# Computational Physics

Spring Semester 2014

Final Reports

PHYS 401
PHYS 618


Texas A&M University

# Power-Law Behavior of the Bak-Tang-Weisenfeld Sandpile Model

Chris Akers
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 24, 2014)

We study the self-organized criticality (SOC) of the famous Bak-Tang-Weisenfeld (BWT) sandpile model. As the simplest known model that exhibits SOC, understanding this model is a significant first step to understanding many natural systems that exhibit SOC, such as earthquakes and rainfall. First, we simulate the typical, widely-used sandpile model and observe that it does support the hypothesis that this model shows perfect SOC in the limit of an infinite system. Second, we modify the sandpile to mitigate some of the finite-sized effects, and we find that model's results also indicate that an infinite BTW sandpile exhibits SOC.

## I. INTRODUCTION

Many natural systems exhibit what has been termed "fractal structure." Here, fractal structure is taken to mean that some correlation functions for that system show non-trivial power-law behavior. For example, in mountain ranges, the height profile can be characterized by how $\Delta h(R)$, the relative difference in height between two points separated by distance $R$, varies with $R$ [1]. It happens to follow the power-law equation:

$$\left\langle [\Delta h(R)]^2 \right\rangle \propto R^{-x}. \tag{1}$$

Here $< >$ denotes averaging over different spatial points at a fixed horizontal separation R. The parameter x is some non-trivial exponent.

Another natural system that exhibits such behavior is earthquakes. Guttenberg and Richter were the first to discover that the statistical distribution of earthquakes follows a power-law [2]. The frequency of earthquakes of total energy E is found to vary as $E^{-z}$, where z is approximately 2 [3]. Likewise, it has recently been shown that the frequency of rain of a given intensity also follows a power-law (for roughly five orders of magnitude) [1].

A firm understanding of fractal structures thus might allow us to calculate the critical exponents of these important natural phenomena, and not simply search for them experimentally. This is our motivation for developing a simplified, tractable model that exhibits the same long-ranged correlations with power-law decay.

Note that systems possessing such power-law correlations are said to have critical correlations. The term critical correlaion is used for historical reasons, because these types of correlations were first studied in equilibrium statistical mechanics near some critical phase transition. Historically, in order to observe such critical phenomena in equilibrium systems, one would need to fine-tune at least one physical parameter (such as temperature or pressure) to a specific critical value.

However, studying such finely tuned systems provides little insight into natural processes (such as mountain range formation and earthquake frequency), because natural processes are rarely so fine-tuned. In the case of mountain ranges, for example, the actual height distribution is a result of many factors (including plate techtonics and erosion), and the temperature has fluctuated greatly over the formation time. Thus, for more physical relevance to these natural phenomena, our model should not need fine-tuning.

In their famous paper, Bak, Tang, and Weisenfeld (BTW) proposed a model that had all of these desired properties. It was simple, showed power-law correlations, and naturally evolved to its critical state (where these power-law correlations arise). That is to say, their model was simple and showed self-organized criticality (SOC). These two traits make it a useful tool for studying fractal structures in nature.

Their model was a sandpile. The motivation for the model goes something like the following. If one drops sand onto a two-dimensional table, the sand will pile up, forming a sort of cone. However, there is a maximum steepness that this sandpile can have. We will say that there is a critical angle $\theta_c$. Once the pile reaches this $\theta_c$ nearly everywhere and one tries to add more sand, the added sand will cause an avalanche, bringing the sand to some other part of the pile that is not too steep. In this way, sandpiles naturally evolve from less than $\theta_c$ everywhere to being $\theta_c$ everywhere. Once they have reached the maximum steepness, they are roughly stabilized there. Their behavior at this critical angle is characterized by stretches of seemingly no response to additional sand interspersed by avalanches that range in size from a few grains to affecting the entire system.

To again highlight the similarity between this model and nature, earthquakes exhibit similar build-up/relax behavior. For earthquakes, stress builds up due to techtonic motion gradually, but is released in bursts of various sizes [3].

The focus of this paper is to explore the interesting properties of the BTW sandpile model. Interestingly, even though it is widely believed that the BTW model shows SOC in the limit of infinite system size, it is actually an open problem to prove that this is the case. Thus, in particular, we will illustrate that this sandpile model indeed exhibits SOC (i.e. naturally evolves to a state that shows power-law correlations). We do this by numerically simulating a simplified sandpile model and studying its properties. We also explore methods for im-

proving this sandpile model to better exhibit power-law behavior.

## II. MODEL

The sandpile model we use obeys the following rules. The simulation occurs on a two-dimensional square lattice of side-length L. Thus there are $N = L^2$ total sites in this latice. We begin the simulation with a flat surface $Z(x, y)$. Here $Z$ indicates how many grains are present at the site given by the coordinates x and y. Every time step we add one grain to a random site, where all sites in the lattice are equally probable for selection.

$$Z(x, y) = Z(x, y) + 1 \tag{2}$$

Equation (2) is the driving force in this model. After we have dropped this grain, we check that the number of grains in this cell obeys $Z(x, y) < 4$. If it does, then we continue to the next time step (selecting a random site and dropping another grain). However, if that site satisfies the instability rule equation (3), then that site topples.

$$Z(x, y) \geq 4 \tag{3}$$

When a site topples, it loses four grains, and each of its four neighbors gain one grain.

$$\begin{aligned} Z(x, y) &= Z(x, y) - 4 \\ Z(x \pm 1, y) &= Z(x \pm 1, y) + 1 \\ Z(x, y \pm 1) &= Z(x, y \pm 1) + 1 \end{aligned} \tag{4}$$

After the topple occurs, one or more of the neighbors might now be unstable from the additional sand. Each unstable site then topples in the same way, obeying equation (4). This proceeds until no sites are unstable. Note that if any sites on the edge of the lattice topple, some sand will "fall off the edge," leaving the system. For example, if a site on the left edge topples, it will lose four grains and each of its three neighbors will gain one grain. The fourth grain is lost.

When we add a grain and at least one topple results, that is called an avalanche. We speak of the size of an avalanche (s) as how many topples occured. So if only one topple occured, s = 1 for that avalanche. If dropping the grain caused a chain reaction in which 120 sites toppled, s = 120. Note that if one site topples twice or more in the same avalanche, both topples do count towards the avalanche size.

Here we supply an example of an avalanche. Suppose that our lattice size is 4x4. Also suppose that, after a few time steps, the lattice configuration looks like:

| | | | |
|---|---|---|---|
| 1 | 0 | 3 | 2 |
| 3 | 1 | 3 | 3 |
| 3 | 2 | 2 | 1 |
| 2 | 1 | 3 | 0 |

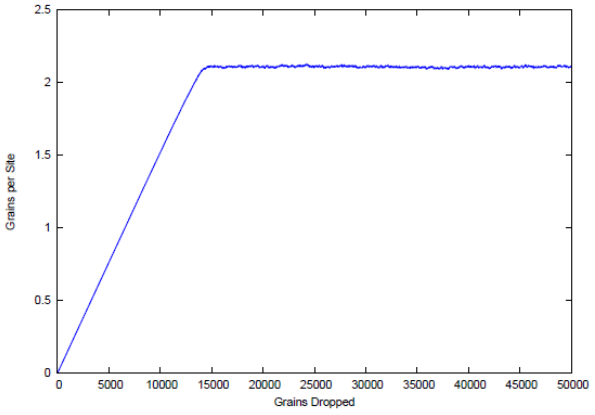



FIG. 1: BTW Sandpile simulation on a 81x81 lattice. As we drop grains onto the lattice, the average grains per site increases linearly until around 13,000 grains have been dropped. Then the lattice settles at roughly 2.1 grains per site, at which point it is said to have reached it critical state.

Now suppose that we advance this configuration one time step that adds a grain to the site second from the right and second from the top. That site will have four grains and thus topple.

| | | | |
|---|---|---|---|
| 1 | 0 | 3 | 2 |
| 3 | 1 | 4 | 3 |
| 3 | 2 | 2 | 1 |
| 2 | 1 | 3 | 0 |

→

| | | | |
|---|---|---|---|
| 1 | 0 | 4 | 2 |
| 3 | 2 | 0 | 4 |
| 3 | 2 | 3 | 1 |
| 2 | 1 | 3 | 0 |

Two of the neighbors are now themselves unstable, and thus topple. This results in further toppling.

| | | | |
|---|---|---|---|
| 1 | 1 | 0 | 4 |
| 3 | 1 | 2 | 0 |
| 3 | 2 | 2 | 2 |
| 2 | 1 | 3 | 0 |

→

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 3 | 1 | 2 | 1 |
| 3 | 2 | 2 | 2 |
| 2 | 1 | 3 | 0 |

Finally, the lattice has arrived at a stable state, where no further topplings need to occur. This avalanche was size s = 4, because four total topplings occured.

That is all there is to this model. All of its complex, interesting properties arise from these simple rules that we laid out above. One such interesting property is that the lattice will stabilize itself at around 2.1 average grains per site, as illustrated in figures 1 and 2.

This behavior is readily explained by the sandpile exhibiting SOC: the sandpile naturally evolves to the state of roughly 2.1 grains per site, which is its critical region, at which point avalanches of all different sizes occur sporadically. Since it is this critical region of SOC that we are interested in, we will be sure to sample statistics from our model only after it has reached its critical state.

However, note that this natural evolution to a stable

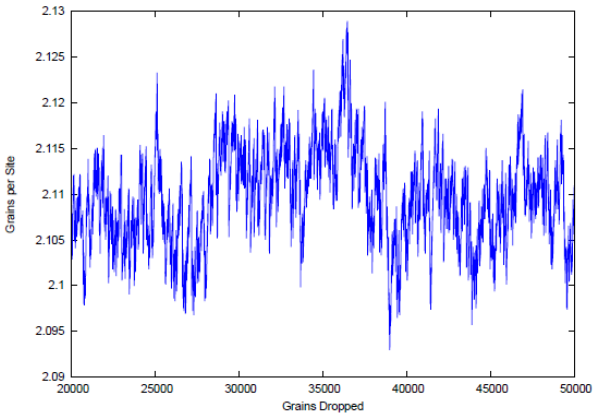



FIG. 2: Zoomed in plot of the average grains per site as a function of grains added. Note the existence of deviations from the equilibrium value. These fluctuations reflect the nature of the critical state, where grains pile up steadily and avalanche sporadically.

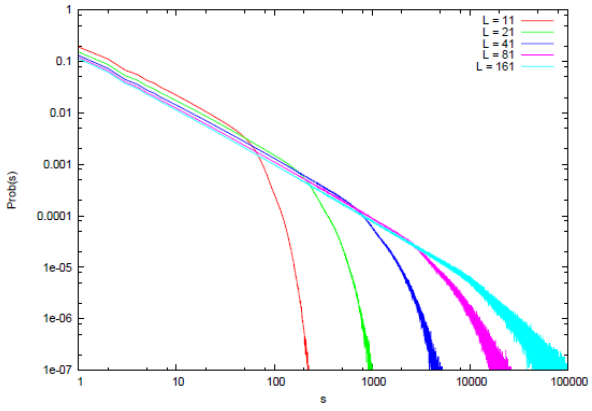



FIG. 3: Log-log plot of the probability of avalanche size. Larger system size correlates with a longer region of power-law behavior.

state is not sufficient for SOC. Indeed, the defining feature of systems with SOC is that they exhibit some sort of power-law correlation in their stable state. It is widely believed that this sandpile model shows such a power-law correlation with respect to the size of its avalanches. That is, it is thought that equation (5) holds, where s is avalanche size and P(s) is the probability of the resulting avalanche being that size given that we place a grain on a site and that site topples.

$$P(s) = cs^{-a} \tag{5}$$

Here c and a are some constants. It will be our ultimate goal to numerically simulate this sandpile model and inspect whether its avalanches really are power-law distributed.

Here, we will analytically argue why the avalanche sizes are expected to follow equation (5). First, note that the expected avalanche size will be of the order of the system size. That is,

$$\langle s \rangle \equiv \sum sP_L(s) \geq cL \tag{6}$$

where L is the length of system and c is some scaling constant. If we take our system size to infinity, $L \to \infty$, and we defiine

$$P_\infty(s) = \lim_{L\to\infty} P_L(s) \tag{7}$$

then its straightforward to see that

$$\langle s \rangle \equiv \int sP_\infty(s)ds = \infty. \tag{8}$$

Such a divergent expectation value can only be generated by probability distributions that fall off more slowly than $s^{-2}$. Furthermore, for our probability distribution to be normalized,

$$\int P_\infty(s)ds = 1. \tag{9}$$

Therefore our probability distribution must converge to zero more quickly than $s^{-1}$. With these two constraints, it is reasonable to assume that our probability distribution takes the form of equation (5), where $1 < a < 2$.

Note that this logic inherently utilizes the fact that we are interested in infinite systems. This is an issue because we can only simulate finite systems. To work around this issue, we assume that finite sized systems will show the correct (power-law) behavior for avalanches that are small compared to their system size, and will show finite-sized effects for larger avalanches. Therefore we hope to show that increasing the system size causes our results to converge to the desired power-law form.

## III. RESULTS

The first test simulation we ran was the typical one described in the methods section. The probability of each avalanche size was plotted for different system sizes in figure 3 below. Note that distributions of the form $P = cs^{-a}$ appear visually as lines on log-log plots.

Figure 3 supports our expectation that finite-sized sandpiles have power-law distributions for avalanches that are small compared to the system size. However, the probability of avalanches of order system size squared (the total number of elements) drops to zero rapidly. This is expected, since avalanches that are larger than the number of sites are extremely rare. They require some number of sites to topple more than once, which is relatively difficult to make happen.

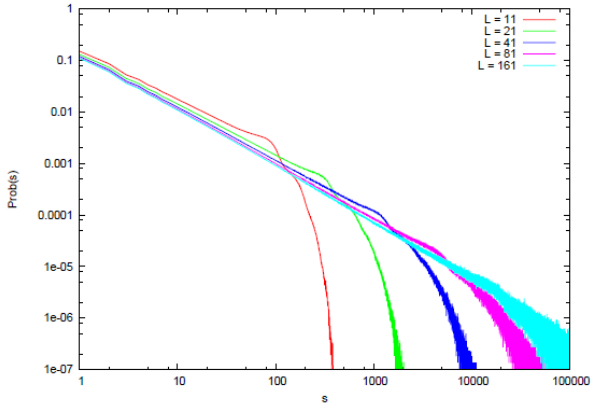



FIG. 4: Log-log plot of the probability of avalanche size for the modified sandpile. As before, larger system size correlates with a longer region of power-law behavior. Note here that the regions of power-law behavior are longer than those in figure 3, so the finite sized effects have indeed been reduced slightly.

In order to improve our statistics and further reduce the finite-sized effects, we modified the standard BTW sandpile model. In particular, we minimize the effects of avalanches starting near the edge. Such avalanches have different statistics from those that start in the bulk, and in an infinitely sized system, all avalanches start in the bulk. Avalanches that start near corners have an amplified version of this problem.

