with only a tenth of the time steps. The atoms are not able to reach equilibrium before the simulation ends.

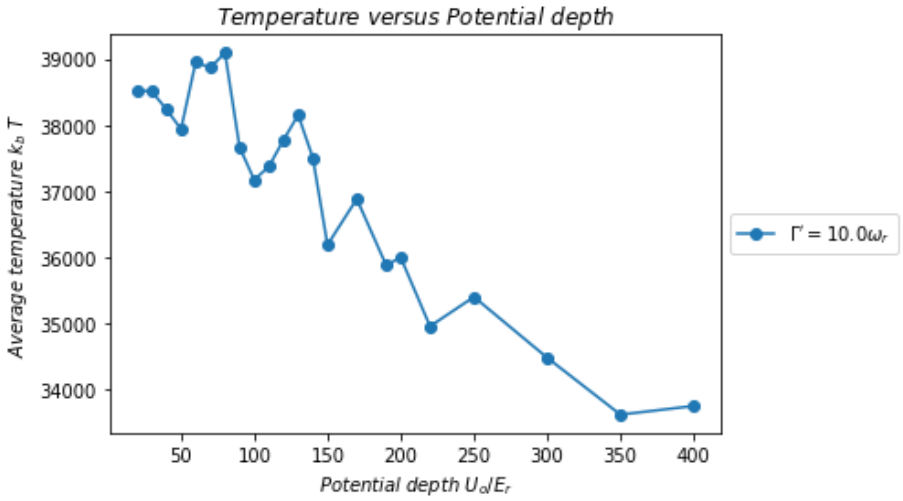

Figure B.3: The lattice simulation with the diffusion multiplied by ten. The dissipation is beyond dominant, and the trend of the plot unrecognizable.