Thus, first, we eliminate corners by closing the top boundary to the bottom boundary. Note that we cannot simultaneously close the left and right boundaries because then the sand would have no way to escape the lattice (and avalanche size would quickly diverge). Second, on even grain drops we can place grains anywhere on the lattice (as normal). However, on odd grain drops we only place grains along the middle-most column (where everywhere along the column is equally likely). Additionally, we only consider avalanches that are started by odd grain drops. In this way, we only count avalanches that start with significant space on all sides. Note that choosing which avalanches we will count does not affect the fact that the distribution should be power-law, since the argument made with equations (6) through (9) still holds.

With these adjustments, the distributions change to those in figure 4. An interesting feature of figure 4 is the distortions that occur in the plots as the power-law region shifts to the more-rapidly falling region. These distortions arise from the finite-sized effects of these closed y-bounderies, and are due to the statistical correlations these closed boundaries cause in avalanches that span the entire system. They do not affect our statistics, since these distortions occur around the rapid fall-off, and we are not interested in that region to begin with. Moreover, the largest system sizes (L = 81 and 161) do not show significant distortions, further illustrating that the distortions are only finite-sized effects and not real errors in this modification of the model.

TABLE I: The critical exponent $a$ from equation (5) for both the standard BTW and modified BTW sandpiles plotted in figures 3 and 4.

| L | Standard | Modified |
|---|---|---|
| 11 | $0.9914 \pm 0.0133$ | $0.9302 \pm 0.0086$ |
| 21 | $1.0278 \pm 0.0023$ | $0.9892 \pm 0.0006$ |
| 41 | $1.0218 \pm 0.0010$ | $1.0258 \pm 0.0007$ |
| 81 | $1.0348 \pm 0.0004$ | $1.0463 \pm 0.0004$ |
| 161 | $1.0500 \pm 0.0004$ | $1.0633 \pm 0.0004$ |

We used gnuplot to interpolate the data in figure 4 to functions of the form (5) in order to find the value of the critical exponent $a$. We only used the avalanche sizes that were clearly in the power-law region for this interpolation, so as not to bias the functions with the non-power-law fall off. For example, in the case of $L = 161$, we interpolated from $s = 1$ to $s = 2000$. For $L = 81$, we interpolated from $s = 1$ to $s = 1000$.

Table 1 tabulates the values of $a$ we found for the different system sizes and sandpiles.

Finally, we return to the question "Does the BTW sandpile show SOC in the limit of infinite system size?" To answer this question with our results, we need to employ finite-size analysis. That is, we need to carefully examine how our data is changing with system size, and determine if it is actually correct to assume that, in the infinite limit, the sandpile shows a perfect power-law.

As is typical for this type of finite-size analysis, we make the claim that there are two "components" to figures 3 and 4. One is the power law component, responsible for the linear region of the graphs. The other is the exponential-damping component, which is responsible for the quick fall-off after the linear region. We can assume this exponential- damping component is of the form $D(s) \propto exp[-s/s_*(L)]$, where $s_*(L)$ is the characteristic length of the damping term for a given system length L. A qualitative inspection of figures 3 and 4 suggests that $s_*(L)$ increases monotonically with L (without reaching any sort of asymptote). If we can show this quantitatively, we will have shown that our results support the conclusion that the BTW Sandpile exhibits SOC.

First, we fit the exponential regions of figures 3 and 4 to functions of the form $D(s) \propto exp[-s/s_*(L)]$. We make sure to start the range of our fit where the linear fit ended. For example, for L = 81 we fit a line from $s = 1$ to $s = 1000$. Now, we fit the dying exponential from $s = 1000$ to $s = 30000$. (We choose that upper bound because $P(s)$ vanishes there). Second, we compile and examine the resulting function $s_*(L)$. Our results are in table II.

From table II, we can see that $s_*(L)$ diverges with $L$. Thus taking $L$ to infinity also sets the characteristic length of the exponential-damping to infinity. This implies that, in the infinite size limit, the probability dis-

TABLE II: The values of $s_*(L)$ found from fitting $D(s) \propto exp[-s/s_*(L)]$ to the exponentially-damped regions of P(s) in figures 3 and 4.

| L | Standard | Modified |
|---|---|---|
| 11 | $20.56 \pm 0.25$ | $39.20 \pm 0.95$ |
| 21 | $100.00 \pm 0.14$ | $203.48 \pm 0.61$ |
| 41 | $500.00 \pm 1.23$ | $1000.00 \pm 3.579$ |
| 81 | $2000.00 \pm 3.14$ | $5000.00 \pm 18.73$ |
| 161 | $10000.00 \pm 11.96$ | $15000.00 \pm 19.04$ |

tribution is a perfect power-law.

## IV. SUMMARY AND CONCLUSIONS

We have seen that the finitely-sized BTW sandpiles exhibit power-law distributions of avalanche sizes to a certain size. After that size, the distribution is damped exponentially. We have also seen that increasingly large sandpiles exhibit power-law distributions to increasingly large cutoffs before the damping dominates. When we analyzed how the avalanche probability distribution might look in the infinite system size limit, we found that the characteristic damping inherent to finite-sized systems disappears. This supports the hypothesis that the infinite BTW sandpile model shows a purely power-law distribution of avalanche sizes (and thus exhibits SOC).

Furthermore, we found increasingly better approximate values for the critical exponent in table I. Our best estimates were around $a = 1.05$, which both lies in our desired range ($1 < a < 2$) and is the currently accepted value [4]. Unfortunately, this value is much lower than that of earthquakes and other natural phenomena. This means, mathematically, that large events are relatively more likely in the sandpile than they are in natural phenomena. This does limit the physical relevance of the BTW sandpile, but it is still useful as the arguably simplest model that exhibits SOC. Many have designed improvements to the sandpile that increase its critical exponent without removing its SOC. It would be wise to study those in future research.

### Acknowledgments

We thank Zheng Zhu for his guidance during this project and Nick Amin for his constant moral support.

[1] D. Dhar, *Theoretical studies of self-organized criticality*, Physica A **369**, 29 (2006).

[2] B. Gutenberg and C. F. Richter, *Seismicity of the Earth and Associated Phenomenon* (Princeton University Press, 1954).

[3] C. Frohlich and S. C. Davis, *Teleseismic b values; or, much ado about 1.0.*, J. Geophys. Res. **98**, 631 (1993).

[4] C. Zehetner, *Gradient Based Bak-Tang-Wiesenfeld Sandpile Model*, pp. 1–7 (2010).

# Percolation Transitions in Two Space Dimensions

Michael T. Allen
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 17, 2014)

In this paper we will investigate percolation phase transitions in two dimensional space using numerical simulations and finite-size scaling analysis. For the purposes of this study we will only consider site percolation for square lattices. The program used to do the numerical simulations implements the Leath algorithm to grow the clusters, and the Hoshen-Kopelman algorithm for labeling the clusters to determine if the system percolates. To complete the study, simulations were performed on a number of different (finite) size lattices and compared using finite-size scaling to determine the optimal value of the percolation threshold.

## I. INTRODUCTION

In mathematics, percolation theory describes the behavior of connected clusters in a random graph. Percolation theory has many practical applications in nature stretching from modeling oil seepage to forest fires. The general problem goes as such: If a liquid is poured on top of some porous material, will the liquid be able to make its way through the pores from top to bottom? This problem can be modeled in one of two ways, using bond percolation or site percolation. For bond percolation we would create a graph of $n$ x $n$ verticies (for the 2D case with a square lattice, the problem can be done for any dimension or lattice shape) where each edge between verticies can either be open with a probability $p$, or closed with a probability 1 - $p$. For site percolation we would create the same $n$ x $n$ graph, except this time each verticie (site) can be open with a probability $p$ or closed with a probability 1 - $p$. The question to be answered for both of these scenarios would be: for a given $p$, what is the probability that an open path exists from top to bottom? [1, 2] Figure 1 shows the difference between the two. In this study we will only be considering site percolation on square lattices.

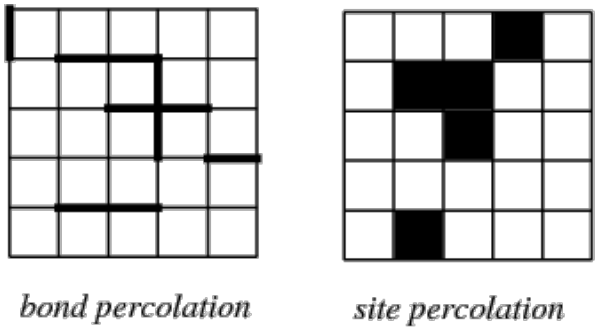



FIG. 1: These pictures show the difference between bond and site percolation. Bond percolation considers the lattice edges as the relevant entities, while site percolation considers the lattice verticies as the relevant entites [2].

## II. PERCOLATION THEORY

The section presented below is a small background on percolation theory following the work presented reference [3].

If we take a square grid and occupy sites with a probability $p$ we will notice that for small values of $p$ the system we will have mostly isolated occupied sites. As we increase the value of $p$ we will start to see isolated clusters emerge (if two neighboring sites are occupied we call them a cluster). As the value of $p$ continues to increase these clusters will begin to merge and at a certain value of the occupance probability, called the percolation threshold ($p_c$), one cluster will begin to dominate and span the system. Above the value $p_c$ all the other clusters will be absorbed into the dominating cluster until $p$=1 where every site on the lattice is occupied. The value $p_c$ depends heavily on the dimensionality of the problem and the shape of the lattice. For an infinite lattice if $p<p_c$ the system will never percolate, and if $p>p_c$ then the system will always percolate.

This behavior where the system goes through a sudden change at a certain value is known as a phase transition. Phase transitions are common in other branches of physics. For example a magnet can lose its magnetization when heated to a certain temperature, or ice melting into water at a specific temperature. For non-infinite systems this step function behavior gets "smeared" out into a continuious function. The smaller the system the more "smeared" the behavior gets. Extracting the information about the infinite system from finite systems will be discussed in a later section. It has been shown that properties of these systems close to the phase transition can be described in very simple terms.

For values of $p$ greater than $p_c$ but less than 1 not all occupied sites are in the infinite (spanning) cluster. We define a function $P(p)$ to be the probability that an occupied site is in the spanning cluster. For values of $p<p_c$ $P(p)$ is obviously 0, because there is no spanning cluster, but for values of $p>p_c$ $P(p)$ can be described in analytical terms.

$$P(p)\sim(p-p_c)^{\beta} \tag{1}$$

This relation is known as a power law or scaling law and the exponent $\beta$ is known as a critical exponent. An

extreamly useful feature of these scaling laws is that it only depends on the dimensionality of the space being examined, and is independent of lattice shape or whether it is site or bond percolation. This property is known as universality. A consequence of universality is that the large scale behaviours of these sytems can be described by mathematical relationships which are independent of small scale construction. This means we can study a wide variety of systems without needing to worry about the details of each individual system. An important point to note is that while the scaling laws and critical exponents posses the property of universality, the percolation threshold does not. The percolation threshold is extremely dependent on the shape of the lattice and the style of percolation (bond or site). For a two dimensional system $\beta$ =5/36. There are many other critical exponents that can be defined which describe the properties of a system at or near the percolation threshold, but the discussion of these is outside the scope of this study.

Here we will describe one more critical exponent that is useful to the application described above. First we define a two point correlation function $g(r)$ as the probability that if one point is in a cluster then another point a distance $r$ away is in the same cluster. This function typically has an exponential decay given by a correlation length ($\xi$).

$$g(r) \sim e^{-r/\xi} \tag{2}$$

For low values of $p$ the correlation length is small (clusters typically have a size of one or two), it will continue to increase with $p$ until the percolation threshold, when the spanning cluster dominates. The spanning cluster is infinite in size (for an infinite lattice), so the cluster size will diverge. To examine values of $p$ greater than the percolation threshold we must remove the spanning cluster from our caluclations or it will dominate and be the only cluster considered. As $p$ increases above $p_c$ more clusters get absorbed into the spanning cluster causing the "typical" size of the remaining clusters to decrease. This behavior of increasing until the threshold, where it diverges, then decreasing can be described mathematically as

$$\xi \sim (\mid p - p_c \mid)^{-\nu} \tag{3}$$

where $\nu$ is another critical exponent with the value of 4/3 for two dimensions. As with the connectivity exponent ($\beta$) it has the property of universality.

So far we have only considered an infinite lattice in our discussion. What happens if the lattice is finite?

## III. FINITE SIZE SCALING

The material in the following section follows the sections with the same names in reference [3].

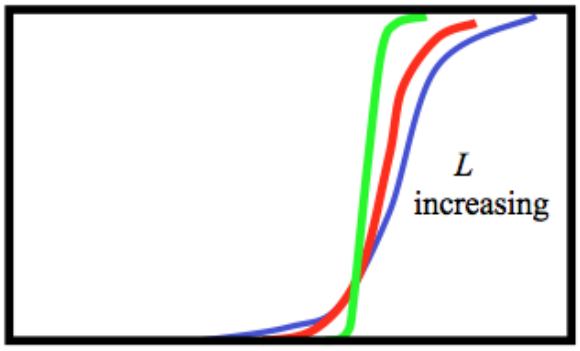



FIG. 2: [3]This image shows that as $L$ increases the "smearing" of the sharp transition is decreased.

The problem of how to deal with a finite lattice is known as finite size scaling. Consider a square lattice of size $L$. If we calculate the probability $P(p)$ that the system will percolate as a function of the occupancy probability $p$ we will find that we no longer get a sharp transition like expected in the infinite case. Instead, the sharp transition gets "smeared out". The amount of "smearing" that occurs is inversely proportional to the size of the system $L$ (Figure 2).

This phenomenon is similar to other thermodynamic phase transition where small systems have "smeared" transition. We can describe this smearing in simple mathemeatical terms. First we look at the two length scales of the problem: the system length and the correlation length. If the system length is much larger than the correlation length the clusters dont notice the finite boundaries and the system behaves like an infinite system. Conversly when the cluster size "sees" the boundaries a new behavior must be introduced. For this reason the important parameter must be the dimensionless ratio of these two lengths $\xi/L$. Then for a given system size $L$ the probability takes the form

$$P(p, L) = f_L[(p - p_c)L^{1/\nu}] \tag{4}$$

where $f_L$ is some function. Now consider how the behavior of the system changes under an arbitrary change in size. Let $L \to kL$. Under this change of scale we expect the essetial behavior to remain the same, that is $P(p, k, L) \to c(k)P(p, L)$. The only function that has this property is the power law, so we can write

$$P(p, L) = L^A F[(p - p_c)L^{1/\nu}] \tag{5}$$

where F is a universal function and A is an exponent to be determined. To determine A we must consider the asymptotic behavior of P. As L $\to \infty$ we must obtain the critical law for the infinite system. Therefore $F[z] \to z^\beta$ for large $L$. To get rid of the $L$ dependence we must have

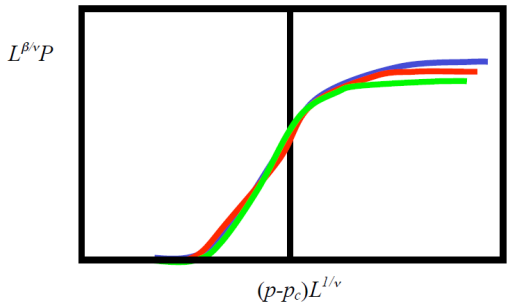



FIG. 3: [3]This image shows scaled curves for different values of $L$ all falling on one universal curve.

$A = -\beta/\nu$. Therefore the finite size scaling law can be written as

$$P(p, L) = L^{-\beta/\nu} F[(p - p_c)L^{1/\nu}] \tag{6}$$

If we plot $L^{\beta/\nu}P$ against $(p-p_c)L^{1/\nu}$ for several values of $L$ all the curves should lie on top of each other to form a single universal curve(Figure 3). This is of great use, as we can now determine the behavior of a system we have not simulated from the behavior of the universal curve.

## IV. NUMERICAL METHODS

To find the percolation threshold $p_c$ for a two dimensional square lattice numerical simulations were used. The program used to perform the simulations would grow a large amount of clusters for a given occupation probability $p$, then determine what percent of these clusters percolated. This would be repeated several times for a range of occupation probabilites and several system sizes, and then averaged to determine errors in the data. These averages, along with their error bars, would then be plotted as a graph of Percolation Probability ($P$) as a function of occupation probability ($p$). Once these were plotted the region around the point where the curves intesetect would be noted. Then the procedure would be repeated for a smaller range of values for $p$ (around the intersection point )with a smaller step size until the desired accuracy for $p_c$ had been achieved. The result is then checked by graphing the scaled curves and making sure they fall on one universal curve.

To grow the clusters the Leath algorithm was used, the basic steps of which go as follows [4]

1) For a $L$ x $L$ system intialize an $L$+1 x $L$+1 lattice where every space is marked as undefined except the edges, which are marked as free.
2) Occupy the center element and record its four neighbors.
3) Choose one neighbor and calculate a random number $n$, if $n$<$p$ then the site is occupied, else it is free.
4) Determine the neighbors for the calculated site, if a neighbor is undefined and not already on the list of cluster neighbors, add it to the list.
5) Go to the next neighbor on the list and repeat steps 3 and 4 until no neighbors remain.

To determine if the cluster grown by the Leath algorithm is percolating, we must first label the clusters on the lattice. To do this we use the Hoshen-Kopelman algorithm [5][4]

Raster scan the grid, every time an occupied site is reached perform the following checks:
1) If the site has no occupied neighbors, it gets a new label.
2) If the site has one occupied neighbor it inherits the neighbors label.
3) If the site has two occupied neighbors choose the lower label and make note that the two different labels are a part of the same cluster
Once this is completed for the entire grid, raster scan it again and collapse all labels that are part of the same cluster.

Once all the clusters are labeled the leftmost column is checked and all labels that correspond to occupied states are recorded and then checked against the labels of the rightmost column. If a label appears in both columns, the system is percolating (This is for right to left percolation, for up to down the leftmost column would be placed with the first row, and the rightmost column would be replaced with the last row).

Graph 1.png

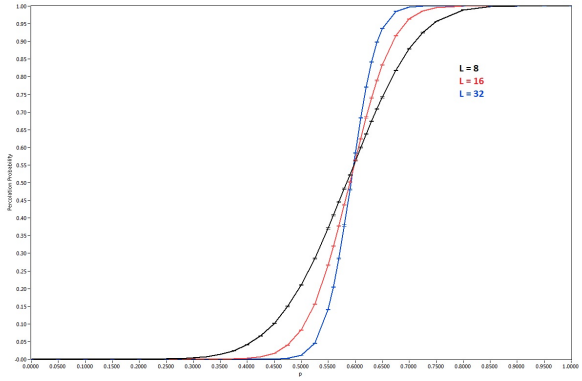



FIG. 4:

## V. RESULTS

For the first set of data systems with sizes $L = 8$, $L = 16$, and $L = 32$ were analyzed for values of $p$ between 0 and 1. The results are shown in Figure 4.

Graph 2.png

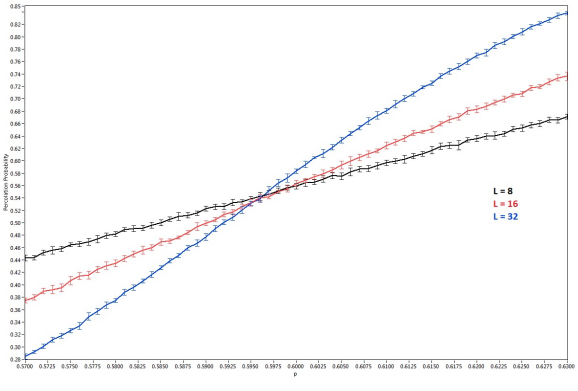



FIG. 5:

Graph 3.png

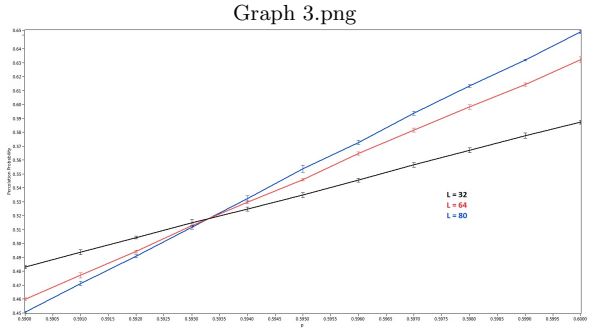



FIG. 6:

Looking at Figure 4 values of $P$ are chosen to be calculated for $p$ between .57 and .63 with a step size of .001. The results are shown in Figure 5.

From Figure 5 values of $P$ are chosen to be calculated for $p$ between .59 and .6 with a step size of .001. For this simulation larger system sizes were considered. The values of $L$ tested were 32, 63, and 80. The results are shown in Figure 6.

From Figure 6 the value of the percolation threshold was determined to be .5925± .0005, which agrees within error of the known value $p_c$=0.59275 [4]. This value of $p_c$ was tested by scaling the data for the $L$=8, 16, and 32 systems (to get an adeqete range of scaled $p$ values). The graph is shown in Figure 7.

The scaled curves were deemed to be overlapping close enough to be within error. Due to the fact that these curves are only an approximation the non scaled data is decided to be more reliable for determining the value of $p_c$. The overlapping scaled curves are used as a check to determine the plausablility of the conclusion drawn from the non scaled data.

Graph 1.png

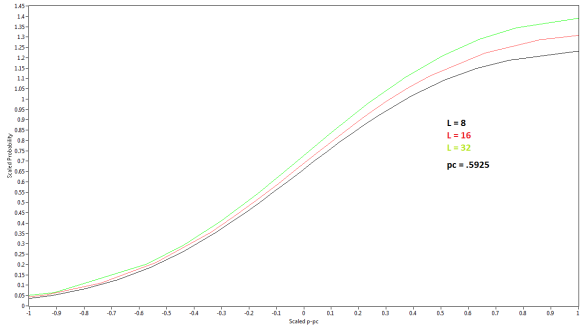



FIG. 7:

## VI. SUMMARY AND CONCLUSIONS

In this paper we have described some of the basic concepts of percolation theory, and analyzed the percolation transition for site percolation on two dimensional square lattices. The percolation threshold $p_c$ was determined to be .5925± .0005, which agrees with the known value within error. Percolation theroy has many useful applications that can range from modeling oil seepage to determining the robustness of a computer network [4]. At the time this paper was completed all data had not yet been collected for larger systems. Some work still remains to be done to refine the estimate of $p_c$ to higher precision.

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] URL `http://en.wikipedia.org/wiki/Percolation_theory`.

[2] URL `http://mathworld.wolfram.com/PercolationTheory.html`.

[3] P. R. King, S. V. Buldyrev, N. V. Dokholyan, S. Havlin, E. Lopez, G. Paul, and H. E. Stanley, *Percolation theory* (02).

[4] K. H., *Week 06: Random walks, percolation, monte carlo integration*, URL `http://katzgraber.org/teaching/SS14/files/lecture-06.pdf`.

[5] F. T., *The hoshen-kopelman algorithm*, URL `http://www.ocf.berkeley.edu/~fricke/projects/hoshenkopelman/hoshenkopelman.html`.

# Self-avoiding random walks and polymers

Nick Amin
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 23, 2014)

A polymer chain can be modeled as a self-avoiding random walk (SAW) of length $N$, where $N$ represents the number of physical monomers in the chain. Unlike a regular random walk, where spatial collisions are allowed, a self-avoiding random walk more accurately represents a physical chain as lattice site occupancy is allowed to be at most one due to the finite dimensions of each monomer. Usage of simple sampling and reptation algorithms allowed the calculation of attrition constants associated with the generation of a SAW, critical end-to-end distance exponents, and radii of gyration. These values are compared to those obtained in previous studies, and most notably, with Flory theory.

## I. INTRODUCTION

Polymers are ubiquitously present in a variety of fields, including biophysics. Their study provides insight into protein behavior, polymerization processes, and DNA [1]. The physical characteristics, such as the length, architecture, and elasticity, of a polymer largely affect their transport properties, or diffusivity.

To build a rudimentary polymer, one can start with a single monomer at a given position and successively concatenate more monomers with given bonding distances and angles in a progressive sequence to form a chain-like structure. Such a polymer can be used as a basis for simulation studies where physical measurements would prove difficult.

## II. MODEL

Imposing the conditions that bond distances between monomers are equivalent, bond angles are constant at 90 degrees, and excluding monomer-monomer interactions, we find that a a self-avoiding random walk (SAW) simplistically models polymers and provides a mathematical tool to study their properties. A SAW is a type of random walk where any given lattice site may not be visited more than once. This imposes an analogous physical constraint to the inability of monomers to share the same volume in space, thus reproducing volume exclusion.

It can be shown that the partition function for SAWs on a lattice of arbitrary dimension takes the form

$$Z \sim \mu^N N^{\gamma-1} \tag{1}$$

in the asymptotic limit as $N \to \infty$, where $\mu$ is the connective constant and $\gamma$ is the critical exponent [2]. As a partition function, this expression gives the scaling of the number of SAWs of length $N$.

Additionally, an important property of these random walks is the mean squared end-to-end distance, $\langle R^2 \rangle$, computed as the square of the linear distance between the two opposite ends of a walk. The functional scaling takes the form

$$\langle R^2 \rangle \sim N^{2\nu} \tag{2}$$

where $\nu$ is known as the Flory exponent and depends on the spatial dimension of the walk. Equivalently,

$$\langle R^2 \rangle \sim N^{2/d_f} \tag{3}$$

where $d_f$ is the fractal dimension of the walk. Through mean-field theory, it can be shown that

$$d_f = \frac{D+2}{3} \tag{4}$$

where $D$ is the spatial dimension of the walk [3]. This Flory expression is exact for $D = 1$, $D = 2$, and $D \geq 4$, since $D = 4$ is the upper critical dimension. In $D = 3$, however, the Flory-predicted value of $d_f = 5/3$ falls short of the empirical value of $d_f = 1.689$ [4].

Similarly to the mean squared end-to-end distance, the radius of gyration [5] of a random walk,

$$R_g^2 \equiv \frac{1}{N}\langle \sum_i^N (\mathbf{r}_i - \mathbf{r}_{CM})^2 \rangle \tag{5}$$

which gives a measure of the distance of points along the walk from the center of mass of the walk, obeys the scaling relation

$$R_g \sim N^{\nu}. \tag{6}$$

## III. METHODS

There are a variety of sampling methods to study the scaling relations of self-avoiding random walks, including simple sampling, reptation, and pivot algorithms [4]. However, this paper will only discuss simple sampling and the reptation algorithm, which were both used to produce and evolve random walks in arbitrary dimensions (2- and 3-dimension examples of SAWs are shown in Fig. 1).

### A. Simple Sampling

Simple sampling of SAWs is achieved using an algorithm that moves a "walker" in random directions until

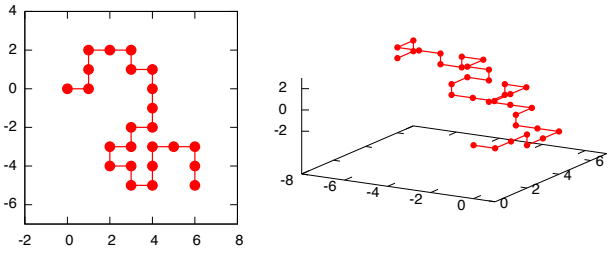


FIG. 1: An example of a 2-D self-avoiding random walk for $N = 24$ (left) and a 3-D self-avoiding-random walk for $N = 35$ (right).

a collision occurs. If the walk reaches the specified size $N$, then we calculate and save the parameters of the walk. The algorithm consists of the steps:

1. Start walker at origin
2. Take step in random direction
3. If this new location has been previously occupied, reject walk and restart at origin
4. Repeat until number of accepted walks reaches number of desired walks

The choice of a random direction in an arbitrary number of dimensions $D$ in a "step" routine is done via:

```
// pick random integer in [0,2*D]
r = rand(2*D)
d = floor(r / 2)
s = r % 2
// shift position in dimension d
pos[d] += 2*s-1
```

Due to the rejection of unavailable walks, the simple sampling method is ergodic in that it can sample all possible walks given adequate time.

If the previous location is stored along with each step, then it is straightforward to prevent the walker from moving to its previously occupied lattice site. This allows for an exponential performance improvement [6]. Despite this, one sees that rejection is highly probable, especially for walks with large $N$. This gives rise to the attrition problem, which is the major drawback to the simple sampling method for generating walks, as generation times for large walks quickly become infeasible (Table I). An attrition factor can be calculated as the ratio of successfully generated walks to the number of attempts (successes + rejections), and is demonstrated in Figure 2. Note that for $D = 3$, $\gamma \approx \frac{7}{6}$ [7]. In higher dimensions, collisions become less of a problem due to the extra degrees of freedom that a walker may make use of during the generation phase.