Appendix C

Simulation Codes


```
! This is a collection of all codes
! Each block is the Fortran, bash, inputs, and plotting
! Begin with the lattice code DLattice.f
program DLattice

! Beginning a new version of the DLattice code
! This is using my naming scheme and conventions from Fortran 90
! MPI is included in this, but hardly necessary
! The code runs very quickly as it doesn't have that many iterations to go through

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Start by figuring out the variables we're going to be using
! This is very different from some of the other sample code we're using
! I prefer this way even if it's really tedious, using implicit none makes sure there's no rogue variables
```

```
implicit none
! Variables for the simulation loops
integer :: i,j,U_num,u ! for the loops
! Remember that atom_num is multiplied by the number of processors in MPI
integer :: nstate,atom_num,tsteps,idum
real*8 :: gammaP_in,dt_in,U_in
real*8 :: state,gammaP,U0,Dfsn,kick,deltaP,jumprate
real*8 :: p,z,t,dt
real*8 :: ninth,cos2z,rand,gasdev
real*8 :: p2sum,restmp,tmpavg
real*8, allocatable :: U_Er(:),finaltemp(:)

! Variables for MPI
integer :: ierr,myid,numprocs

! Variables for the random seed generator
integer, allocatable :: seed(:)
integer :: n
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Now we need to establish the MPI stuff
! This isn't entirely necessary for just the DLattice, but we'll be
! using this later as the base for the DBiharmonic and DMultiFreq.

include 'mpif.h'

! Set up the variables we'll need to for the MPI stuff

! Now start the MPI function, give the ID of each process
```

```
call MPI_INIT (ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,numprocs,ierr)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Now we'll read from the input files for the program

! All input files are in 10-series, out in 20-series
! Are your values given in a column rather than row?
! Then your read functions need to also be in a column
open(11,file='lattice_inputs.txt')
read(11,*) atom_num
read(11,*) tsteps
read(11,*) U_num
read(11,*) GammaP_in
read(11,*) dt_in
! This is essentially asking how many potentials should we be looking for
! That way we can use some arbitrary number of potentials
allocate(U_Er(U_num))
allocate(finaltemp(U_num))
open(12,file='lattice_potentials.txt')
read(12,*) U_Er



!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Now we need a random number generator
! If we just feed the processors the same seed, they'll produce the
```

```
! same results for the simulation. This is bad.
! So intead, we are going to create seeds for the number generator that
! is based on a combination of time and processor ID. That way, the
! simulation I run tomorrow is different from today, and each process
! will be running its own simulation, rather than copying.

! Special note here!
! When you ask for the random_seed generator like this, it calls from the OS
! That means it's a little dependent on how good that RNG actually is
! Here, I'm trusting the OS to give me good random numbers
! If this is in doubt, it needs to be replaced by something else in a similar manner

call random_seed(size=n)
allocate(seed(n))
call random_seed(get=seed)

call random_seed(put=seed)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! We have all of the variables, now we run the loops

! Do some quick setting up
! Initial state for atoms
nstate = 1
! And state for using in calculations
state = dble(nstate)

! Division takes a long time, do it once here
ninth = 1.d0/9.d0
```

```
!!! Apparently my logic is wrong here

! We are talking hbar, k, and m to be equal to 1
! We scale some variables according to those values
! E_r = hbar^2 k^2 / 2m & E_r = hbar * w_r ==> E_r = 1/2 and w_r = 1/2
! ~U = U0/E_r ==> ~U = 2*U0
! So these values we scale to are effectively halved in the code
! Doing potentials one at a time in the loop

gammaP = gammaP_in * 0.5d0

! We also scale the time step with gamma
! Typical value for dt_in is 0.01
dt = dt_in/gammaP
! Are three for loops bad form?
! Is running scripts for each of these better?
! I don't really care
do u = 1,U_num
! Remember that ACTUAL potential is scaled by Er
! That means U0/Er is U0 / (1/2)
   U0 = U_Er(u) * 0.5d0
! And we also have to set our restmp to 0
   restmp = 0.d0
   tmpavg = 0.d0
! The second loop will scan through our atoms
   do i = 1,atom_num

! Set the initial conditions for the new atom
```

```
! These don't really matter a ton,
      p = gasdev(idum)
      z = gasdev(idum)
      t = 0.d0

! And now run through the time steps
      do j = 1,tsteps

         cos2z = dcos(2.d0*z)
! First take into account the diffusion of the atom
! This is analytical value assuming ONLY diffusion of same-well
! Well jump is unlikely in grand scheme of things, also scales inverse with dt
! Results in sudden VERY large jumps in momentum with very small dt
         Dfsn=(0.1d0*ninth*gammaP)*(35.d0+(state*7.d0*cos2z))
! Then how much momentum does the atom feel from this
! Random direction times the momentum to simulate random direction
         kick = ((2.d0*Dfsn*dt)**(0.5d0))*gasdev(idum)
! Total change in momentum is kick and the force from the potential
         deltaP = kick + (state*U0*dsin(2.d0*z)*dt)

! Now we have to ask if we changed wells or not

! Probability we jumped in this moment
         jumprate = ninth*dt*gammaP*(1.d0+(state*cos2z))

         call random_number(rand)

! Compare and see if we jumped
```

```
         if (rand .lt. jumprate) then
! This is the path if we jumped
            state = -state
         end if
! And update the rest of the variables
         p = p + deltaP
         z = z + p*dt
         t = t + dt

! For the lattice, we're interested in taking the average of momentum
! squared. Easiest way is to take a running sum
! Scale that sum by the total number we'll be summing
 tmpavg = tmpavg + (p*p)/(atom_num*tsteps)

      end do ! end of time steps
! Still waiting for the tmpavg to finish...
   end do ! end of atom loop
! Now our tmpavg for THIS potential is done
! While we have it, let's reduce it and save
   call MPI_REDUCE(tmpavg,restmp,1,MPI_DOUBLE_PRECISION,
     & MPI_SUM,0,MPI_COMM_WORLD,ierr)
! Now we should only do this on one core
! Otherwise it would just do this 8 times
   if (myid.eq.0) then
! Now we have the all summed together restmp
! We need to divide by number of processes to get true average
! Then converting to kb T we need to multiply by 2
      restmp = 2.d0 * restmp/numprocs
```

```
      open(unit=21,file='out.dat',position='append')
      write(21,*) restmp, U_Er(u), gammaP_in
   end if

end do ! end of potentials loop


! Now since we just did all of the saving in the loop, no need for end section

call MPI_FINALIZE(ierr)
end program

FUNCTION GASDEV(IDUM)
      IMPLICIT REAL*8 (A-H,O-Z)
      IMPLICIT INTEGER*4 (I-N)
      REAL*8 GASDEV,X(2)
      SAVE ISET,GSET
      DATA ISET/0/
      IF(ISET.EQ.0) THEN
 1       CALL RANDOM_NUMBER(X)
         V1=2.0d0*X(1)-1.d0
         V2=2.0d0*X(2)-1.d0
         R=V1**2+V2**2
      IF(R.GE.1.0d0) GO TO 1
         FAC=DSQRT(-2.0d0*DLOG(R)/R)
         GSET=V1*FAC
         GASDEV=V2*FAC
         ISET=1
```

```
      ELSE
         GASDEV=GSET
         ISET=0
      ENDIF
      RETURN
      END

! Bash script DLatticeRunner.sh
#!/bin/bash

GAMMA="2.0 5.0 10.0 15.0"

for gamma in $GAMMA; do

awk -v gamma=$gamma '{if($3 == "GammaPin") {print gamma" ! GammaPin"} else {print $0}}' lattice_inputs0.txt > lattice_inputs.txt
mpirun -n 4 ./DLattice.exe

filename='Latticeout_'$gamma'_gamma.txt'
mv ./out.dat ./$filename
done

! Input data file lattice_inputs
1250 ! atom number
10000 ! time steps
22 ! Number of potentials
15.0 ! GammaPin
0.05 ! dt_in
```

```
! Input data file lattice_potentials
20
30
40
50
60
70
80
90
100
110
120
130
140
150
170
190
200
220
250
300
350
400

! Lattice plotting DLatticePlotter1.py
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Created on Mon May 20 13:45:01 2019

@author: anthony
"""

# DLattice plotting program

# This is a little different from DBiharmonic and DMultiFreq
# Here we will only have one file and folder to work from, so it's not too crazy

# We'll start with the same libraries that we like to use
################################################################################
# Section 0: Importing Libraries
################################################################################

# Responsible for creating all of our pretty graphs at the end
import matplotlib.pyplot as plt
# The bread and butter of working with scientific Python
# Goal is to take more use from this through arrays
import numpy as np
# Very excellent for dealing with files
import tkinter as tk
# Helps us pull up that nice little dialogue box
import tkinter.filedialog as fd
# Really just after curve_fit right now
# We really don't need it for this program anyways though
import scipy as sc
from scipy.optimize import curve_fit
```

```
# Used for dealing with directories and whatnot
import os
# regexp has a wonderful little thing to pull out bits from strings
# Need this for pulling variable data from
import re

#################################################################################
# Section 1: Loading in the file
#################################################################################
# This section is all about finding all of the data from whatever directory

# A diaglogue window will popup asking for where the folders are
# You'll tell it the overarching folder where all of the data folders are
# Data folders should be in the form ***Force_***Potential
# It will pull the Force for each data set based on that folder name

# This diaglogue window will start wherever initialdir says
# You can always just move away from that folder
# If it throughs an error, change so initialdir = '/'
root = tk.Tk()
root.withdraw()
# You can change thsi variable to a known directory if you're doing a lot of work out of the same folder
startdir = '/'
datafile = fd.askopenfilename(initialdir = startdir,title='Where is the file?')

# This is quite a bit simpler than later plotting files
# This saves everything into one big variable
data = np.genfromtxt(datafile)
```

```
# Data will have three columns
# The first is the scaled kbT value
# The second is the potential depth
# The third is the gamma value used
    # For any one data file, unless something funky is going on, the third column should all be the same value


################################################################################