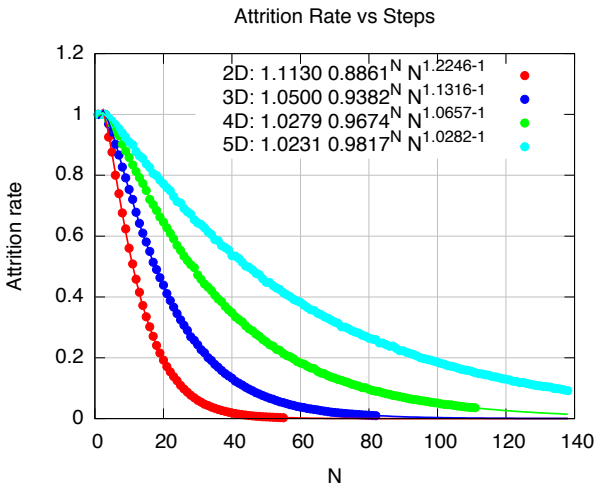



FIG. 2: Attrition rate (calculated as the number of successfully simply generated SAWs divided by the total number of attempts) for various dimensions. Each curve is fit with the form of the partition function, $Z \sim \mu^N N^{\gamma-1}$.

TABLE I: Number of steps needed to achieve specified attrition rate $a_r$. Dimension 1 is not applicable to this calculation because of the requirement that walkers may not move backwards to their previous location, so the attrition problem is trivially non-existent in 1 dimension.

| $D$ | $a_r = 0.1$ | $a_r = 0.001$ | $a_r = 0.0001$ |
|---|---|---|---|
| 2 | 26 | 46 | 66 |
| 3 | 45 | 82 | 119 |
| 4 | 79 | 150 | 220 |
| 5 | 134 | 259 | 384 |

Due to the rejection of unavailable walks, the simple sampling method is ergodic in that it can sample all possible walks given adequate time.

### B. Reptation Algorithm

The reptation algorithm, which gets its name from the similarities of the walk to a snake, solves the attrition problem plaguing the simple sampling method by allowing multiple measurements of quantities for a single walk. The algorithm relies on a reptation step which provides a basis for the "slithering" of the walk:

1. Use simple sampling or create a custom walk with desired length $N$
2. Reptate
   (a) With 50/50 probability, choose one end of walk ("head" or "tail")

(b) Take last step from chosen side and try to attach it to opposite side with a random direction

(c) Check for a collision. If there is one, reject the move and do not modify the walk; otherwise, apply the move

(d) Calculate and store the desired parameters of the walk

3. Repeat reptation until desired amount of statistics reached

Unlike the simple sampling generation method, the reptation algorithm is able to slowly slither out of potential trapping configurations, where the final end of the walk is locked into the interior of the rest of the walk, providing only a finite amount of steps until the walk is forced to collide.

From this algorithm, only one simply-generated SAW is needed to collect an arbitrary amount of data. One sees that regardless of the success of a reptation step, walk parameters can be calculated and stored, allowing each reptation to serve as a new data point.

Since this method does not depend on the initial walk, we can generate an arbitrary walk. This is necessary for large $N$ walks, when the initial simply-generated walk is difficult to attain even once. As an alternative, a staircase-like snake has been used for this study. To generate the staircase, the walker takes a "forward" step in each dimension randomly (Fig. 3).

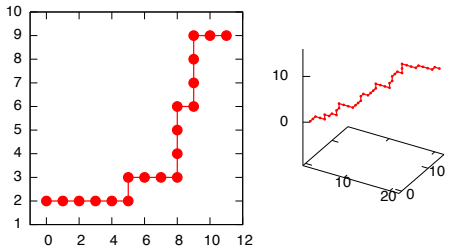



FIG. 3: Staircase 2-D SAW for $N = 15$ (left) and a 3-D SAW for $N = 50$ (right).

Bias due to the choice of shape of this initial snake is easily eliminated by performing many "thermalization" reptations until measured quantities equilibrate. As an example, the staircase snake would have a larger $\langle R^2 \rangle$ value than a spiral snake initially. Allowing walks in different dimensions to evolve over time (reptations), a rough number of reptations associated with overall equilibration can be observed and later used to remove skewing of $\langle R^2 \rangle$ values. Given a staircase starting state, Figure 4 shows that walks are sufficiently equilibrated after one million reptations. Even after thermalization, some end-to-end lengths of walks fluctuate wildly due to the nature of a random walk, but retain an average close to the desired value. Consequently, all walks used for data collection were reptated at least 1.5 million times to ensure thermalization before collecting any data.

Unlike the simple sampling method, the reptation algorithm can rarely produce configurations from which it cannot be extricated. This includes a double spiral, where both ends of the walk are trapped within the rest of the walk. As a result, the algorithm is not ergodic.

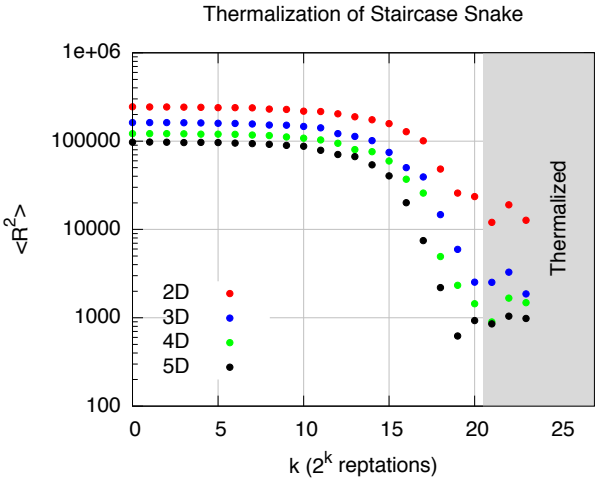



FIG. 4: Thermalization for initial staircase SAWs with $N = 700$. $\langle R^2 \rangle$ values are averaged over $2^k$ reptations with the same initial walk. Each point is also an average over 10 runs. It is seen that somewhere between $2^{20}$ and $2^{21}$ reptations, an equilibrated value is reached with expected fluctuations. Thus, it is sufficient to discard the first $1.5 \cdot 10^6$ reptations before beginning the collection of data. Note that the higher dimensional walks thermalize faster than the lower dimensional walks due to the increased degrees of freedom.

## IV. RESULTS

With the framework for SAW generation using an initial staircase configuration and the reptation algorithm to store large quantities of observables, the radius of gyration and the end-to-end length can be measured with high precision. In order to complement the computational power of the reptation algorithm, a high performance computing cluster was used to produce walks of length 10 to 1200 in increments of 10 with 1.5 million thermalization reptations followed by 100 runs of data collection with 2 million reptations each.

The radius of gyration of walks was computed as a function of $N$ for each dimension and displays expected properties. In 1-D, the radius of gyration scales as $N$; in 4 dimensions and higher, it scales as $\sqrt{N}$, which is consistent with the non-avoiding random walk radius of gyration. In particular, making use of Eq. 6 with $D = 3$,

TABLE II: Fractal dimension from fit and from Flory expression ($D = 1, 2, 4, 5$) or precise calculation ($D = 3$). D

| $D$ | Fit $d_f$ | Theoretical $d_f$ |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1.3362 ± 0.0028 | 4/3 |
| 3 | 1.6912 ± 0.0039 | 1.689 ± 0.0086 (95% c.l.) |
| 4 | 1.9001 ± 0.0045 | 2 |
| 5 | 1.9709 ± 0.0038 | 2 |

we obtain $d_f = 1.6986 \pm 0.0033$, which is consistent with a radius of gyration exponent $\nu = 0.588$ ($d_f = 1.701$) in Ref. [8].

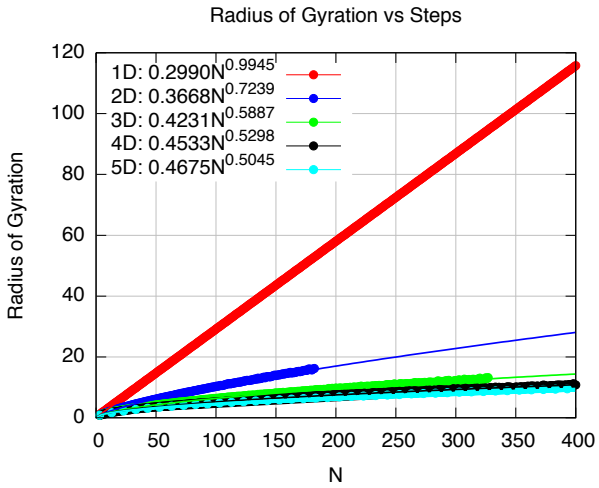



FIG. 5: Radius of gyration (from the square root of Eq. 5) for walks of different dimensions as a function of walk length. Power-law scaling of radius of gyration explicitly shows the Flory exponent $\nu$.

Plotting $\langle R^2 \rangle$ as a function of length yields a wealth of information about the fractal dimension of these self-avoiding random walks. Comparisons between obtained values of the fractal dimension and those from Flory are delineated in Table II. One dimension is trivially consistent. Dimensions 2 and 3 are within approximately 1-$\sigma$ of the Flory and precisely calculated values. Dimension 5 is ≈8-$\sigma$ away from the mean-field theory value, and $D = 4$ appears to be inconsistent. However, as dimension 4 is the upper critical dimension and the critical exponents have the mean-field theory values, the obtained $d_f$ values for $D = 4, 5$ require more statistics to deliver accurate results.

## V. SUMMARY AND CONCLUSIONS

A self-avoiding random walk provides a suitable mathematical model to provide a numerical alternative to measuring certain observables of a polymer. Simple sampling and the reptation algorithm both allow for generation of these walks and for their subsequent analysis. However, the former is coupled with the problem of exponential attrition. Using the latter, it was ultimately shown that the fractal dimension calculated for $D = 1, 2, 3$ agrees with Flory theory and empirical calculation, whereas $D = 4, 5$ are not fully consistent due to the necessity for larger statistics.

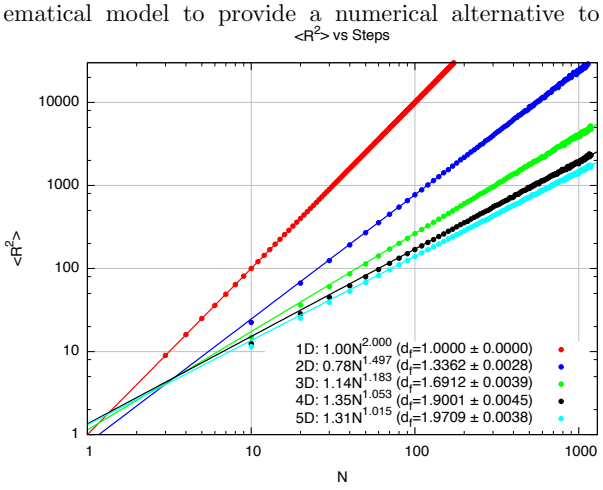



FIG. 6: Average end-to-end distance for walks of different dimensions are plotted against walk length. Error bar sizes are comparable to point marker sizes, so they are excluded for clarity. From the power scaling, a critical exponent and fractal dimension have been extracted. Note that the first few points have been excluded from the fit in order to eliminate undesirable finite-size effects.

Calculations described here may be improved in the future with the addition of the pivot algorithm and more statistics.

### Acknowledgments

I thank the Texas A&M University Brazos HPC cluster that contributed to the research reported here and Texas A&M University for access to their Eos cluster.

[1] N. G. McCrum and C. P. Buckley, *Principles of Polymer Engineering* (Oxford University Press, 1997).

[2] J. Machta and T. R. Kirkpatrick, *Self-avoiding walks and manifolds in random environments*, Phys. Rev. A **41**, 5345 (1990), URL `http://link.aps.org/doi/10.1103/PhysRevA.41.5345`.

[3] P. J. Flory, *Principles of Polymer Chemistry* (Cornell University Press, Ithaca, 1953).

[4] N. Madras and A. D. Sokal, *The pivot algorithm: A highly efficient Monte Carlo method for the self-avoiding walk*, J. Stat. Phys. **50**, 109 (1988).

[5] I. Jensen, *Enumeration of self-avoiding walks on the square lattice*, Journal of Physics A: Mathematical and General **37**, 5503 (2004), URL `http://stacks.iop.org/0305-4470/37/i=21/a=002`.

[6] C. Beleno and K. Yao, *Polymers*, URL `http://www.iskp.uni-bonn.de/uploads/media/polymers.pdf`.

[7] J. Batoulis and K. Kremer, *Statistical properties of biased sampling methods for long polymer chains*, Journal of Physics A: Mathematical and General **21**, 127 (1988), URL `http://stacks.iop.org/0305-4470/21/i=1/a=020`.

[8] J. D. Moroz and R. D. Kamien, *Self-Avoiding Walks with Writhe*, eprint arXiv:cond-mat/9705066 (1997), cond-mat/9705066.

# RNG Testing

William Baker
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 25, 2014)

Abstract: I examine the performance of several random number generator's (RNG's) in a square (16x16) Ising lattice by computing the heat capacity ($C_V$) at the critical transition temperature ($T_C$) and compare to the analytical solution for a finite lattice. $10^8$ sweeps using the Wolff Algorithm over 15 independent lattices were computed to generate precise data points for the heat capacity calculated for each independent lattice. The goal is to emulate and add to previously produced results.

## I. INTRODUCTION

Random number generators are crucial elements of many computer-based simulations. The Monte Carlo method is one such simulation technique which uses random numbers to simulate, for example, the distribution of energy in a spin-lattice, the temperature of a box, or an integral which is too difficult to solve by hand. The 2D Ising model is one such situation in which Monte Carlo sampling plays a major role. Individual spins in a 2 dimensional lattice are selected at random and are flipped with a probability which also requires a random number to be generated. There are several algorithms for spin flipping including the Wolff Algorithm [1]. The Wolff Algorithm selects spins at random and adds surrounding spins to the cluster with probability

$$\mathcal{P}(T) = 1 - exp(-2 * J/k_bT) \tag{1}$$

Where $J$ is the spin-spin coupling constant, and $k_b$ is the Boltzmann constant, both of which are set to unity. $T$ is the absolute temperature of the lattice and at $T = T_C = \frac{2}{ln(1+\sqrt{2})} \approx 2.269185$, the probability for a spin to be added to the cluster, and thus flip, is %58.58.

Studies have been done on several RNG's [2], showing the inadequacies of several "good" generators, including r250 and r1279. Using an iMac12.2 workstation, I carried out extensive simulations on a $16x16$ Ising square lattice with periodic boundary conditions for which exact results are known [3]. Several RNG's were tested including r250, r1279, randu, and drand48, which are all generators available in the Gnu Scientific Library (GSL) and were tested in [2]. drand48 is a linear congruential algorithm, and r250, r1279, and randu are shift register algorithms. A few other generators were tested as well including the Mersenne Twister, Random123, fishman20, knuthran2002, ran1, ran2, and ran3, all of which, with the exception of Random123, are available in the GSL. Random123 is available through DeShaw Research and is a "counter-based" RNG.

## II. MODEL

The Partition function for a 2D Ising model is given explicitly by:

$$Z_{nm}(T) = \frac{1}{2}(2sinh(2K)^{\frac{mn}{2}}) \sum_{i=1}^{4} Z_i(K) \tag{2}$$

where $K = 1/T$, and $Z_i$ are the partial partition functions and are defined by:

$$\begin{aligned} Z_1 &= \prod_{r=0}^{n-1} 2cosh(\frac{1}{2}m\gamma_{2r+1}), \\ Z_2 &= \prod_{r=0}^{n-1} 2sinh(\frac{1}{2}m\gamma_{2r+1}), \\ Z_3 &= \prod_{r=0}^{n-1} 2cosh(\frac{1}{2}m\gamma_{2r}), \\ Z_4 &= \prod_{r=0}^{n-1} 2sinh(\frac{1}{2}m\gamma_{2r}), \end{aligned} \tag{3}$$

where $m$ is the length of the lattice and $\gamma$ is defined such that

$$\begin{aligned} \gamma_0 &= 2K + ln(tanh(K)) \\ \gamma_l &= ln(c_l + (c_l^2 - 1)^{\frac{1}{2}}), l > 0 \\ c_l &= cosh(2K)coth(2K) - cos(\frac{l\pi}{n}) \end{aligned} \tag{4}$$

Given that the heat capacity is defined by:

$$C_V = K^2(d^2/dK^2)ln(Z) \tag{5}$$

we can calculate the exact value for the heat capacity of a finite lattice. This calculation was performed in Mathematica and gave me a heat capacity per spin of $\frac{C_V(T_C)}{16^2} = 1.498704959$.

The Hamiltonian of a 2D Ising model with no external magnetic field is given by

$$\mathcal{H}(\{s_i\}) = -\sum_{i \neq j}^{N} J_{ij} s_i \, s_j \tag{6}$$

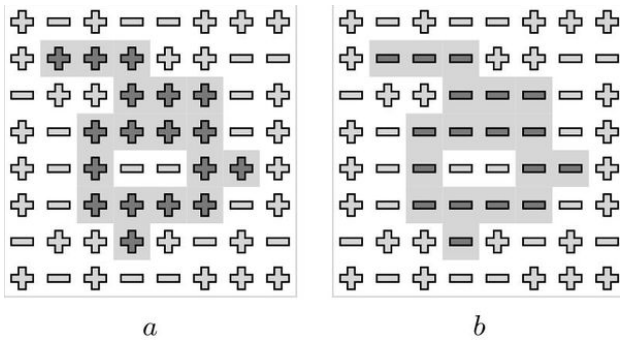


FIG. 1: A cluster of positive spins on a 2D lattice is grown using the Wolff Algorithm, and then flipped.

Where again, we set $J_{ij} = 1$. From this, the energy of the system at each step can be calculated, and the $C_V$ can be calculated from:

$$\begin{aligned} C_V &= \frac{Var(E)}{T^2} \\ &= \frac{< E^2 > - < E >^2}{T^2} \end{aligned} \tag{7}$$

## III. METHODS

### A. Clustering Algorithm

The Ising Model used was a $16x16$ matrix of spins with periodic boundary conditions allowing for spins of spin $+1$ or $-1$, and the algorithm used for flipping was the Wolff Algorithm. 15 independent chains were generated and swept through with the Wolff Algorithm $10^8$ times. The Wolff algorithm selects a spin from the lattice at random and flips it, then checks the surrounding lattice sites for spins with orientations equivalent to the original spin orientation and flips them with a probability given by (1). Successful flips incite further investigation of surrounding lattice sites and a cluster is grown through a self referencing function. The energy is calculated at each sweep and the average energy and squared energy are recorded. Not until the end of each independent markov chain is the heat capacity recorded.

### B. Random Number Generation

Each RNG was implemented individually and run independently. Being that random number generation is used only when initializing the spin lattice, determining the initial spin to flip, and proposing additions to the cluster, replacing the generator was trivial. The error and standard deviation for each set of heat capacities recorded was calculated using the Jackknife method of averaging, which removes each data point and averages over the resulting array of data points for each value in the array. The averages are obtained using the formula:

$$mean_i = \frac{(actualmean * N - value_i)}{N - 1} \tag{8}$$

and the error and standard deviation are calculated using the average of these averages.

#### 1. Mersenne Twister

The Mersenne Twister [4] algorithm for random number generation is based on the linear recurrance:

$$x_{k+m} := x_{k+m} \oplus (x_k^u | x_{k+1}^l) A \tag{9}$$

where $n$ is the degree of recurrence, $m$ is between 1 and $n$, inclusive, $\oplus$ is the bitwise addition operator, and $x_n$ is an initial seed. $A$ is a matrix with values defined by the algorithm. The period of a stream of random numbers is $2^{19937} - 1$

#### 2. Random 123

Random 123 [5] is a counter based RNG, which returns values based on a incremental counter and a key. The generator is deterministic, in that if you give it the same key, you will produce the same string of numbers. Given an initial state vector $s_0$ (defined by the key), random sequences are generated by successive iteration of:

$$\begin{aligned} s_m &= f(s_{m-1}) \\ u_{k,n} &= g_{k,n mod J}(s_{[n/J]}) \end{aligned} \tag{10}$$

where $u$ is an output based on the key ($k$) and the counter ($n$), and is given by an output function $g$, operating on the state $s$, which is changed with each generation. The key is initalized using any other random number generator. There are a few different counter-based random number generators available in the Random123 library, and the one used here is threefry4x64, which requires only common bitwise operation in the calculation of the stream. Threefry is also designed to perform well across several architectures making it an attractive choice. The period of a stream of random numbers is at least $2^{128}$.

#### 3. r250 and r1279

R250 and r1279 are shift-register generators, whose sequences are defined by:

$$\begin{aligned} x_n &= x_{n-103} \otimes x_{n-250} \\ x_n &= x_{n-418} \otimes x_{n-1279} \end{aligned} \tag{11}$$

respectively. The period of r250 is about $2^{250}$ and the period of r1279 is about $2^{1279}$

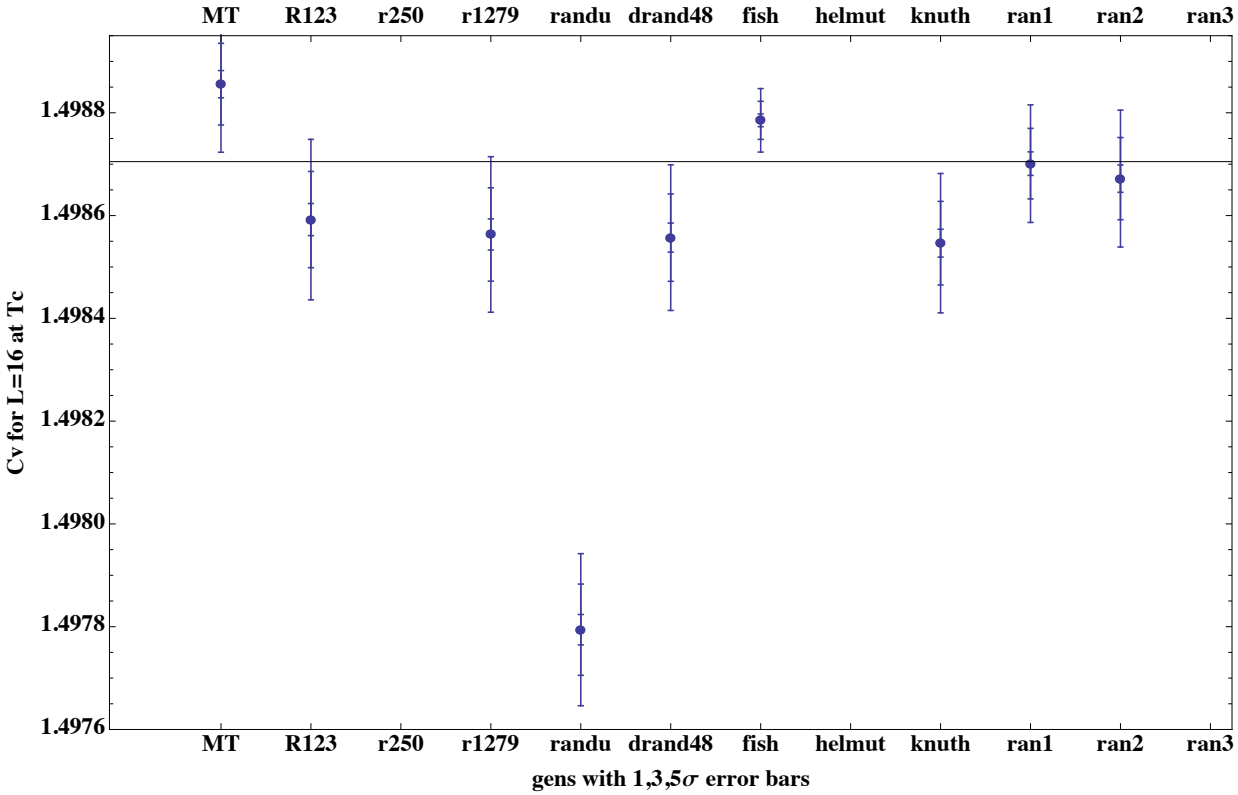



FIG. 2: Generator averages with error bars shown for 1,3, and 5 $\sigma$ for each generator. Helmutrand and ran3 are several standard deviations above the expected heat capacity, and are not shown. Similarly, r250 is several hundred sigma below the expected heat capacity, and is not shown.

#### 4. *drand48, fishman20, and helmutrand*

Drand48, fishman20, and helmutrand are linear congruential generators, meaning that each generated number is produced through a linear function of the preceding number, modulo some relatively prime number. Drand48 returns the upper 32 bits from each pseudo-randomly generated sequence while fishman20 and helmutrand simply return integers. The sequence is updated via the algorithm:

$$x_{n+1} = (a * x_n + c) mod(m) \tag{12}$$

where for drand48, using 48-bit unsigned arithmetic, $a = 0x5DEECE66D$, $c = 0xB$, and $m = 2^{48}$. The seed sets the initial condition for the upper 32 bits and the lower 16 are set by default to $0x330E$. In helmutrand, $a = 65539$, $c = 0$, and $m = INTMAX - 1$.

#### 5. *randu*

Randu is the linear congruential generator infamous for producing highly correlated random numbers, as can be seen when plotting triplets of the numbers in a 3D space. The sequence of numbers is given by a simple modular formula:

$$x_{n+1} = (65539 * x_n) mod(2^3 1) \tag{13}$$

where $x_0$ is specified by the seed. The period of this generator is $2^{29}$ numbers.

#### 6. *ran1, ran2, knuthran2002*

Ran1, ran2, and knuthran2002 are combined recursive generators, which return integers calculated using a linear combination of the previous two generated numbers, modulo some relatively prime number, where the first two numbers are set by the seed. Simply put:

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2}) mod(m) \tag{14}$$

#### 7. *ran3*

Ran3 is a lagged Fibonacci generator which returns integers based on a recurrance scheme by which the new integer is calculated using two of the previous integers,

combined using some operation, modulo some relatively prime number. The most general form is:

$$x_n = x_{n-j} \star x_{n-k} mod(m) \tag{15}$$

where the star denotes some operation.

## IV. RESULTS

It can be seen, upon examination of the produced data in Table 1 , that even generators with long periods do not produce truly random numbers. Heat capacities computed using the shift-register generators were lower than the exact value, and in the case of r250, specifically, the heat capacity was off by over $1500\sigma$. The congruential generator drand48 and counter based generator Random123 produced specific heats close to the exact value and the Mersenne Twister produced values close to but higher than the exact result, as shown in Figure 2. These generators produced results that were correct to within their error bars, in stark contrast with randu, helmutrand and r250. The generators drand48 and knuthran2002 produce only slight less accurate heat capacities, and fishman20 produces not only accurate heat capacities, but precise as well. ran1 and ran2 produced almost exact results, however ran 3 gave a heat capacity far above the exact value, which was surprising, considering the similarity of the algorithms.

## V. SUMMARY AND CONCLUSIONS

Pseudo-Random Number Generators often have correlations in the numbers produced. Sometimes, as in the case of randu, these correlations can be easily discovered, but often times these correlations are subtle and difficult to discover. The Wolff algorithm for spin flipping in a 2D Ising Model can be used to discover the existence of these subtle correlations, and can often times expose "high quality" generators to have systematic effects on the numbers produced. Counter based random number generators such as Random123 and linear recurrance generators such as the Mersenne Twister can produce numbers which give fairly accurate representations of physical phenomena, while shift register algorithms such as r250 and r1279 give results skewed towards higher energy and lower specific heats. While r1279 still gives results only slightly less accurate than Random123, it stands to reason that, because it's algorithm is similar to that of r250, it may have similar correlation effects however these effects would still be difficult to find. The reasons for this are subtle and are proposed to be correlations in the sequences which affect the Wolff algorithm in a special way [2]. Exact reasons are yet unknown and may be interminably difficult to discover, as the primary purpose behind using the Wolff Algorithm to analyze random number generators is to discover their efficacy at producing truly random strings, not to discover the exact correlations. It would be fair to say, that with deviations from the actual heat capacity of several hundred $\sigma$, you should stay away from r250 and helmutrand if you want to run a Monte Carlo simulation.

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] U. Wolff, *Collective Monte Carlo updating for spin systems*, Phys. Rev. Lett. **62**, 361 (1989).

[2] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, *Monte Carlo simulations: Hidden errors from "good" random number generators*, Phys. Rev. Lett. **69**, 3382 (1992).

[3] A. E. Ferdinand and M. E. Fisher, *Bounded and inhomogeneous ising models. i. specific-heat anomaly of a finite lattice*, Phys. Rev. **185**, 832 (1969).

[4] M. Matsumoto and T. Nishimura, *Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Trans. Model. Comput. Simul. (1998).

[5] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw, in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (2011).