# Secton 2: Plotting
################################################################################
# This is just simple plotting with matplotlib
# No fitting or anything of the sort here
plt.plot(data[:,1],data[:,0],'-o',label=r"$\Gamma'=%s\omega_r$" % data[0,2])
plt.xlabel(r'$Potential\ depth\ U_o/E_r$')
plt.ylabel(r'$ Average\ temperature\ k_b\ T$')
#plt.ylim(0,250)
plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
plt.title(r'$Temperature\ versus\ Potential\ depth$')
plt.show()

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Now begins DBiharmonic
! Begin with DBiharmonic.f
program DBiharmonic

c**************************************************************
  IMPLICIT REAL*8 (A-H,O-Z)
  IMPLICIT INTEGER*4 (I-N)
```

```
  dimension par(20)

c-------------------------------------------------MPI stuff START
      REAL*8, ALLOCATABLE :: SAMPLE(:),SAMPLEB1(:),SAMPLEB2(:)
      CHARACTER*8, ALLOCATABLE :: SAMPNAME(:)

      include 'mpif.h'
      INTEGER,  ALLOCATABLE :: SEED(:),OLDSEED(:),NSEED(:)
      double precision starttime,endtime
      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid,ierr)
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr)
c-------------------------------------------------MPI stuff END
      input=33
  open(unit=nf_in,file='input.dat',status='old')
c num_atom
      read(nf_in,*) num_atom
c   gammaP
      read(nf_in,*) gammaP_in
c   ntstep
      read(nf_in,*) ntstep
c   phi
      read(nf_in,*) phi
c   U_0
      read(nf_in,*) U_in
c   dt0
      read(nf_in,*) dt_in
c   idum
```

```
      read(nf_in,*) idum
c   jdum
      read(nf_in,*) jdum
c   F_0
      read(nf_in,*) F_in

      close(nf_in)
c-------------------------------------------------
c     INITIALLIZE RANDOM NUMBER GENERATOR:
c      CALL SYSTEM_CLOCK(COUNT=idum)
      CALL RANDOM_SEED(SIZE=n)
      ALLOCATE(SEED(n),NSEED(n))
c     Set maximum seeds according to the intel manual:
      seedmax1=2147483562
      seedmax2=2147483398
      nseed(1)=nint(real(seedmax1)/2.0/numprocs)
      nseed(2)=nint(real(seedmax2)/2.0/numprocs)
      DO i=1,n
         SEED(i)=idum+nseed(i)*myid
      enddo
      CALL RANDOM_SEED(PUT=SEED)
      DEALLOCATE(SEED)
c-------------------------------------------------
C     SAMPLE VARIABLES: (BLOCK DATA AVERAGE)

c     Setting NBLOCK to number of processors
      nblock=numprocs
```

```
      NSAMPLE=2
      ALLOCATE(SAMPLE(NSAMPLE),SAMPLEB1(NSAMPLE))
      ALLOCATE(SAMPLEB2(NSAMPLE),SAMPNAME(NSAMPLE))
      SAMPNAME(1)='<v>    :'
      SAMPNAME(2)='Diff   :'

      DO I=1,NSAMPLE
         SAMPLEB1(I)=0.D0
         SAMPLEB2(I)=0.D0
      ENDDO

c*****************************************************************
!      nf=51 222  format(a9,20(f20.13))
!      open(unit=nf,file='p_list.dat',status='unknown')
      nstate = 1
  state=dble(nstate)
      oneninth = (1.d0/9.d0)


! Both of these values are scaled by a 1/2.
! Try to match real values based on units
      gammaP = gammaP_in * 0.5d0
      F_0 = F_in * 0.5d0

      dt=dt_in/gammaP_in
      tsample=dt*ntstep

! This line is a correction of the potential from the last version
```

```
! Now you feed in the U0/Er value, and this halves it for the actual U value

      U_0 = U_in * 0.5d0

      w_d = dsqrt(U_in)
  pi= 3.141592653589793d0
  period=2.d0*pi/w_d

      A_d = 1.d0
      B_d = 1.d0

c     Write parameters to output file
      if(myid.eq.0)then
         nf_ou=10
         open(unit=nf_ou,file='output.dat',status='unknown')
         write(nf_ou,*) 'dt = ',dt
         write(nf_ou,*)'U0/Er=',U_overER
         write(nf_ou,*)'tsample=',tsample
         write(nf_ou,*)'period=',period
         write(nf_ou,*) 'phi = ',phi
         write(nf_ou,*) 'F_0 = ',F_0
         write(nf_ou,*) 'nblock=',nblock
         close(nf_ou)
      endif

!      CALL CPU_TIME ( time_begin)
      starttime=MPI_WTIME()
```

```
c*****************************************************************
c     Transition time for adiabatic turn on

      ramptime = 10.d0

!  rampup=(1.d0 - exp(-t/ramptime))

c*****************************************************************
      xmean=0.d0
      xmean2=0.d0
      np=0
      vmean=0.d0

c     Start the party

      do i=1,num_atom


        z0=gasdev(idum)
        p0=gasdev(idum)
        t0=0.d0

        p = p0
        z = z0
t = t0

        do j=1,ntstep
```

```
          cosz2=dcos(2.d0*z)
          Dpp = (gammaP/90.d0)*(35.d0 + state*7.d0*cosz2)
          punch = ((2.d0*Dpp*dt)**(0.5d0))*gasdev(idum)
          biharmonic = F_0*(A_d*dcos(w_d*t)+B_d*dcos(2.d0*w_d*t+phi))
          drive = biharmonic
          deltaP = (U_0*sin(2.d0*z)*dt)*state + punch + drive*dt


 call RANDOM_NUMBER(rand)

          ratejump = (oneninth)*dt*gammaP*(1.d0 + state*cosz2)
          if (rand.lt.ratejump)then !you change wells
nstate = - nstate
state=dble(nstate)
!           print*, 'switch'
 endif

          p = p + deltaP
          z = z + (p*dt)
 t = t + dt

        end do

c       drift and diffusion calculation
        np=np+1
        xd = z - z0
        timed = t - t0
        vmean=vmean+xd/timed
```

```
          xmean=xmean+xd
          xmean2=xmean2+xd*xd

        end do

c       ---------------------------------------------------
c       SAMPLE VARIABLES:

        vmean=vmean/dble(np)
        xmean=xmean/dble(np)
        diff=(xmean2/dble(np)-xmean*xmean)/2.d0/timed
        SAMPLE(1)=vmean
        SAMPLE(2)=diff

C       STORE SAMPLE VARIABLES:
c       Gather data from all processes
        ndim=nsample
        call MPI_REDUCE(sample,sampleb1,ndim,MPI_DOUBLE_PRECISION,
     &       MPI_SUM,0,MPI_COMM_WORLD,ierr)
        do k=1,nsample
           sample(k)=sample(k)*sample(k)
        enddo
        call MPI_REDUCE(sample,sampleb2,ndim,MPI_DOUBLE_PRECISION,
     &       MPI_SUM,0,MPI_COMM_WORLD,ierr)

        if(myid.eq.0)then

         open(unit=nf_ou,file='output.dat',status='old',position='append')
```

```
c     Drift and effective diffussion

         write(nf_ou,*)'Block data: '

C     SAMPLE VARIABLES:
         DO K=1,NSAMPLE
            SAMPLEB1(K)=SAMPLEB1(K)/DBLE(NBLOCK)
            SAMPLEB2(K)=(SAMPLEB2(K)-DBLE(NBLOCK)*SAMPLEB1(K)
     &           *SAMPLEB1(K))/DBLE(NBLOCK-1)
            write(nf_ou,222)SAMPNAME(K),SAMPLEB1(K)
            write(nf_ou,222)'    (dis)',DSQRT(SAMPLEB2(K))
         ENDDO

c      Peclet:
       pi= 3.141592653589793d0
       pe=sampleb1(1)*pi/sampleb1(2)
       write(nf_ou,222)'Peclet : ',pe

c      CALL CPU_TIME ( time_end)
         endtime=MPI_WTIME()
         write(nf_ou,*)
     &        'total time spent in root ID:'
     &        ,endtime-starttime,' seconds ('
     &        ,(endtime-starttime)/60.,' minutes)'
c      write(nf_ou,*)
c      write(nf_ou,*)'total CPU time spent:'
c     &     ,time_end-time_begin,' seconds'
```

```
         close(nf_ou)
      endif

 222  format(a9,20(f20.13))
      call MPI_FINALIZE(ierr)
      end

*****************************************************************
* Algorithms to generate random numbers(Ref. W.H.Press et Al.  *
* "NUMERICAL RECIPES".Cambridge U.P. 1986)                      *
* Function GASDEV generates a Gaussian Distribution            *
* of zero mean and variance unity.(Box-Mueller formula)        *
*****************************************************************
      FUNCTION GASDEV(IDUM)
      IMPLICIT REAL*8 (A-H,O-Z)
      IMPLICIT INTEGER*4 (I-N)
      REAL*8 GASDEV,X(2)
      SAVE ISET,GSET
      DATA ISET/0/
      IF(ISET.EQ.0) THEN
 1       CALL RANDOM_NUMBER(X)
         V1=2.0d0*X(1)-1.d0
         V2=2.0d0*X(2)-1.d0
         R=V1**2+V2**2
      IF(R.GE.1.0d0) GO TO 1
```

```
         FAC=DSQRT(-2.0d0*DLOG(R)/R)
         GSET=V1*FAC
         GASDEV=V2*FAC
         ISET=1
      ELSE
         GASDEV=GSET
         ISET=0
      ENDIF
      RETURN
END

! Now bash scripts to run DBiharmonic starting with BiharmShell2.sh
#!/bin/bash

GAMMA="2.0 3.0 4.0 5.0 6.0 7.5 9.0 11.0 15.0 20.0 25.0"
FF="250 300 350 400"
for ff in $FF; do
echo "Starting F0 = "$ff
newdir=$ff'Force_200Potential'
mkdir $newdir
awk -v F0=$ff '{if($3=="F_0") {print F0"  ! F_0 in units of Fr"} else {print $0}}' inputstart > ./inputstep
for gamma in $GAMMA; do
echo "Now running gamma=$gamma"
awk -v gamma=$gamma '{if($3=="gammaP") {print gamma" ! gammaP"} else {print $0}}' inputstep > ./input0

./FortranRunner.sh

./DataCollector.sh > ./$newdir/${gamma}Gam.dat
```

```
done
done

! And then DataCollector.sh
dirs=$(ls -d PHI_*)

echo "# phi ; <v> ; error <v> ; diff ; error diff ; Peclet"

for dir in $dirs ; do

cat $dir/output.dat | awk '{if($1=="phi") {phi=$3} if($1=="<v>") {v=$3 ; getline ; dv=$2 ; \
 getline ; d=$3 ; getline ; dd=$2 ; getline ; pe=$3 }}END{print phi"   "v"   "dv"    "d"     "dd"    "pe}'

done

! And FortranRunner.sh
#!/bin/bash

PHI="    0.0000000000000     0.3141592653590     0.6283185307180     0.9424777960769     1.2566370614359     1.5707963267949

for phi in $PHI; do
    mydir='PHI_'$phi
    mkdir $mydir
    awk -v phi=$phi '{if($3 == "phi:") {print phi" ! phi: phase difference"} else {print $0}}' input0 > $mydir/input.dat
    cd $mydir
    mpirun -n 8 ../a.out
    cd ..
done
```

```
! Then we need the input file inputstart
1250        ! atom_num: Number of Atoms
7.5d0         ! gammaP
100000       ! ntstep
1.570796327  ! phi: phase difference
200.d0          ! U_0: potential depth
0.05        ! dt0: time step to be scaled by gammaP
3456789      ! IDUM: random number genetator seed
219823       ! JDUM: 2nd random number generator seed
0           !  F_0
0            ! NOTHING: not used

! And the plotting program DBiPlot6.py
# -*- coding: utf-8 -*-

# From the ashes of version 5 version 6 returns anew.
# I'm starting to make enough changes in the formatting that a new version number is in order
# Nothing fundamentally has changed, it's just significantly prettier

################################################################################
# Section 0: Importing Libraries
################################################################################

# Responsible for creating all of our pretty graphs at the end
import matplotlib.pyplot as plt
# The bread and butter of working with scientific Python
# Goal is to take more use from this through arrays
```

```
import numpy as np
# Very excellent for dealing with files
import tkinter as tk
# Helps us pull up that nice little dialogue box
import tkinter.filedialog as fd
# Really just after curve_fit right now
import scipy as sc
from scipy.optimize import curve_fit