TABLE I: Values for the heat capacity for fifteen independent runs with $L = 16$ at $T_C$, obtained using the Wolff Algorithm. The last number in each column, labeled "$\sigma$", gives the difference between the simulation value and the exact value, measured in terms of the standard deviation of the simulation.

| | $MT$ | $R123$ | $r250$ | $r1279$ | $randu$ | $drand48$ | $fishman$ | $helmut$ | $knuthran$ | $ran1$ | $ran2$ | $ran3$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1.49908 | 1.49850 | 1.45514 | 1.49919 | 1.49781 | 1.49811 | 1.49867 | 1.50625 | 1.49820 | 1.49826 | 1.49847 | 1.50905 |
| | 1.49925 | 1.49867 | 1.45617 | 1.49844 | 1.49796 | 1.49939 | 1.49876 | 1.50627 | 1.49910 | 1.49854 | 1.49857 | 1.50955 |
| | 1.49856 | 1.49858 | 1.45529 | 1.49841 | 1.49784 | 1.49878 | 1.49857 | 1.50629 | 1.49878 | 1.49857 | 1.49889 | 1.50923 |
| | 1.49858 | 1.49772 | 1.45488 | 1.49812 | 1.49794 | 1.49883 | 1.49851 | 1.50621 | 1.49821 | 1.49860 | 1.49830 | 1.50969 |
| | 1.49877 | 1.49929 | 1.45571 | 1.49812 | 1.49789 | 1.49874 | 1.49881 | 1.50619 | 1.49844 | 1.49910 | 1.49834 | 1.51031 |
| | 1.49935 | 1.49918 | 1.45520 | 1.49867 | 1.49801 | 1.49863 | 1.49874 | 1.50609 | 1.49895 | 1.49885 | 1.49860 | 1.50881 |
| | 1.49846 | 1.49836 | 1.45534 | 1.49847 | 1.49807 | 1.49889 | 1.49891 | 1.50648 | 1.49856 | 1.49902 | 1.49891 | 1.51033 |
| | 1.49849 | 1.49860 | 1.45492 | 1.49907 | 1.49810 | 1.49889 | 1.49883 | 1.50606 | 1.49869 | 1.49879 | 1.49882 | 1.50918 |
| | 1.49864 | 1.49884 | 1.45561 | 1.49861 | 1.49682 | 1.49814 | 1.49905 | 1.50618 | 1.49791 | 1.49908 | 1.49870 | 1.50994 |
| | 1.49911 | 1.49933 | 1.45583 | 1.49888 | 1.49787 | 1.49821 | 1.49903 | 1.50646 | 1.49822 | 1.49928 | 1.49894 | 1.50857 |
| | 1.49893 | 1.49860 | 1.45546 | 1.49820 | 1.49787 | 1.49869 | 1.49889 | 1.50619 | 1.49845 | 1.49856 | 1.49796 | 1.50973 |
| | 1.49857 | 1.49834 | 1.45528 | 1.49830 | 1.49811 | 1.49801 | 1.49890 | 1.50636 | 1.49796 | 1.49847 | 1.49853 | 1.50877 |
| | 1.49828 | 1.49832 | 1.45499 | 1.49779 | 1.49684 | 1.49800 | 1.49887 | 1.50610 | 1.49890 | 1.49863 | 1.49860 | 1.50936 |
| | 1.49941 | 1.49821 | 1.45487 | 1.49891 | 1.49813 | 1.49843 | 1.49866 | 1.50602 | 1.49884 | 1.49852 | 1.49962 | 1.50925 |
| | 1.49928 | 1.49826 | 1.45579 | 1.49920 | 1.49759 | 1.49854 | 1.49850 | 1.50650 | 1.49892 | 1.49815 | 1.49875 | 1.50923 |
| $< C >$ | 1.49886 | 1.49859 | 1.45537 | 1.49856 | 1.49779 | 1.49856 | 1.49879 | 1.50625 | 1.49855 | 1.4987 | 1.49867 | 1.50941 |
| $error$ | 0.00015 | 0.0001 | 0.04333 | 0.00014 | 0.00091 | 0.00014 | 0.00008 | 0.00754 | 0.00015 | 0.000003 | 0.00003 | 0.01070 |
| $\sigma$ | 5.69048 | −3.61301 | −1548.33 | −4.68286 | −30.7763 | −5.22109 | 6.50783 | 693.523 | −5.85132 | −0.170754 | −1.24074 | 284.476 |

# Predator-Prey and Multiple Species Interactions Using the Lotka-Volterra Model

Robert Bordovsky
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 26, 2014)

We explore numerical solutions of the Lotka-Volterra model for Predator-Prey systems. Using Matlab's Ode45 numerical solver we examine solutions for two and three species predator prey systems for different parameters and initial conditions. We also study how the stochastic solutions to the two species predator prey and single species logistic equation compare to their continuous deterministic counterparts.

## I. INTRODUCTION

The Lotka-Volterra equations, developed independently by Lotka and Volterra in the 1920's, provide perhaps the simplest model of predator-prey interactions. The equations naturally extend the classic Logistic population model for single species first investigated by Verhulst in the 1830's, and provide a crude explanation of observed phenomena in the natural world such as the oscillations between populations of predator and prey species [1]. In addition, the types of interacting systems described by the Lotka-Volterra Model also arise in other fields of study such as economics [2] and chemistry for which Alfred Lotka initially proposed the model [3]. For this reason, there is great mathematical interest in studying the solutions to these equations.

The purpose of this study was to build on our previous work done with population models and explore the result of adding complexities that address the known limitations of the deterministic equations as originally formulated. We start this by revisiting previously solved systems and adding complexity in the form of stochastic processes. This is a topic that has already been thoroughly explored by previous scholars [4] [5], so we have limited the scope of this paper to a fairly simple model. We also demonstrate the effect of adding a time-delay to the single species model in the method of Haberman [1] to demonstrate the destabilizing influence such effects have. Analysis of the model with more than three species is limited to seeking stable solutions of the continuous, deterministic Lotka-Volterra equations and examining the results of adjusting the parameters of that model.

## II. MODEL

### A. The Stochastic Model: Wiener Processes

In stochastic modeling, it is fairly common to use some numerical integration scheme, such as the Euler method, while adding random fluctuations at each time step, but there is wide variety in how these fluctuations are chosen. We ran across several different stochastic models for predator prey systems [4] [5], but chose to model the Lotka-Volterra system using a Wiener Process, also know as Standard Brownian motion. This decision was made due to the simplicity of this stochastic model, and the large amount of literature available on the subject [6]. A Wiener process W, is a continuous time stochastic process such that the difference in value between two time steps $W_{t_1} - W_{t_2}$ is a normally distributed Gaussian number with variance $\Delta t = t_1 - t_2$. The differential for this process is known as Guassian White Noise. In practice, we can add this noise to our process and scale the variance to the desired level by adding a term $\sigma * \text{randn} * \sqrt{\Delta t} * (CurrentPopulation)$ to the population value for each time step we take while doing the integration. In Matlab, the function randn returns a psuedo-random number drawn from the standard normal distribution, and the $\Delta t$ represents the time elapsed in between steps. This basic method is used to analyze the results of several different noise levels.

### B. Other Numerical Methods: MatLab Ode45 Solver

All numerical calculations and plots for this project were done in the computational software Matlab. One of the built in functions of this software worth highlighting is Matlab Ode45. This is a differential equation solver that uses a 4th and 5th order Runge-Kutta method with variable step sizes to integrate equations from a given set of initial conditions. This solver was used to produce results for all deterministic equations in this paper. Stochastic and delayed-difference equation were solved using the less sophisticated Euler method of integration.

### C. The Logistic Equation

The continuous-time logistic equation is widely known in the formulation

$$\frac{dN(t)}{dt} = N(t)[a - bN(t)] \tag{1}$$

Here the parameter $a > 0$ represents the growth rate of the species whenever individual members of the population do not compete with each other for food or resources. The parameter $b > 0$ is a measure of how these competitive interaction negatively impact total growth and serve as a correction term for a system with finite resources.

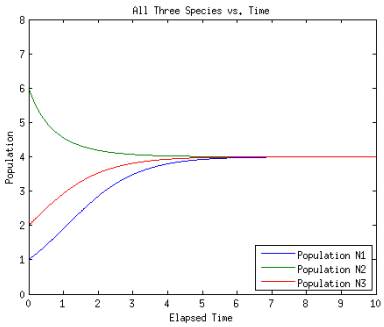



FIG. 1: Ode45 Solution of Continuous Logistic Equation: The red, green, and blue lines represent three separate solutions with initial populations of 2.0, 6.0, and 1.0 respectively. The growth rates are $a = 1.0$ and $b = -0.25$ for all three species giving each a carrying capacity of 4.0. The solutions for initial populations both above and below this equilibrium value converge to this stable equilibrium.

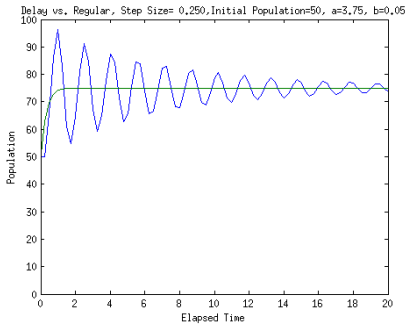



FIG. 2: Oscillatory Solution of the Time-Delayed Logistic Equation: The solution to the time delayed equation is plotted in blue on top of the solution of the continuous equation in green. Parameters were $a = 3.75$, $b = 0.05$ and both populations started from an initial value of 50. The population of the time-delayed equation initially overshoots the equilibrium value of 75 and then over-corrects. It continues this motion in a decaying oscillation around the equilibrium value.

This equation has one non-trivial equilibrium at $\frac{a}{b}$ which is stable and is known as the *carrying capacity*.

A representative output of the continuous equation is displayed in Figure 1. The system behaves as expected with all three populations asymptotically converging to the carrying capacity of the system, regardless of their initial values.

While the continuous equation is easy to analyze, more realistic models for single species population growth are possible if the differential equation is broken into discrete time steps and then numerically solved with simpler integration techniques. These difference equations allow us to build expected behaviors into our population models such as time-delays between birth and resource depletion, or random births and death.

To account for the time delay between birth and overconsumption of resources, we used the time delay logistic equation outlined by Richard Haberman [1]. The proposed model is

$$\frac{dN(t)}{dt} = N(t)[a - bN(t - \Delta t)]$$

written as a continuous equation. We can break this down into an equation of finite difference and solve for future time steps based on past values.

$$N_{t+1} = N_t + N_t[a - bN_{t-1}]\Delta t \qquad (2)$$

This form is readily integrated in with a simple **for** loop:

```
for i = 2:1:step_number
  N(i+1) = N(i)*((a*step_size)-(b*step_size)*N(i-1))
             +N(i);
end
```

Results of running this integration are plotted in Figure 2. The inclusion of time steps results in oscillatory behavior that is not present in the continuous equation. In the interpretation of the model, this occurs because the species continues to reproduce past the point that the environment can sustain all members. It is only at time $\Delta t$ after the population passes the equilibrium that the species feels the constraints of limited resources. This causes the death rate to increase to bring the population down, but it again passes the equilibrium.

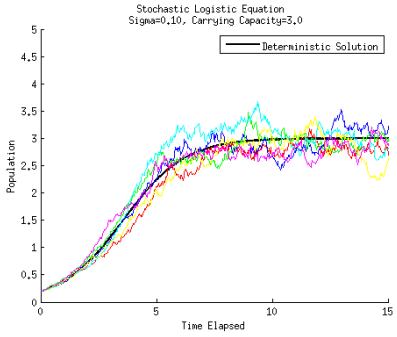



FIG. 3: Stochastic Solution of Logistic Equation: Initial population = 0.2, $a = 0.75$, $b = 0.25$ and $\sigma = 0.10$ and $\Delta t = 0.33$. Each color represents an independent solution of the stochastic process. The black line is a plot of the continuous solution. For each individual run, there is significant variation from the deterministic solution, but there is still a preference for stochastic solutions to stay around the equilibrium value.

Another expected behavior that we would like to build into the our populations models is some sort of random process to reflect the fact that certain processes, such as births and deaths, are not deterministic. Similar to the time-delay equation, we are able to write a stochastic equation in an iterative fashion. Using the Euler method [7], we can write the discrete time difference

equation:

$$N(t+1) = N(t)+N(t)[a-bN(t)]\Delta t+\sigma*N(t)*dW(t) \quad (3)$$

using the for loop

```
for i=1:N
   x(i+1)=x(i)+(x(i)*(a-b*x(i)))*h
           + sigma*x(i)*randn*sqrt(h);
end;
```

Here sigma is the parameter that controls the "strength" of the noise. Quite clearly, a larger value of sigma will lead to greater fluctuations in the stochastic solution.

In Figure 3, we plotted one of the more visually appealing results, $\sigma = 0.10$. From the plotted solutions of the several trial runs we can see that our model creates fluctuations in the population value compared to the deterministic solution. As time increases the deviation from the deterministic solution increase as well. This is particularly apparent in the region from $t = 0$ to about $t = 5$.

### D. Two-Species Lotka-Volterra Model

The Lotka-Volterra equations for two interacting species are

$$\begin{aligned} \frac{dN_1(t)}{dt} &= N_1(t)[a - bN_1(t) - cN_2(t)] \\ \frac{dN_2(t)}{dt} &= N_2(t)[-k + \lambda N_1(t)] \end{aligned} \quad (4)$$

In this formulation, $N_1$ represents the prey species, and $N_2$ represents the predator species. The parameters $a$ and $b$ in the prey equation have the same meaning as in the Logistic model discussed above, and the new parameter $c > 0$ represents the additional death of prey from predators feeding on them. In the equation for the predator, $k > 0$ is the rate at which predators die in the absence of food and the term $\lambda > 0$ is the growth of the species based on the presence of prey.

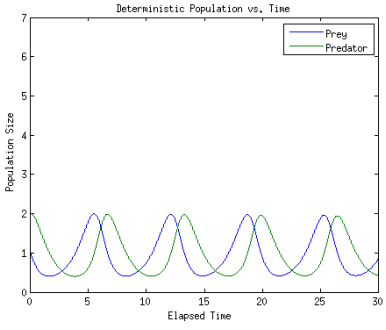



FIG. 4: Single Predator Single Prey System: This constitutes the base Predator Prey System for this study. The blue line indicates the size of the prey population and the green line indicates the size of the predator population. Solutions exhibit oscillatory behavior. The parameters for this system are $a = 1.0, b = 0.0065, c = 1.0, k = 1.0, \lambda = 0.05$.

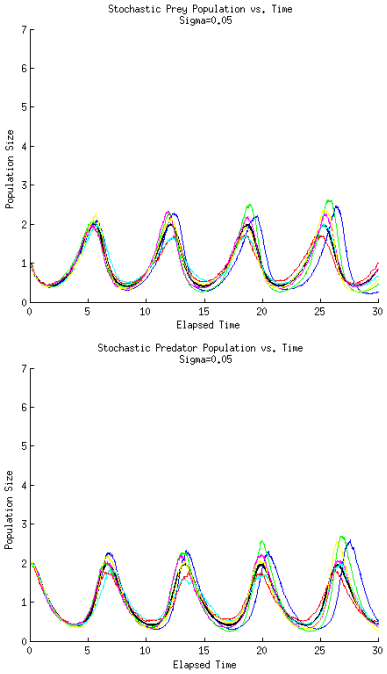



FIG. 5: Stochastic Predator-Prey Population vs. Time; $\sigma = 0.05$: Each color on the graph represent an individual run of the test code plotted on top of the deterministic solution, indicated by the bold black line. The top graph shows the Prey Population vs. Time while the bottom graph shows the Predator Population vs. Time.

Figure 4 shows that bounded oscillatory solutions to the Lotka-Volterra equations exist over finite times. This provides some support for the model as this behavior has been observed for real biological systems, such as the Lynx-Hare populations in North America [1]. This plot also demonstrates that time-delays are not the only possible cause for oscillations in population over time.