# Used for dealing with directories and whatnot
import os
# regexp has a wonderful little thing to pull out bits from strings
# Need this for pulling variable data from
import re

################################################################################
# Section 1: Loading in files
################################################################################
# This section is all about finding all of the data from whatever directory

# A diaglogue window will popup asking for where the folders are
# You'll tell it the overarching folder where all of the data folders are
# Data folders should be in the form ***Force_***Potential
# It will pull the Force for each data set based on that folder name

# This diaglogue window will start wherever initialdir says
# You can always just move away from that folder
# If it throughs an error, change so initialdir = '/'
root = tk.Tk()
```

```
root.withdraw()
startdir = fd.askdirectory(initialdir = '/',title='Where are the folders?')
#startdir = '/home/anthony'
# Then tell Python where we're going to be working from
os.chdir(startdir)
print('Working out of %s' % startdir)
# And we'll figure out everything that's in the folder we've selected
biglist = (os.listdir(startdir))
# But now we need to isolate only the folders with data that we want
# We'll save them all here
foldlist = []
# Now we check if the folders match our naming scheme
for i in range(0, len(biglist)):
    # This is kind of a sloppy workaround, but whatever
    if 'Pot' in biglist[i]:
        foldlist.append(biglist[i])
        # Now we'll have a list of only the folders with good stuff in them

# We're going to make a list of all data
datalist = []
# It'd be nice to know what forces and gammas are used too
forcelist = []
gammalist = []
for i in range(0,len(foldlist)):
    # Start with a blank file list each time
    # This creates a string of the path name for a particular data folder
    path = (startdir + '/' + foldlist[i])
    # Now we change directory to that particular folder
```

```
# Remember to change back afterwards!
os.chdir(path)
# And produce a list of those files in our folder
filelist = (os.listdir(path))

for j in range(0,len(filelist)):
    # Now we're going to go through a load each of the files
    # Give this thing a name for what we're going to be saving
    # tempname is going to be the ***Gam.dat file
    tempname = filelist[j]
    # Combine with folder name, trim off '.dat'
    newname = foldlist[i] + '_' + tempname[:-4]
    # Load a data file into a temporary variable
    # This now has six columns of data we want
    tempload = np.genfromtxt(filelist[j])
    # From the file name,
    rs = re.findall("[-+]?[.]?[\d]+(?:,\d\d\d)*[\.]?\d*(?:[eE][-+]?\d+)?",newname)
    tempgam = float(rs[2])
    gammalist.append(tempgam)
    tempforce = float(rs[0])
    forcelist.append(tempforce)
    for k in range(0,len(tempload)):
        # This is in a particular order for data
        # Element 1 is the force F0, element 2 is the gamma
        # element 3 is the phi, element 4 is the average velocity
        # element 5 is the error of that velocity, 6 is the diffusion
        # 7 the error in diffusion, 8 the Peclet number
        # Filthy backend, clean frontend
```

```
            temptuple = (tempforce,tempgam,tempload[k,0],tempload[k,1],tempload[k,2],tempload[k,3],tempload[k,4],tempload[k,5])
            datalist.append(temptuple)

# Now we have this big ol' list of all the numbers we want
# Turn it into an array
darray = np.array(datalist)
# Now any time we want a particular value, we just have to use
# array[array[:,*] == value]

# Lastly, remove any duplicates from the forces and gammas
forcelist = list(set(forcelist))
gammalist = list(set(gammalist))
gammalist.sort(key=float)
forcelist.sort(key=float)

################################################################################
# Section 2: Analysis
################################################################################
# We don't really need the breaking apart like from version 2

# Give the function that we'll be fitting with
# x is the phis we feed in
def func(x,A,phi0):
    return A*(np.sin(x-phi0))

fittedlist = []
for i in range(0,len(forcelist)):
    # Let's pull out all of the data with a single force
```

```
    tempforcearray = darray[darray[:,0] == forcelist[i]]
    # Now we want to do a similar thing to get the gammas pulled out individually
    for j in range(0,len(gammalist)):
        # This is an array of ONLY one force and one gamma at a time
        tempgammaarray = tempforcearray[tempforcearray[:,1] == gammalist[j]]
        # Fit our function to the data from this set
        # fittemp[0] is A, fittemp[1] is phi0
        # Set bounds so that A can't be smaller than 0
        # phi can only be between negative and positive pi
        fittemp,fiterror = curve_fit(func,tempgammaarray[:,2],tempgammaarray[:,3],bounds=((0,-.6*np.pi),(1000000,1.5*np.pi)))
        # Now we'll save the force, gamma, A, and phi0
        temptuple = (forcelist[i],gammalist[j],fittemp[0],fittemp[1])
        fittedlist.append(temptuple)

# And I like these NumPy arrays so I'll convert the fittedlist
fittedarray = np.array(fittedlist)

#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#
# Doing the Peclet Shit
#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#!#

peclist = []
for i in range(0,len(forcelist)):
    # Let's pull out all of the data with a single force
    tempforcearray = darray[darray[:,0] == forcelist[i]]
    # Now we want to do a similar thing to get the gammas pulled out individually
    for j in range(0,len(gammalist)):
        # This is an array of ONLY one force and one gamma at a time
```

```
        tempgammaarray = tempforcearray[tempforcearray[:,1] == gammalist[j]]
        # ...but instead finds the max Peclet number of all of them
        # Just in case that number isn't exactly at the pi/2 phase
        maxPec = max(tempgammaarray[:,7])
        # Same old method of saving to a tuple
        temptuple = (forcelist[i],gammalist[j],maxPec)
        # Then putting it all into a big list of tuples
        peclist.append(temptuple)

# Now we define a function for plotting like the later sections
def pecplot():
    # Convert that list of tuples into a Numpy array
    pecarray = np.asarray(peclist)
    # Then break it into generic x, y, and z
    x = pecarray[:,0] # This is magnitude of force F0
    y = pecarray[:,1] # This is gammaP value
    z = pecarray[:,2] # This is the corresponding Peclet number
    fig = plt.figure()
    ax = plt.gca(projection='3d')
    surf=ax.plot_surface(x, y, z)
    fig.colorbar(surf)
    plt.show()

##############################################################################
# Secton 3: Plotting
##############################################################################

# Now all that's left is plotting the points together
```

```
colorset = ['r','b','g','m']
def phiplot():
    for i in range(0,len(forcelist)):
        # Now we're doing the same thing as last section with data...
        tempforcearray = darray[darray[:,0] == forcelist[i]]
        #...but also doing the same thing for pulling the A and phi0 values
        tempfitarray = fittedarray[fittedarray[:,0] == forcelist[i]]
        for j in range(0,len(gammalist)):
            tempgammaarray = tempforcearray[tempforcearray[:,1]==gammalist[j]]
            tmpfitarray2 = tempfitarray[tempfitarray[:,1] == gammalist[j]]

            aaa = len(gammalist)
            color = ((aaa-j)/aaa,0,j/aaa)
            plt.plot(tempgammaarray[:,2],tempgammaarray[:,3],'o',,label=r"$\Gamma'=%s\omega_r$" % gammalist[j],xunits=np.radians)
            ydata = func(tempgammaarray[:,2],tmpfitarray2[0,2],tmpfitarray2[0,3],)
            plt.plot(tempgammaarray[:,2],ydata,xunits=np.radians)
        plt.hlines(0,0,2*np.pi,colors='k',alpha=0.5,linestyles='--')
        ax = plt.gca()
        plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
        plt.ylabel(r'$Average\ velocity\ \langle v\rangle /v_r$')
        plt.xlabel(r'$\phi\ (radians)$')
        ax.set_xticks([0,.5*np.pi,np.pi,1.5*np.pi,2*np.pi])
        ax.set_xticklabels(["$0$",r"$\frac{1}{2}\pi$",r"$\pi$",r"$\frac{3}{2}\pi$",r"$2\pi$"])
        plt.title(r'$Average\ velocity\ for\ F_o = %s F_r$' % forcelist[i])
        plt.show()

# Now we'll plot the A and phi0 values
# Start with A
```

```
def IplotA():
    for i in range(0,len(forcelist)):
        tempforcearray = fittedarray[fittedarray[:,0] == forcelist[i]]
        plt.plot(tempforcearray[:,1],tempforcearray[:,2],label=r"$F_o = %s$" %forcelist[i])

    plt.xlabel(r"$\Gamma'/\omega_r$")
    plt.ylabel('Atomic current '+r'$I$')
    #plt.title(r'$Fitted\ amplitude\ versus\ scattering\ rate$')
    plt.title('Atomic current '+r'$I$'+' versus scattering rate')
    plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
    plt.show()
def IplotB():
    for i in range(0,len(gammalist)):
        tempgammaarray = fittedarray[fittedarray[:,1] == gammalist[i]]
        plt.plot(tempgammaarray[:,0],tempgammaarray[:,2],label=r"$\Gamma'=%s\omega_r$" % gammalist[i])

    plt.xlabel(r"$Driving\ force\ F_o/F_r$")
    plt.ylabel('Atomic Current '+r'$I$')
    plt.title('Atomic current '+r'$I$'+' versus force')
    plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
    plt.show()

# And now the phi0 values
def phiplotA():
    for i in range(0,len(forcelist)):
        tempforcearray = fittedarray[fittedarray[:,0] == forcelist[i]]
        plt.plot(tempforcearray[:,1],tempforcearray[:,3],label=r"$F_o = %s$" %forcelist[i])
    ax = plt.gca()
```

```
    ax.set_yticks([0,.5*np.pi,np.pi])
    ax.set_yticklabels(["$0$",r"$\frac{1}{2}\pi$",r"$\pi$"])
    plt.xlabel(r"$Scattering\ rate\ \Gamma'/\omega_r$")
    plt.ylabel(r"$Phase\ \phi_o\ (radians)$")
    plt.title('Phase offset versus scattering rate')
    plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
    plt.show()

def phiplotB():
    for i in range(0,len(gammalist)):
        tempgammaarray = fittedarray[fittedarray[:,1] == gammalist[i]]
        plt.plot(tempgammaarray[:,0],tempgammaarray[:,3],label=r"$\Gamma'=%s\omega_r$" % gammalist[i])
    ax = plt.gca()
    plt.xlabel(r"$Driving\ force\ F_o/E_r$")
    ax.set_yticks([0,.5*np.pi,np.pi,1.5*np.pi])
    ax.set_yticklabels(["$0$",r"$\frac{1}{2}\pi$",r"$\pi$",r"$\frac{3}{2}\pi$"])
    plt.ylabel(r"$Phase\ \phi_o\ (radians)$")
    plt.title('Phase offset versus driving force')
    plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
    plt.show()

# Here is the tkinter/menu mini-section
# All plotting should be defined as functions
# Attach those functions to buttons in a pop-up menu
# Allows individualized and specific plotting

# Version 4.2 is looking to turn the tk section into a class rather than lines
# Goal is to fix the bug where an extra tk window pops up
```

```
root = tk.Tk()
frame = tk.Frame(root)
frame.pack()

plotopt1 = tk.Button(frame,text="Phi plot",command=phiplot)
plotopt1.pack()
plotopt2 = tk.Button(frame,text="Amplitude versus gamma",command=IplotA)
plotopt2.pack()
plotopt3 = tk.Button(frame,text="Amplitude versus force",command=IplotB)
plotopt3.pack()
plotopt4 = tk.Button(frame,text="Phase versus gamma",command=phiplotA)
plotopt4.pack()
plotopt5 = tk.Button(frame,text="Phase versus force",command=phiplotB)
plotopt5.pack()
plotopt6 = tk.Button(frame,text="Peclet Surface",command=pecplot)
plotopt6.pack()
button = tk.Button(frame,text="(Click to exit)",fg="red",command=root.destroy)
button.pack(side="bottom")
tk.mainloop()

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Starting the multi-frequency codes
! Begin with DMultiFreq.f
      program DMultiFreq

! As this code seems to be all working now, I'll call it 1.0
! Making this all work as expected requires a couple other pieces
```

```
! Bash: NewMultiShell, FortranRunner, DataCollector
!  FortranRunner2 also exists, with 40 phi points instead of 20
! Text: P, Q, MultiInStart
! From there, NewMultiShell will end up making MultiIn0
! Python: PQ_Plotter
! PQ_Plotter works for both MultiFrequency and Gating
c*************************************************************
  IMPLICIT REAL*8 (A-H,O-Z)
  IMPLICIT INTEGER*4 (I-N)
          REAL*8 :: mdrive1,mdrive2
  dimension par(20)

c-------------------------------------------------MPI stuff START
      REAL*8, ALLOCATABLE :: SAMPLE(:),SAMPLEB1(:),SAMPLEB2(:)
      CHARACTER*8, ALLOCATABLE :: SAMPNAME(:)

      include 'mpif.h'
      INTEGER,  ALLOCATABLE :: SEED(:),OLDSEED(:),NSEED(:)
      double precision starttime,endtime
      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid,ierr)
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr)
!-------------------------------------------------MPI stuff END
! Notes for the curious, 7/28/19
! This code is in full working condition but is a bit ugly
! Many things aren't labeled well, comments aren't great
! A much better example is DGating or DLattice
! Eventually this will be rewritten to follow their style
```

```
!****************************************************************

      input=33
  open(unit=nf_in,file='input.dat',status='old')
c num_atom
      read(nf_in,*) num_atom
c   gammaP
      read(nf_in,*) gamma_in
c   ntstep
      read(nf_in,*) ntstep
c   phi
      read(nf_in,*) phi
c   U_0
      read(nf_in,*) U_in
c   dt0
      read(nf_in,*) dt0
c   idum
      read(nf_in,*) idum
c   jdum
      read(nf_in,*) jdum
c   F_0
      read(nf_in,*) F0_in

      read(nf_in,*) pqratio

      close(nf_in)
c-----------------------------------------------
c     INITIALLIZE RANDOM NUMBER GENERATOR:
```

```
c      CALL SYSTEM_CLOCK(COUNT=idum)
      CALL RANDOM_SEED(SIZE=n)
      ALLOCATE(SEED(n),NSEED(n))
c     Set maximum seeds according to the intel manual:
      seedmax1=2147483562
      seedmax2=2147483398
      nseed(1)=nint(real(seedmax1)/2.0/numprocs)
      nseed(2)=nint(real(seedmax2)/2.0/numprocs)
      DO i=1,n
         SEED(i)=idum+nseed(i)*myid
      enddo
      CALL RANDOM_SEED(PUT=SEED)
      DEALLOCATE(SEED)
c-----------------------------------------------
C     SAMPLE VARIABLES: (BLOCK DATA AVERAGE)

c     Setting NBLOCK to number of processors
      nblock=numprocs

      NSAMPLE=2
      ALLOCATE(SAMPLE(NSAMPLE),SAMPLEB1(NSAMPLE))
      ALLOCATE(SAMPLEB2(NSAMPLE),SAMPNAME(NSAMPLE))
      SAMPNAME(1)='<v>    :'
      SAMPNAME(2)='Diff   :'

      DO I=1,NSAMPLE
         SAMPLEB1(I)=0.D0
         SAMPLEB2(I)=0.D0
```

```
      ENDDO

c*****************************************************************
!      nf=51 222  format(a9,20(f20.13))
!      open(unit=nf,file='p_list.dat',status='unknown')
      nstate = 1
  state=dble(nstate)
      oneninth = (1.d0/9.d0)
      deltaP = 0.d0

      dt=dt0/gamma_in

      tsample=dt*ntstep

      F0=F0_in/2.d0
      gammaP = gamma_in/2.d0

      U_overER = U_in
      U_0 = U_in / 2.d0

      w_1 = dsqrt(U_overER) * .74 ! As per figure 4.9
  pi= 3.141592653589793d0
  !period=2.d0*pi/w_d
      w_2 = (pqratio) * w_1

      A = 1.d0
      B = .3d0
```

```
!     Write parameters to output file
      if(myid.eq.0)then
         nf_ou=10
         open(unit=nf_ou,file='output.dat',status='unknown')
         write(nf_ou,*) 'dt = ',dt
         write(nf_ou,*)'U0/Er=',U_overER
         write(nf_ou,*)'tsample=',tsample
         write(nf_ou,*)'period=',period
         write(nf_ou,*) 'phi = ',phi
         write(nf_ou,*) 'F_0 = ',F0_in
         write(nf_ou,*) 'nblock=',nblock
 write(nf_ou,*) 'pqratio=',pqratio
         close(nf_ou)
      endif

!      CALL CPU_TIME ( time_begin)
      starttime=MPI_WTIME()

c**********************************************************************
c     Transition time for adiabatic turn on

      ramptime = 10.d0

!  rampup=(1.d0 - exp(-t/ramptime))

c**********************************************************************

      xmean=0.d0
```

```
      xmean2=0.d0
      np=0
      vmean=0.d0

c     Start the party

      do i=1,num_atom

! Set the initial conditions, momentum and position random
! Mostly just to make sure not every single atom is starting the exact same way
        z0=gasdev(idum)
        p0=gasdev(idum)
        t0=0.d0
! Now actually say these are the variables of the atom
        p = p0
        z = z0
t = t0

        ! This is the loop for running through the entire time sequence
        do j=1,ntstep

! Use a variable in place os dcos(z*2) every time
         cosz2=dcos(2.d0*z)
! This is diffusion from the spontaneous emission
         Dpp = (gammaP/90.d0)*(35.d0 + state*7.d0*cosz2)
! Actual kick of momentum from emission
         kick = ((2.d0*Dpp*dt)**(0.5d0))*gasdev(idum)
! Split the multi-frequency drive into multiple parts because it's really long
```

```
 sinw1 = dsin(w_1 * t)
 sin2w1= dsin(2.d0 * w_1 * t)
 cosw1 = dcos(w_1 * t)
 cos2w1= dcos(2.d0 * w_1 * t)
 sinw2 = dsin((w_2 * t) + phi)
 cosw2 = dcos((w_2 * t) + phi)
         mdrive1 = F0 * w_1 * sinw2 *((A*cosw1)+(2.d0 * B * cos2w1))
 mdrive2 = F0 * w_2 * cosw2 * ((A * sinw1) + (B * sin2w1))
! Combine the two parts back together
 mdrive = mdrive1 + mdrive2
! Force is a negative, equation pulled directly from Brown thesis
! deltaP is actual change in the momentum of the atom
         deltaP = (U_0*sin(2.d0*z)*dt)*state + kick - mdrive*dt

! We need to ask if the atom changes wells
 call RANDOM_NUMBER(rand)

         jumprate = (oneninth)*dt*gammaP*(1.d0 + state*cosz2)
! Now compare random number to probability of changing
         if (rand.lt.jumprate)then !you change wells
! switch the state
    nstate = - nstate
! Make sure that state is a double for use in the above equations
    state=dble(nstate)
!           print*, 'switch'
 endif

! Now update the variables of the atom
```

```
          p = p + deltaP
          z = z + (p*dt)
t = t + dt
        end do

c       Drift and diffusion calculation
! np is only keeping index on things
        np=np+1
        xd = z - z0
        timed = t - t0
        vmean=vmean+xd/timed
        xmean=xmean+xd
        xmean2=xmean2+xd*xd

      end do

c     ------------------------------------------------------
c     SAMPLE VARIABLES:

      vmean=vmean/dble(np)
      xmean=xmean/dble(np)
      diff=(xmean2/dble(np)-xmean*xmean)/2.d0/timed
      SAMPLE(1)=vmean
      SAMPLE(2)=diff

C     STORE SAMPLE VARIABLES:
c     Gather data from all processes
      ndim=nsample
```

```
      call MPI_REDUCE(sample,sampleb1,ndim,MPI_DOUBLE_PRECISION,
     &       MPI_SUM,0,MPI_COMM_WORLD,ierr)
      do k=1,nsample
         sample(k)=sample(k)*sample(k)
      enddo
      call MPI_REDUCE(sample,sampleb2,ndim,MPI_DOUBLE_PRECISION,
     &       MPI_SUM,0,MPI_COMM_WORLD,ierr)

      if(myid.eq.0)then

       open(unit=nf_ou,file='output.dat',status='old',position='append')

c     Drift and effective diffussion

         write(nf_ou,*)'Block data: '

C     SAMPLE VARIABLES:
         DO K=1,NSAMPLE
            SAMPLEB1(K)=SAMPLEB1(K)/DBLE(NBLOCK)
            SAMPLEB2(K)=(SAMPLEB2(K)-DBLE(NBLOCK)*SAMPLEB1(K)
     &           *SAMPLEB1(K))/DBLE(NBLOCK-1)
            write(nf_ou,222)SAMPNAME(K),SAMPLEB1(K)
            write(nf_ou,222)'    (dis)',DSQRT(SAMPLEB2(K))
         ENDDO

c      Peclet:
       pi= 3.141592653589793d0
       pe=sampleb1(1)*pi/sampleb1(2)
```

```
       write(nf_ou,222)'Peclet : ',pe

c      CALL CPU_TIME ( time_end)
         endtime=MPI_WTIME()
         write(nf_ou,*)
     &        'total time spent in root ID:'
     &        ,endtime-starttime,' seconds ('
     &        ,(endtime-starttime)/60.,' minutes)'
c      write(nf_ou,*)
c      write(nf_ou,*)'total CPU time spent:'
c     &     ,time_end-time_begin,' seconds'

         close(nf_ou)
       endif

 222   format(a9,20(f20.13))
       call MPI_FINALIZE(ierr)
       end
*****************************************************************
* Algorithms to generate random numbers(Ref. W.H.Press et Al.  *
* "NUMERICAL RECIPES".Cambridge U.P. 1986)                      *
* Function GASDEV generates a Gaussian Distribution            *
* of zero mean and variance unity.(Box-Mueller formula)        *
*****************************************************************
       FUNCTION GASDEV(IDUM)
       IMPLICIT REAL*8 (A-H,O-Z)
       IMPLICIT INTEGER*4 (I-N)
       REAL*8 GASDEV,X(2)
```

```
      SAVE ISET,GSET
      DATA ISET/0/
      IF(ISET.EQ.0) THEN
 1       CALL RANDOM_NUMBER(X)
         V1=2.0d0*X(1)-1.d0
         V2=2.0d0*X(2)-1.d0
         R=V1**2+V2**2
      IF(R.GE.1.0d0) GO TO 1
         FAC=DSQRT(-2.0d0*DLOG(R)/R)
         GSET=V1*FAC
         GASDEV=V2*FAC
         ISET=1
      ELSE
         GASDEV=GSET
         ISET=0
      ENDIF
      RETURN
      END

! Now the bash scripts to run it, start with NewMultiShell.sh
#!/bin/bash

echo "I used to say that I am a test script"
echo "Why have a test branch when everything can be master"
echo "I need a P.txt and Q.txt file to work"
# This whole thing is a hot mess
# First we read the text files into arrays
readarray Pall < ./P.txt
```

```
readarray Qall < ./Q.txt
# q is just for simple indexing
echo ${Pall[@]}
echo ${Qall[@]}
q=0
# Cycle through every value of Pall
for p in ${Pall[@]}; do