As in the case of the logistic equation we added stochastic elements to the predator prey system by implementing the following

```
for i=1:N
  x(i+1)=x(i)+x(i)*(a - b*x(i) - c*y(i))*h
          +sigma*x(i)*sqrt(h)*randn;
  if y(i) < extinction
    y(i+1) = 0;
  else
    y(i+1)=y(i)+y(i)*(-k+lamda*x(i))*h
            +sigma*y(i)*sqrt(h)*randn;
  end
end
```

Note that, in addition to adding random noise, we have created a way for our predator to go extinct by includ-

ing an **if** statement with an extinction threshold. This is meant to reflect the physical case where there is no longer enough members of the predator species to successfully breed. The extinction threshold 0.05 was chosen after experimentation showed that extinction for this value was unlikely for small amounts of stochastic noise ($\sigma = 0.05$), but that for large $\sigma$, extinction occurred frequently enough to capture in our results. While the values of sigma we used were chosen for their mathematical convenience, showing that extinction of the predator is a possible outcome verifies that the stochastic Lotka-Volterra model might be a good fit for biological systems if the sigma parameter is chosen more judiciously.

In Figure 5 stochastic solutions to the two species predator prey equation are plotted along with the deterministic solution. Each color in this graph represents a different trial of the stochastic solution. For small fluctuations, $\sigma = 0.05$, it does not appear likely that the predator population will face extinction during the time period analyzed. However, for $\sigma = 0.15$ extinction is a very real possibility. In Figure 7, we actually see the predator denoted by the green line go extinct around time step 25. Correspondingly, the prey population of that same trial grows very rapidly after that time.

Another notable development in the high noise case is that for one of the trials the stochastic predator and prey populations get out of phase with the deterministic solutions and the other trials. These sorts of deviation is not seen in the lower noise cases and suggest that if real populations were to be models in this way, great care must be taken to choose a realistic noise coefficient along with the other parameters.

A good alternative view of the oscillatory nature of the system is seen in the phase plane solutions of Figure 6. There, the solutions for several different stochastic trials are plotted on top of the deterministic solution.

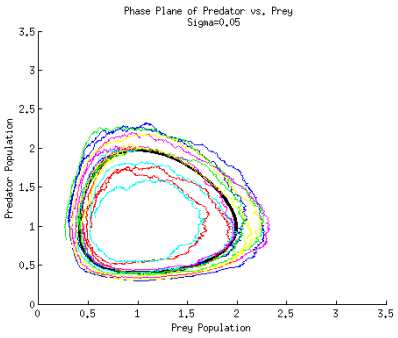



FIG. 6: Stochastic Predator-Prey Phase Plane; $\sigma = 0.05$: The phase plane solutions of the results from Figure 5. Each color on the graph represent an individual run of the test code. The deterministic solution is plotted as a bold black line. Without random fluctuations the result is a closed loop. After fluctuation are added there is still orbiting behavior, but the loops are no longer closed corresponding to the jagged oscillations of the populations in time.

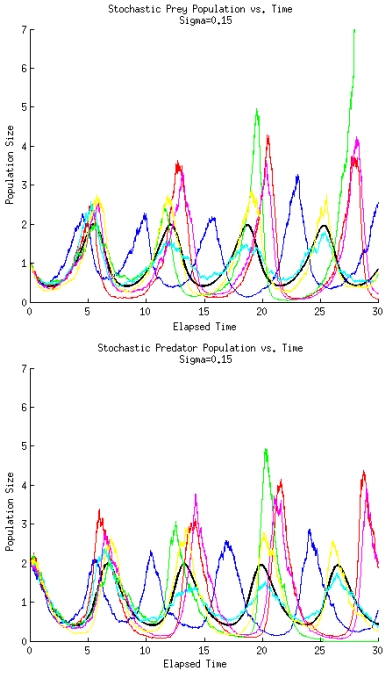



FIG. 7: Stochastic Predator-Prey Population vs. Time; $\sigma = 0.15$: Each color on the graph represent an individual run of the test code plotted on top of the deterministic solution indicated by the bold black line. For this large amount of noise wild deviations from the deterministic solution are possible. We see in the predator population (and correspondingly in the prey population) that the run represent by the blue line is completely out of phase with the deterministic solution after two periods. We also see that around time = 25 the predator population for the green run drops below the extinction threshold. As a result the prey population after this time grows dramatically.

### E. Multiple-Species Lotka-Volterra Model

We can extend the predator-prey interactions of the continuous Lotka-Volterra model simply by adding a new differential equation for a third species and establishing the parameters by which it interacts with the other species. A succinct way to write this is in the matrix notation

$$\frac{d}{dt}\begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} = \begin{bmatrix} N_1 & 0 & 0 \\ 0 & N_2 & 0 \\ 0 & 0 & N_3 \end{bmatrix} \left( \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} N_1 \\ N_2 \\ N_2 \end{bmatrix} \right) \tag{5}$$

In this formulation, the parameters $E_1, E_2$, and $E_3$ govern whether a species plays the role of a predator or prey in the system. If $E_i < 0$ then species $N_i$ is a predator, alternatively if $E_i > 0$ the species $N_i$ is a prey. It

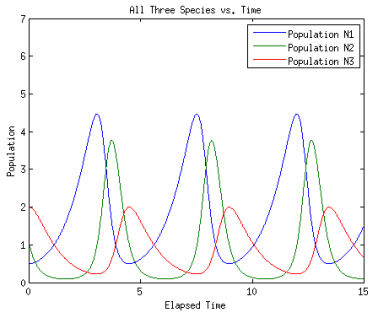



FIG. 8: Stable Solution of Two Predator One Prey System: A three species food chain where $N_1$ is preyed upon by $N_2$ who is in turn preyed upon by $N_3$ who sits at the top of the food chain. Initial population are $N_1(0) = 0.5$, $N_2(0) = 1.0$, and $N_3(0) = 2.0$. Parameter values are $E_1 = 1$, $E_2 = -1$, $E_3 = -1$, $a_{11} = 0$, $a_{12} = -1$, $a_{13} = 0$, $a_{21} = 1$, $a_{22} = 0$, $a_{23} = -1$, $a_{31} = 0$, $a_{32} = 1$, $a_{33} = 0$.

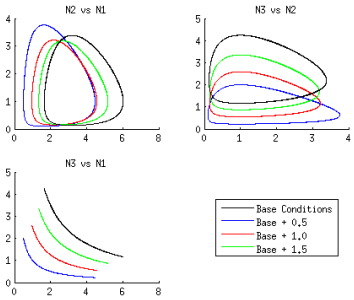



FIG. 9: Phase Plane Cross Sections of Two Predator One Prey System: A plot of the 2-dimensional phase planes of the three species food chain examined in Figure 8. Here the interaction parameters are kept constant while phase plane solutions of different initial populations are plotted. The black line represents the solution for initial population $N_1(0) = 0.5$, $N_2(0) = 1.0$, and $N_3(0) = 2.0$. All three initial populations are then incremented by 0.5 and the equations are solved again to produce the blue line. This is done again to produce the red and then green cases in accordance with the legend. Changing the initial population results in a change of shape in the phase plane

is not difficult to see that this reduces to the two-species model discussed earlier if $E_1 = a$, $E_2 = -k$, $a_{11} = b$, $a_{12} = c$, $a_{21} = \lambda$, $a_{22} = 0$, and $N_3 = 0$, noting that we no longer require $a_{ij} > 0$ as we did in the previous models.

For completeness, I will mention that we can generalize to an arbitrary number of species using the equation.

$$\frac{d}{dt}N_i = N_i(E_i + \sum_{j=1}^{j=3} A_{ij}N_j) \tag{6}$$

However, because each parameter $E_i$ and $a_{ij}$ must be individually selected, models contain more than three species quickly become very painful to work with.

Interesting solutions for the multi-species model were found for a three species food chain in which a top predator $N_3$ preys on a middle predator $N_2$ who hunts the prey $N_1$. A bounded solution for this type of system is plotted in Figures 8 and 9. Figure 8 shows the populations of each species as a function of time, and Figure 9 shows 2-Dimensional phase plane solutions of the system for different initial population values. Figure 8 is probably the best example of the Lotka-Volterra model producing results that are in line with what one would intuitively expect from a natural system. Initially, a lack of prey leads to a high death rate among both predators. This greatly increases the population of prey causing a growth in the predator population until the number of prey decreases again. The three species then continue to follow each other this way leading to closed loops in the phase plane.

If we adjust the parameters to make the top predator die off, we get the results in Figure 10. Again this results

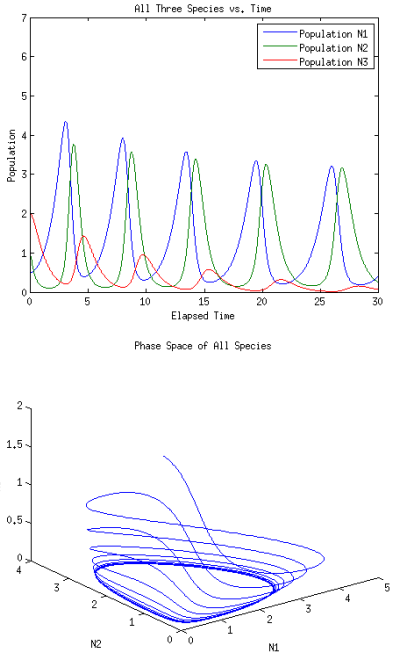



FIG. 10: Two Predator One Prey System With Dying Predator: This is a three species food chain in which the top predator dies off slowly, reducing it to the two species predator-prey equation. The plot on top shows the population of each species vs. time, while the bottom plot shows the populations in 3-Dimensional phase space. As the population of $N_3$ declines in time, the solution in phase space spirals down to a steady orbit in the $N_1, N_2$ phase plane. Initial population are $N_1(0) = 0.5$, $N_2(0) = 1.0$, and $N_3(0) = 2.0$. Parameter values are $E_1 = 1$, $E_2 = -1$, $E_3 = -1$, $a_{11} = 0$, $a_{12} = -1$, $a_{13} = 0$, $a_{21} = 1$, $a_{22} = 0$, $a_{23} = -1$, $a_{31} = 0$, $a_{32} = 0.88$, $a_{33} = 0$. Notice that only parameter $a_{32}$ has changed from the case in which all three predator coexisted without extinction.

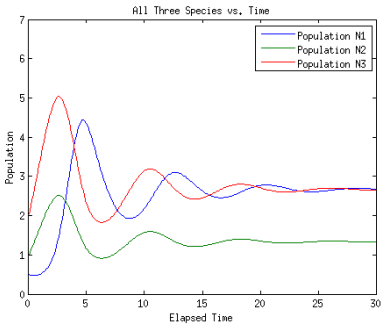



FIG. 11: Bounded Solution of One Predator Two Prey System: Initial populations are $N_1(0) = 0.5$, $N_2(0) = 1.0$, and $N_3(0) = 2.0$. Parameter values are $E_1 = 1$, $E_2 = 1$, $E_3 = -1$, $a_{11} = 0$, $a_{12} = 0.25$, $a_{13} = 0.25$, $a_{21} = -0.25$, $a_{22} = -0.25$, $a_{23} = 0$, $a_{31} = -0.25$, $a_{32} = -0.25$, $a_{33} = 0$.

of the Lotka-Volterra model agree with intuition. As the population of the top predator goes to zero, the populations of the remaining two species settles into a cycle of single-predator prey interaction. This is readily visualized in 3-Dimensional phase space. As the top predator dies off the phase plane graph settles into a 2-Dimension loop in the remaining species phase plane.

The last figure, Figure 11, shows the interaction of one predator species with two prey species. The parameters were chosen such that the predator would have no preference for one species over the other, and that both species felt the negative effects of the predator in the same way. The result is populations that initially oscillate in time, but eventually settle into a stable equilibrium.

## III. SUMMARY AND CONCLUSIONS

This paper covered a great number of scenarios while working within the Lotka-Volterra model. Many of the qualitative behaviors observed agree both intuitively with what one expects for a biological predator-prey model, and in the case of oscillatory solutions, with what has been observed in nature. We also showed the model can be refined further to reflect desired characteristics of a biological system. Particularly, random fluctuations or time-delays between resource depletion and decreased birth rate can be incorporated into the model. However, problems still remain in implementing these changes. Choosing parameters to use when running the model remains a challenge. We saw in the stochastic case that solutions are highly dependent on the amount of noise in the system. We also saw that oscillations in species population can arise in multiple ways.

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

We also thank the Texas A&M Math Department Calclab for access to Matlab software.

[1] Haberman, Richared. Introduction to Mathematical Models in Biology. In. *Mathematical Models, Mechanical Vibrations, Population Dynamics, and Traffic Flow* New Jersey: Printice-Hall Inc., 1977, p. 119-242

[2] Goodwin, R.M. , "A Growth Cycle", Socialism, Capitalism and Economic Growth, Feinstein, C.H. (ed.), Cambridge University Press, (1967)

[3] Lotka, A.J., "Contribution to the Theory of Periodic Reaction", J. Phys. Chem., 14 (3), pp 271274 (1910)

[4] M.S. Bartlett, *On Theoretical Models for Competitive and Predatory Biological Systems*. Biometrika, Vol. 44, No 1/2 pp. 27-42 (1957)

[5] P.H. Leslie and J.C. Gower, *The properties of a stochastic model for the predator-prey type of interaction between two species*. Biometrika (1960) Vol. 47, No 3/4, pp. 219

[6] Glenn Marion, *The Logistic Growth SDE* [Online Lecture Notes, Powerpoint], Biomathematics and Statistics Scotland. Availible at: http://www.bioss.ac.uk/ glenn/mathmod/stochlecture.pdf

[7] Iacus, Stefano M. Numerical Methods for SDE. In. *Simulation and Inference for Stochastic Differential Equations*. Springer, 2008

[8] Erica Chauvet*, Joseph E. Paullet, Joseph P. Previte, and Zac Walls*, (2002) A Lotka-Volterra Three-species Food Chain, Mathematics Magazine 75 243-255.

# Hubbard Model Exact Diagonalization

Wenchao Ge and Han Cai
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 23, 2014)

We study the computational techniques on exact diagonalization of one-dimensional Hubbard model(1DHM)[1, 2]. The model is reviewed for a system with $L$ sites and fixed electron numbers $N_\uparrow$ and $N_\downarrow$. The dimension of matrix form of the Hamiltonian is greatly reduced using particle and spin conservations, and translational symmetry. The ground state eigenvalues of 1DHM with different electron numbers are calculated using Lanczos algorithm.

## I. INTRODUCTION

One of the most successful descriptions of electrons in solids is the band theory. It is based on reducing many-body interactions to an effective one-body description. However, there are various situations of physical interests where the band theory fails by construction. One of the main motivations for studying the Hubbard model [3] is that it is the simplest generalization beyond the band theory description of solids.