# For some infuriating reason, the readarray also saves spaces after the numbers
# For most functions, this is just fine
# For bc, the world is ending
# A quick division by 1 (since they're integers) will fix this
tempP=$(($p/1))
tempQ=${Qall[$q]}
tempQ=$(($tempQ/1))
q=$(($q+1))
# "scale" here decides how many digits the ratio will keep
ratio=$(bc <<< "scale=5;($tempP/$tempQ)")
# Now we have the ratio that we'll need to save
awk -v ratio=$ratio '{if($3=="pqratio") {print ratio" ! pqratio"} else {print $0}}' MultiInStart > ./MultiIn0

# FortranRunner for 20 points, FortranRunner2 for 40 points
./FortranRunner2.sh
./DataCollector.sh > ${tempP}over${tempQ}pqratio.dat

done

! Then FortranRunner2.sh
#!/bin/bash
```

```
# FortranRunner2 has exactly one difference of using more phi values, 40 of them instead of just 20
PHI="    0.0000000000000 0.15707963267948966 0.3141592653589793 0.47123889803846897 0.6283185307179586 0.7853981633974483 0.94247

for phi in $PHI; do
    mydir='PHI_'$phi
    mkdir $mydir
    awk -v phi=$phi '{if($3 == "phi:") {print phi" ! phi: phase difference"} else {print $0}}' MultiIn0 > $mydir/input.dat
    cd $mydir
    mpirun -n 8 ../a.out
    cd ..
done

! And finally DataCollector.sh
dirs=$(ls -d PHI_*)

echo "# phi ; <v> ; error <v> ; diff ; error diff ; Peclet"

for dir in $dirs ; do

cat $dir/output.dat | awk '{if($1=="phi") {phi=$3} if($1=="<v>") {v=$3 ; getline ; dv=$2 ; getline ; \
 d=$3 ; getline ; dd=$2 ; getline ; pe=$3 }}END{print phi"   "v"   "dv"    "d"     "dd"    "pe}'

done

! There are three input files, starting with MultiInStart
1250         ! atom_num: Number of Atoms
10.0d0        ! gammaP
```

```
100000       ! ntstep
1.570796327  ! phi: phase difference
200          ! U_0: potential depth
0.05        ! dt0: time step to be scaled by gammaP
3456789      ! IDUM: random number genetator seed
219823       ! JDUM: 2nd random number generator seed
20         ! F_0: force amplitude in units of F_r
1.d0 ! pqratio
! Then list of P values P.txt
1
1
2
1

! List of Q values Q.txt
3
2
3
1

! And finally the plotting program PQ_Plotter.py
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Most recent edition for plotting the Multi-Frequency data AND the gating data
# Previous version was BasicMultiPlotv2_1

# This is looking for data files with the name **over**pqratio.dat
```

```
# The 'over' is the important bit

# One large running issue is that the fitting doesn't always play nice with the data
# In particular, when the denominator is even, there is almost no net motion
# Not only is it very small and hard to fit, it's also very complex
# The fit looks nothing like the data
# Ignore it, because we're just going to use v_max anyways

################################################################################
# Section 0: Import the libraries
################################################################################

import numpy as np
import matplotlib.pyplot as plt
import tkinter as tk
import tkinter.filedialog as fd
import scipy.optimize as opt
import os
import re

################################################################################
# Section 1: Just about everything else
################################################################################

# This one isn't quite as pretty as I had in previous versions
# That also means it's a lot less complicated though

root = tk.Tk()
```

```
root.withdraw()
# Start by opening a window that leads us to the data
# I'm assuming only one folder of data right now
startdir = fd.askdirectory(initialdir='/home/',title='Open the folder with data files')
# Change to that directory
os.chdir(startdir)
# Now pull a list of everything in that folder
startlist = os.listdir(startdir)
filelist = []
# Sort it out so that we only take the pieces in our naming scheme
for i in range(0,len(startlist)):
    if 'over' in startlist[i]:
        # Save all of those to a list of the data files only
        filelist.append(startlist[i])

# This is the fitting function we'll use to match the data
# This follows what Brown used
def fitfunc(x,A,B,phi0):
    return A*np.sin(x*B - (np.pi*.5) - phi0)
parray = []
qarray = []
datalist = []
#colorset=['r','b','m','g','c']
# Now we'll run through each file and do everything in one swing
for i in range(0,len(filelist)):
    # Load the file in a numpy array
    temparray=np.genfromtxt(filelist[i])
    # Search for the p and q values and save them as integers
```

```
        rs = re.findall("[-+]?[.]?[\d]+(?:,\d\d\d)*[\.]?\d*(?:[eE][-+]?\d+)?",filelist[i])
        p = int(rs[0])
        q = int(rs[1])
        parray.append(p)
        qarray.append(q)
        # Plug in the data to a fitting program
        fitvar,fiterr = opt.curve_fit(fitfunc,temparray[:,0],temparray[:,1],p0=(1,q,0))
        # Use the fitted parameters to make a fitted plot
        fittedcurve = fitfunc(temparray[:,0],fitvar[0],fitvar[1],fitvar[2])
        # Temp values for saving into a tuple
        tmpval1 = p/q
        tmpval2 = p*q
        # This tuple will get thrown into an array later
        # In order, it's p/q, then p*q, then max velocity
        temptuple = (tmpval1,tmpval2,max(temparray[:,1]))
        datalist.append(temptuple)
        # Finally plot one data file at a time
        plt.plot(temparray[:,0],temparray[:,1],'o',label='%s over %s' % (p,q))
        plt.plot(temparray[:,0],fittedcurve)
        ax = plt.gca()
        plt.xlabel(r'$Phase\ \phi\ (radians)$')
        ax.set_xticks([0,.5*np.pi,np.pi,1.5*np.pi,2*np.pi])
        ax.set_xticklabels(["$0$",r"$\frac{1}{2}\pi$",r"$\pi$",r"$\frac{3}{2}\pi$",r"$2\pi$"])
        plt.ylabel(r'$Average\ Velocity\ \langle v\rangle /v_r$')
        plt.title('Average velocity of multi-frequency ratchet')
        plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
        plt.show()
    #plt.show()
```

```
# Take that information from the tuples/datalist and plot them
dataarray = np.array(datalist)
# This is the max velocity vs. p/q graph
plt.stem(dataarray[:,0],dataarray[:,2],'-b',alpha=0.3,markerfmt='bo',basefmt='w')
for i in range(0,len(qarray)):
    plt.text(dataarray[i,0]-.05,dataarray[i,2]+0.3,r'$\frac{%s}{%s}$' % (parray[i],qarray[i]))
plt.ylim(0,8)
plt.title(r'$Velocity\ versus\ p/q\ ratio$')
plt.xlabel(r'$p/q$')
plt.ylabel(r'$Velocity\ v_{max}$')
plt.show()
# This is the max velocity vs. p*q graph
plt.plot(dataarray[:,1],dataarray[:,2],'bo')
#for i in range(0,len(qarray)):
    #plt.text(dataarray[i,1]+0.02,dataarray[i,2]+0.02,r'%s/%s' % (parray[i],qarray[i]))
plt.xlabel(r'$p*q$')
plt.ylabel(r'$Velocity\ v_{max}$')
plt.title(r'$Velocity\ versus\ p*q\ ratio$')
plt.show()

root.quit()

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Finally is DGating, beginning with DGating.f
program DGating

! Most recent as of 6/6/19 and finished
! No new edits besides commenting and documentation
```

```
! Welcome to the second iteration of the DGating program
! The previous one is based on old versions of code
! Rather than go through every single line fixing things,
! it's probably easier to just do this whole thing over

! So this is going to be based on DLattice rather than DBiharmonic
! If questions you have aren't answered here or the README, go ahead
! and check the DLattice documentation
! I've learned quite a few lessons recently that are talked about there

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Declaring Variables
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

implicit none
! This is the part where I regret every variable I've made
! Integers needed for loops
integer :: i,j
! Other integers
integer :: atom_num,tsteps,idum
! Now every other variable
! We have our inputs
real*8 :: GammaP_in,U_in,dt_in,driveratio,A,B,pqratio,phi
! Then calculated values from those inputs
real*8 :: GammaP,U0,dt,w_v,w_1,w_2,state
! We've got some convenience variables
real*8 :: ninth, cos2z,pi
```

```
! We've got our big three we take in our loops
real*8 :: p,z,t,p0,z0,t0,p_f,z_f,t_f
! And some other variables needed in the loops
real*8 :: kick,deltaP,rand,Dfsn,Fd,jumprate,gasdev
! And some things very unique to the gating ratchet
real*8 :: U0_t,Gamma_t
! Finally this whole mess of other things we need for data analysis
real*8 :: dispmean,vmean,dispsqrd,diff,v_sum,diff_sum
real*8 :: vmean2,diff2,v_sum2,diff_sum2,v_var,diff_var
real*8 :: v_std,diff_std,v_err,diff_err,peclet,v_avg,diff_avg

! And then we need our MPI and random number jargon
integer :: ierr,myid,numprocs,n
integer,allocatable :: seed(:)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! MPI
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Establish all the MPI stuff
! This allows parallel processing, though it adds some work for later
include 'mpif.h'

! Set up the variables we'll need to for the MPI stuff

! Now start the MPI function, give the ID of each process
call MPI_INIT (ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,numprocs,ierr)
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Reading in Variables
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Reading in from text files outside of the program
! This way we don't need to recompile every time we change variables

! All input files are in the 10-series, output in 20-series
! Every input variable is in its own row, with a label after
! Then we need to put each "read" operation on its own row as well
open(11,file='input.dat')
read(11,*) atom_num
read(11,*) tsteps
read(11,*) U_in
read(11,*) GammaP_in
read(11,*) phi
read(11,*) dt_in
read(11,*) driveratio
read(11,*) A
read(11,*) B
read(11,*) pqratio

! Remember that UNLIKE MultiFreq, A/B and pqratio are NOT related

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Random Number Generator
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Now we need a random number generator
```

```
! If we just feed the processors the same seed, they'll produce the
! same results for the simulation. This is bad.
! So intead, we are going to create seeds for the number generator that
! is based on a combination of time and processor ID. That way, the
! simulation I run tomorrow is different from today, and each process
! will be running its own simulation, rather than copying.

! Special note here!
! When you ask for the random_seed generator like this, it calls from the OS
! That means it's a little dependent on how good that RNG actually is
! Here, I'm trusting the OS to give me good random numbers
! If this is in doubt, it needs to be replaced by something else in a similar manner

call random_seed(size=n)
allocate(seed(n))
call random_seed(get=seed)

call random_seed(put=seed)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Fixing Values
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! We have now read in the values we need, let's fix them up
! Following the instructions of where to put the 1/2's

GammaP = GammaP_in * 0.5d0
U0 = U_in * 0.5d0
! Then we also scale the time step based on the scattering rate
```

```
dt = dt_in/GammaP
! And the driving frequencies, in a round-a-bout kind of way
! First is the vibrational frequency, w_v = 2sqrt(U0)*w_r
! Where w_r is still 1/2, so
w_v = sqrt(U0)
! Now we pull w_1 as a ratio from w_v
w_1 = w_v * driveratio
! And finally w_2 is pulled from w_1
w_2 = w_1 * pqratio

! Some other details we should establish here as well

! We'll start the initial state as 1, this doesn't really matter
state = 1.d0
! We acutally don't re-establish this each time because again,
! it doesn't really matter
! Rather than do the same division each time, name it a variable
ninth = 1.d0/9.d0

! And let's also set up a couple variables we'll use later
vmean = 0.d0
dispmean = 0.d0
dispsqrd = 0.d0

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! LOOPS
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
! There is no need for a potentials loop because we only use one
! Then we start with the atom loop
do i = 1,atom_num
! Set initial conditions for our new baby atom
! the gasdev actually doesn't give us realistic values for the start
! Not really a big deal as long as you give it long enough inside of
! the lattice
   p = gasdev(idum)
   z = gasdev(idum)
   t = 0.d0
   p0 = p
   z0 = z
   t0 = t
! And then initial state is just whatever state we had last

! Now we go into our time loop
   do j = 1,tsteps
! Now things get a bit more interesting
! Not only do we have this time-dependent driving force
! We ALSO have a time-dependent potential and gamma
      U0_t=U0*(1+A*dcos(w_1*t))
      Gamma_t=GammaP*(1+A*dcos(w_1*t))
! Now everything else looks pretty similar
! Less characters is easier, just call it a variable
      cos2z = dcos(2.d0*z)
! The diffusion term, now including a time-dependence
      Dfsn=(0.1d0*ninth*Gamma_t)*(35.d0+(state*7.d0*cos2z))
! The atom recieves a kick from this diffusion
```

```
      kick=((2.d0*Dfsn*dt)**(0.5d0))*gasdev(idum)
! The driving force acting here
      Fd = B * dcos(w_2*t + phi)
! And then put all of it together for a momentum change
      deltaP=kick+(state*U0_t*dsin(2.d0*z)*dt) + Fd*dt

! Now let's determine if we're going to jump or not
! First, the probability of that happening
      jumprate = ninth*dt*Gamma_t*(1.d0+(state*cos2z))
! And then the random number we compare it to
      call random_number(rand)
!  and compare them
      if (rand .lt. jumprate) then
! did we jump?
         state = -state
      end if
! And now we update our running variables
      p = p + deltaP
      z = z + p*dt
      t = t + dt

   end do ! end of the time loop

! Now we're not super concerned with the temperature
! Let's instead compare start and end states
   z_f = z - z0
   p_f = p - p0
   t_f = t - t0
```

```
! I don't know that I needed this anyways, but we have them
! Now let's look at our running averages
! We have three we're interested in
! First is the final displacement of the atom, averaged together
   dispmean = dispmean + z_f/atom_num
! Second is the overarching velocity, just total displacement over time
   vmean = vmean + z_f/(t_f*atom_num)
! Last is square of displacement, used for diffusion later
   dispsqrd = dispsqrd + (z_f * z_f)/atom_num

end do ! end of the atom loop
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Finalizing Data
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! So now we're outside of our computation, time to finalize everything
! First is to calculate diffusion
diff = (dispsqrd - (dispmean**2.d0))/(2.d0*t_f)
! Now we have a bunch of these variables that need putting together
! Specifically, we're looking towards vmean and diff
call MPI_REDUCE(vmean,v_sum,1,MPI_DOUBLE_PRECISION,
     & MPI_SUM,0,MPI_COMM_WORLD,ierr)
! Remember that this '0' says this value ONLY goes to processor 0
call MPI_REDUCE(diff,diff_sum,1,MPI_DOUBLE_PRECISION,
     & MPI_SUM,0,MPI_COMM_WORLD,ierr)
! We also want to figure out the squares of these values
! This is for the error calculation later
```

```
! Start by squaring those two values on EACH processor
vmean2 = vmean * vmean
diff2 = diff * diff
call MPI_REDUCE(vmean2,v_sum2,1,MPI_DOUBLE_PRECISION,
     & MPI_SUM,0,MPI_COMM_WORLD,ierr)
call MPI_REDUCE(diff2,diff_sum2,1,MPI_DOUBLE_PRECISION,
     & MPI_SUM,0,MPI_COMM_WORLD,ierr)

! And sure, we've summed those values, but they're not averages until..
v_avg = v_sum/numprocs
diff_avg = diff_sum/numprocs

! And then we can calculate standard error from that
! Only do this once, so:
if (myid.eq.0) then
! This is the variance in velocity
   v_var = (v_sum2-(numprocs*v_sum*v_sum))/(numprocs-1)
   diff_var=(diff_sum2-(numprocs*diff_sum*diff_sum))
     & /(numprocs-1)
! So then standard deviation is one more step away from variance
   v_std = dsqrt(v_var)
   diff_std = dsqrt(diff_var)
! And lastly we have the standard error we're looking for
   v_err = v_std/sqrt(dble(numprocs))
   diff_err = diff_std/sqrt(dble(numprocs))

! And the Peclet number?
   pi = 3.141592653589793d0
```

```
   peclet = v_avg*pi/diff_avg
end if
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Testing and Debugging
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Going through, checking these values, they both seem reasonable
!print *,p,z,t
!print *,p_f,z_f,t_f

! Next are the values directly derived from those
! Again, not zero, not immediate red flags
!print *,dispmean,vmean,dispsqrd

! Now we start looking into the "Finalizing Data" section
! This isn't true! --> wow these were all zero
! Remember that after the "reduce" function, only processor 0 has
!  the value stored
! Fixed! --> But diff is still zero for some reason
!print *,diff,v_sum,diff_sum
! These values are NEARLY zero if you only use one or two atoms
! Make sure to have just a small pool to make sure these averages work

! Step by step
! And these seem fine
!print *,v_sum,diff_sum,v_sum2,diff_sum2

! Towards the end here now
```

```
! Oh no
!print *,v_var,diff_var,v_std,diff_std,v_err,diff_err

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Data Writing
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! First, reminder that all ID's for inputs are 10's, 20's for output
! Matching the outputs of DBiharmonic and DMultiFreq
! There's actually quite a bit of information here
! Not all of it is picked up by DataCollector
! Consider changing this in the future?

! We only want this written once, so only do one processor
if (myid.eq.0) then
   open(unit=21,file='output.dat',status='replace')
! First is a list of the input variables and time spent
   write(21,*) 'dt = ',dt
   write(21,*) 'U0/Er = ',U_in
   write(21,*) 'time spent = ',t_f
   write(21,*) 'w_1 = ',w_1
   write(21,*) 'w_2 = ',w_2
   write(21,*) 'phi = ',phi
   write(21,*) 'Amplitude(A) = ',A
   write(21,*) 'Force(B) = ',B
   write(21,*) 'Processors: ',numprocs
! And next is the data we actually want
   write(21,*) 'Output Data:'
```

```
   write(21,*) '<v> = ',v_avg
   write(21,*) 'v_err = ',v_err
   write(21,*) 'Diff = ',diff_avg
   write(21,*) 'Diff_err = ',diff_err
   write(21,*) 'Peclet = ',peclet
end if

call MPI_FINALIZE(ierr)
end program

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Algorithms to generate random numbers(Ref. W.H.Press et Al.         !
! "NUMERICAL RECIPES".Cambridge U.P. 1986)                           !
! Function GASDEV generates a Gaussian Distribution                  !
! of zero mean and variance unity.(Box-Mueller formula)              !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      FUNCTION GASDEV(IDUM)
      IMPLICIT REAL*8 (A-H,O-Z)
      IMPLICIT INTEGER*4 (I-N)
      REAL*8 GASDEV,X(2)
      SAVE ISET,GSET
      DATA ISET/0/
      IF(ISET.EQ.0) THEN
 1       CALL RANDOM_NUMBER(X)
         V1=2.0d0*X(1)-1.d0
         V2=2.0d0*X(2)-1.d0
         R=V1**2+V2**2
      IF(R.GE.1.0d0) GO TO 1
```

```
            FAC=DSQRT(-2.0d0*DLOG(R)/R)
            GSET=V1*FAC
            GASDEV=V2*FAC
            ISET=1
         ELSE
            GASDEV=GSET
            ISET=0
         ENDIF
         RETURN
         END

! Move on to the bash scripts that run it, start with GatingShell.sh
#!/bin/bash

echo "Welcome to the first DGating script!"
echo "Brown's thesis only looks at p/q, and we follow suit here"
echo "I will need a P.txt and a Q.txt to work"

#starttime='date +%s'
# First we read the text files into arrays
readarray Pall < ./P.txt
readarray Qall < ./Q.txt
# q is just for simple indexing
echo ${Pall[@]}
echo ${Qall[@]}
q=0
# Cycle through every value of Pall
for p in ${Pall[@]}; do
```

```
# For some infuriating reason, the readarray also saves spaces after the numbers
# For most functions, this is just fine
# For bc, the world is ending
# A quick division by 1 (since they're integers) will fix this
tempP=$(($p/1))
tempQ=${Qall[$q]}
tempQ=$(($tempQ/1))
q=$(($q+1))
# "scale" here decides how many digits the ratio will keep
ratio=$(bc <<< "scale=5;($tempP/$tempQ)")
# Now we have the ratio that we'll need to save
awk -v ratio=$ratio '{if($3=="pqratio") {print ratio" ! pqratio"} else {print $0}}' GatingInStart > ./GatingIn0

# GFortranRunner uses 40 phi points
./GFortranRunner.sh
./DataCollector.sh > ${tempP}over${tempQ}pqratio.dat

echo "I just finished ${tempP} over ${tempQ}"
done
#endtime='date +%s'
echo "All done!"
#runtime=$((endtime-starttime))
#echo "I took ${runtime} seconds to run"

! Then GFortranRunner.sh
#!/bin/bash

# GFortran uses 40 phi values instead of just 20
```

```
PHI="    0.0000000000000 0.15707963267948966 0.3141592653589793 "

for phi in $PHI; do
    mydir='PHI_'$phi
    mkdir $mydir
    awk -v phi=$phi '{if($3 == "phi:") {print phi" ! phi: phase difference"} else {print $0}}' GatingIn0 > $mydir/input.dat
    cd $mydir
    mpirun -n 8 ../DGating.exe
    cd ..
done

! And DataCollector.sh
dirs=$(ls -d PHI_*)

echo "# phi ; <v> ; error <v> ; diff ; error diff ; Peclet"

for dir in $dirs ; do

cat $dir/output.dat | awk '{if($1=="phi") {phi=$3} if($1=="<v>") {v=$3 ; getline ; dv=$2 ; getline ; \
d=$3 ; getline ; dd=$2 ; getline ; pe=$3 }}END{print phi"   "v"   "dv"    "d"     "dd"    "pe}'

done

! And PQ_Plotter.py is again used for the gating ratchet
```

Bibliography

[1] Peter Hänggi and Fabio Marchesoni. Artificial brownian motors: controlling transport on the nanoscale. *Reviews of Modern Physics, Vol. 81, Jan-Mar*, 2009.

[2] Martin Brown. *Monte Carlo simulations of cold atom ratchets*. PhD thesis, University College London, 2008.

[3] E. Klinman and E. Holzbaur. Walking forward with kinesin. *Trends in Neurosciences Vol. 41, No. 9*, 2018.

[4] B. Lewandowski, G. De Bo, J. W. Ward, M. Papmeyer, S. Kuschel, M. J. Aldequnde, P. M. Gramlich, D. Heckmann, S. M. Goldup, D. M. D'Souza, A. E. Fernandes, and D. A. Leigh. Sequence-specific peptide synthesis by an artificial small-molecule machine. *Science 339 (6116)*, 2013.

[5] J. Bang et al. Experimental realization of feynman's ratchet. *New J. Phys. 20 103032*, 2018.

[6] R. Feynman, R. B. Leighton, M. Sands, M. Gottlieb, and R. Leighton. *The Feynman Lectures on Physics*. Addison-Wesley, 1964.

[7] David Cubero and Ferruccio Renzoni. *Brownian Ratchets*. Cambridge University Press, 2016.

[8] J. Dalibard and C. Cohen-Tannoudji. Laser cooling below the doppler limit by polarization gradients: simple theoretical models. *J. Opt. Soc. Am. B Vol. 6, No. 11*, 1989.

[9] J. Dalibard, Y. Castin, and K. Mølmer. Wave-function approach to dissipative processes in quantum optics. *PRL 68(5):580-583*, 1992.

[10] R. Kosloff. Time-dependent quantum-mechanical methods for molecular-dynamics. *Eur. Phys. L. 44(3):341-347*, 1988.

[11] J. Dalibard and C. Cohen-Tannoudji. Atomic motion in laser light: connection between semiclassical and quantum descriptions. *J. Phys. B: At. Mol. Phys. 18 1661*, 1985.

[12] K. I. Petsas, G. Grynberg, and J.-Y. Courtois. Semiclassical monte carlo approaches for realistic atoms in optical lattices. *Eur. Phys. J. D 6, 29-47*, 1999.

[13] Daniel A. Steck. Rubidium 85 d line data, December 2010.

[14] M. Brown and F. Renzoni. Ratchet effect in an optical lattice with biharmonic driving: A numerical analysis. *Phys. Rev. A 77, 033405*, 2008.

[15] R. Gommers, M. Brown, and F. Renzoni. Symmetry and transport in a cold atom ratchet with multifrequency driving. *Phys. Rev. A 75, 053406*, 2007.

[16] R. Gommers, V. Lebedev, M. Brown, and F. Renzoni. Gating ratchet for cold atoms. *PRL 100, 040603*, 2008.

[17] Ralf Gommers. *Symmetry and transport in cold atom ratchets*. PhD thesis, University College London, 2007.

[18] Ajithamithra Dharmasiri. Preliminary observation of vibrational resonances and propagation modes in cold atom dissipative 3d optical lattices. Master's thesis, Miami University, 2019.

[19] K. I. Petsas, A. B. Coates, and G. Grynberg. Crystallography of optical lattices. *Phys. Rev. A Vol. 50 No. 6*, 1994.

[20] V. Lebedev and F. Renzoni. Two-dimensional rocking ratchet for cold atoms. *Phys. Rev. A 80, 023422*, 2009.

[21] J. Dalibard et al. Potentialities of a new $\sigma_+ - \sigma_-$ laser configuration for radiative cooling and trapping. *J. Phys. B: At. Mol. Phys. 17 4577*, 1984.