The Hubbard model is used to describe the transitions of solids between conducting and insulating. It is the simplest model of interacting particles in a lattice involving only two terms in the Hamiltonian: a hopping term between particles and an on-site interaction of each particle. Although simple, the Hubbard model is able to capture the gross physical features of many systems characterized by more general interaction parameters. Recently, it has been an interesting model for high temperature superconductivity [4] in two and higher dimensions. This model can also be extended to describe interacting bosons in lattice such as ultracold atoms [5].

In one dimension, this model is exactly solvable using Bethe ansatz [6]. For higher dimensions, one has to seek for numerical methods. Mean field theory is a general method for many models in condensed matter theory but it is less accurate in lower dimensions since fluctuations are ignored in the method. Exact diagonalization method, on the other hand, can give essentially exact numerical results to any dimensions although extensive computer memory are used when the dimension and the lattice size increase. The basic idea of exact diagonalization is to reduce the dimension of the Hilbert space of the Hamiltonian by using symmetries and iterative methods, such as Lanczos algorithm [7]. The remarkable feature of the Lanczos algorithm is that high accuracy of ground-state eigenvalues can be obtained within a hundred iterations.

In this paper, we study the eigenvalues of 1DHM using exact diagonalization method for the lattice size up to $L = 8$. We show that the eigenvalues converge quickly under the basic Lanczos algorithm. We also correct the artificial degeneracy of the ground-state eigenvalues by using a projection Lanczos algorithm which generates an orthogonal set of Lanczos vectors. The structure of the paper is as the following: in Sec. II, we briefly introduce the Hamiltonian of 1DHM in the basis for a system with $L$ sites and fixed electron numbers $N_\uparrow$ and $N_\downarrow$ and analyze how to apply translational symmetry in Sec. III, for a further reduction of the Hilbert space dimension. We use Lanczos algorithm to calculate the ground state energy in Sec. IV.

## II. BASIS CONSTRUCTION

### A. The model

The Hamiltonian of the Hubbard model is given by

$$H = -t \sum_{<ij>,\sigma} (c^\dagger_{i\sigma} c_{j\sigma} + H.c.) + U \sum_i n_{i\uparrow} n_{j\downarrow} \quad (1)$$

which contains two parts: tight-binding Hamiltonian $H_0$ and operator of the on-site interaction D.

$$H_0 = -t \sum_{<ij>,\sigma} (c^\dagger_{i\sigma} c_{j\sigma} + H.c.) \quad (2)$$

$$D = U \sum_i n_{i\uparrow} n_{j\downarrow} \quad (3)$$

It's well known in condensed matter physics, tight-binding part can be diagonal in the Bloch basis, while on-site interaction in the Wainnier basis. Because of the non-commutativity, Hubbard Hamiltonian can't be diagonal in neither basises. In physics, tight-binding part prefers to delocalize the electron in the form of plane wave, while on-site interaction which counts the number of doubly occupied sites favours localization. The Hubbard model may be understood as arising from the competition between the two contributions. In convention, the ratio $u = \frac{U}{4t}$ is a measure for the relative contribution of both terms and is the intrinsic, dimensionless coupling constant of the Hubbard model.

### B. symmetries

To be specific, let us derive all the general concepts of basis construction for the Hubbard model on an one-dimensional chain. For a single site i, the Hilbert

space of the model (1) consists of four states,
(i)$|0\rangle$ means no electron at site $i$,
(ii)$c^\dagger_{i\downarrow}|0\rangle$ means one down-spin electron at site $i$,
(iii)$c^\dagger_{i\uparrow}|0\rangle$ means one up-spin electron at site $i$,
(iv)$c^\dagger_{i\uparrow}c^\dagger_{i\downarrow}|0\rangle$ two electron at site $i$.
Consequently, for a finite cluster of $L$ sites, the full Hilbert space has dimension $4^L$. This is a rapidly growing number, and without symmetrization we could not go beyond $L \approx 16$ even on the biggest supercomputers. Given a symmetry of the system, i.e. an operator A that commute with $H$, the Hamiltonian will not mix states from different eigenspaces of A. Therefore, the matrix representing $H$ will acquire a block structure, and we can handle each block separately. The Hubbard Hamiltonian (1) has a number of symmetries:

Particle number conservation: corresponds to global gauge transformations. $H$ commutes with total particle

$$N_e = \sum_{i,\sigma} n_{i\sigma} \tag{4}$$

Total spin conservation: corresponds to $SU(2)$ symmetery. $H$ commutes with all components of the total spin

$$S^\alpha = \frac{1}{2}\sum_i \sum_{mu,nv} c^\dagger_{i\mu}\sigma^\alpha_{\mu\nu}c_{i\nu} \tag{5}$$

where $\sigma^\alpha$ denotes the Pauli matrices.
Particle-hole symmetry: corresponds to Shiba transformation symmetry.

$$J^{sh}_\sigma = (c^\dagger_{L\sigma} - c_{L\sigma})(c^\dagger_{L-1\sigma} + c_{L-1\sigma})...(c^\dagger_{2\sigma} - c_{2\sigma})(c^\dagger_{1\sigma} + c_{1\sigma}) \tag{6}$$

it only applies for even number of lattice sites $H$
Translational invariance: assuming periodic boundary conditions, i.e., $c^\dagger_{L\sigma} = c^\dagger_{0\sigma}$, $H$ commutes with the translation operator

$$T : c^\dagger_{i\sigma} \to c^\dagger_{i+1\sigma} \tag{7}$$

Inversion symmetry: H is symmetric with respect to the inversion

$$I : c^\dagger_{i\sigma} \to c^\dagger_{L-i\sigma} \tag{8}$$

For the basis construction the most important of these symmetries are the particle number conservation and spin $S^z$ conservation. Note that that both $S^z = (N_\uparrow - N_\downarrow)/2$ and $N_e = (N_\uparrow + N_\downarrow)$ is equivalent to the conservation of the total number of spin up and spin down electrons, respectively.

### C. an example

Let us start with building the basis for a system with L sites and fixed electron numbers $N_\uparrow$ and $N_\downarrow$. Beacuase of the anticommutation relation of Fermion, we need to define an order to guarantee the uniqueness. It is convenient to first sort the electrons by the spin index, then by the lattice index, i.e.,

$$c^\dagger_{3\uparrow}c^\dagger_{2\uparrow}c^\dagger_{0\uparrow}c^\dagger_{3\downarrow}c^\dagger_{1\downarrow}|0\rangle \tag{9}$$

is a valid ordered state. This ordering ahs the advantage that the bearest-neighbor hopping in the Hamiltonian does not lead to complicated phase factors. Now we can find all the basis states: there are $\binom{L}{N_\uparrow}$ ways of distributng $N_\uparrow$ up-spin electrons on $L$ sites, and similarly, $\binom{L}{N_\downarrow}$ ways of distributing $N_\downarrow$ down-spin electrons on $L$ sites. Hence, the total number of states is $\binom{L}{N_\uparrow}\binom{L}{N_\downarrow}$. Summing up dimensions from all the $(N_\uparrow, N_\downarrow)$ blocks, we obtain

$$\sum_{N_\uparrow=0}^{L}\sum_{N_\downarrow=0}^{L}\binom{L}{N_\uparrow}\binom{L}{N_\downarrow} = 2^L 2^L = 4^L \tag{10}$$

The biggest block has dimension $\binom{L}{L/2}$. This is a considerably reduction than original dimension, in next section, we will use translational invariance to reduce the dimension by a factor of $L$

Then we need to implement the basis structure on a computer. An efficient way to represent a basis is using one integer number and bit operations. From now on and next section, we work with a lattice of $L = 4$ sites and $N_\uparrow = 3$, $N_\downarrow = 2$. We can then translate the state in (9) into a bit pattern, integer pattern as well

$$\begin{aligned} c^\dagger_{3\uparrow}c^\dagger_{2\uparrow}c^\dagger_{0\uparrow}c^\dagger_{3\downarrow}c^\dagger_{1\downarrow}|0\rangle &\to (\uparrow,\uparrow,0,\uparrow)\otimes(\downarrow,0,\downarrow,0) \\ &\to 1101\otimes 1010 \\ &\to |13\rangle\otimes|10\rangle \end{aligned} \tag{11}$$

The complete basis is gien by all 24 pairs of the four up-spin and six down-spin states, see Table I

Of course, we can further simplify the representation to combine the two indices to an overall index $n = i\cdot 2^L + j$. In my implementation, I didn't use the overall index representation.

TABLE I: Complete basis of the Hubbard model, with $L = 4$, $N_\uparrow = 3$ and $N_\downarrow = 2$.

| *no.* | $\uparrow$ *pattern* | *no.* | $\downarrow$ *pattern* |
|---|---|---|---|
| 0 | $0111 = 7$ | 0 | $0011 = 3$ |
| 1 | $1011 = 11$ | 1 | $0101 = 5$ |
| 2 | $1101 = 13$ | 2 | $0110 = 6$ |
| 3 | $1110 = 14$ | 3 | $1001 = 9$ |
| | | 4 | $1010 = 10$ |
| | | 5 | $1100 = 12$ |

### D. The Hamiltonian Matrix

Having found all basis states, now we apply the Hamiltonian (1) to basis to obtain the matrix elements, for our simple state (9) we obtain:

$$\begin{aligned} \uparrow hopping &\rightarrow -t(1011+1110)\otimes 1010 \\ \downarrow hopping &\rightarrow -t1101\otimes(0110+1100+1001-0011) \\ Uterm &\rightarrow U1101\otimes 1010 \end{aligned} \quad (12)$$

One tricky point in this part is the phase of 0011,

$$\begin{aligned} \langle 1010|-tc^\dagger_{3\downarrow}c_{0\downarrow}|0011\rangle &= -t\langle 0|c_{1\downarrow}c_{3\downarrow}c^\dagger_{3\downarrow}c_{0\downarrow}c^\dagger_{1\downarrow}c^\dagger_{0\downarrow}|0\rangle \\ &= t\langle 0|c_{1\downarrow}c^\dagger_{1\downarrow}c_{3\downarrow}c^\dagger_{3\downarrow}c_{0\downarrow}c^\dagger_{0\downarrow}|0\rangle \\ &= t \end{aligned} \quad (13)$$

The minus signs arise because of periodic boundary conditions. Whenever an electron is wrapped around the boundary and the number of electrons it commutes through is odd, i.e., the excitation number in certain spin is even. Translate these hopping into integer pattern

$$\begin{aligned} \uparrow hopping &\rightarrow -t(|11\rangle+|14\rangle)\otimes|10\rangle \\ \downarrow hopping &\rightarrow -t|13\rangle\otimes(|6\rangle+|12\rangle+|9\rangle-|3\rangle) \\ Uterm &\rightarrow U|13\rangle\otimes|10\rangle \end{aligned} \quad (14)$$

## III. USING TRANSLATION SYMMETRY

Running time of matrix diagonalization is generally scaled as $\sim M^3$, in which M is the dimension of the matrix. Because of the dependence, it is crucial to further reduce the space dimension with extra symmetry, even with the cost of more complicate coding effort.

### A. eigenstate

Unlike total spin, our previous basis is not eigenstate of translational operator $T$, therefore we introduce the projector

$$P_k = \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi ijk/L}T^j \quad (15)$$

in which $k = 0, 1, ...L-1$ For a given state $|n\rangle$ , the state $P_k|n\rangle$ is an eigenstate of T

$$TP_k|n\rangle = \frac{1}{L}\sum_{j=0}^{L-1} e^{2\pi ijk/L}T^{j+1}|n\rangle = e^{2\pi ijk/L}P_k|n\rangle \quad (16)$$

The normalization of the state $P_k|n\rangle$ requires a lot of care. We find

$$\begin{aligned} P_k^\dagger &= \frac{1}{L}\sum_{j=0}^{L-1} e^{-2\pi ijk/L}T^{-j} = \frac{1}{L}\sum_{l=0}^{L-1} e^{2\pi ilk/L}T^l = P_k \\ P_k^2 &= \frac{1}{L}\sum_{m,j=0}^{L-1} e^{-2\pi i(m-j)k/L}T^{m-j} \\ &= \frac{1}{L}\sum_{l=0}^{L-1} e^{2\pi ilk/L}T^l = P_k \end{aligned} \quad (17)$$

Hence, $\langle n|P_k^\dagger P_k|n\rangle = \langle n|P_k|n\rangle$
There is a tricky point in the translation operator, in some cases, we need to attach a phase because of periodic boundary condition and anticommutation relation of Fermion, for example

$$\begin{aligned} T|1100\rangle_\uparrow &= Tc^\dagger_{3\uparrow}c^\dagger_{2\uparrow}|0\rangle \\ &= c^\dagger_{0\uparrow}c^\dagger_{3\uparrow}|0\rangle = -c^\dagger_{3\uparrow}c^\dagger_{0\uparrow}|0\rangle = -|1001\rangle_\uparrow \end{aligned} \quad (18)$$

### B. example

Let us consider the ↑ pattern from I: All the 4 states are connected with a translation by one site.

$$\begin{aligned} |0\rangle &= T^0|0\rangle = 0111 \\ |1\rangle &= T^{-1}|0\rangle = 1011 \\ |2\rangle &= T^{-2}|0\rangle = 1101 \\ |3\rangle &= T^{-3}|0\rangle = 1110 \end{aligned} \quad (19)$$

Applying the projector to the representative of the cycle, $P_k|0\rangle$, we can generate $L$ linearly independent states, which in our case reads

$$\begin{aligned} P_0|0\rangle &= (0111+1011+1101+1110)/L \\ P_1|0\rangle &= (0111-i1011-1101+i1110)/L \\ P_2|0\rangle &= (0111-1011+1101-1110)/L \\ P_3|0\rangle &= (0111+i1011-1101-i1110)/L \end{aligned} \quad (20)$$

The advantage of these new states, which are linear combinations of all members of the cycle in a spirit similar to discrete Fourier transformation, becomes clear when we apply the Hamiltonian: Whereas the Hamiltonian mixes the states in (19), all matrix elements between the states in (20) vanish. Hence, we have decomposed the four-dimensional Hilbert space into four one-dimensional blocks.
In the next step, we repeat this procedure for the ↓ patterns of I. There is two permutation group, $S_2$ and $S_4$, in ↓ while only one permutation group, $S_4$, in ↑.

$$\begin{aligned} |3\rangle, |6\rangle, |12\rangle, |9\rangle &\in S_4 \\ |5\rangle, |10\rangle &\in S_2 \end{aligned} \quad (21)$$

To get the complete symmetrized basis, we need to combine the up and down spin representatives, thereby taking into account relative shifts between the states. For our sample case the combined representatives, in each blocks,

$$\begin{aligned} &P_k(0111 \otimes 0011)\\ &P_k(0111 \otimes 0110)\\ &P_k(0111 \otimes 1100)\\ &P_k(0111 \otimes 1001)\\ &P_k(0111 \otimes 0101)\\ &P_k(0111 \otimes 1010) \end{aligned} \tag{22}$$

in general, we need to combine all the permutation groups one by one from two spins and translate $\downarrow$ spin by $\nu$ times, $\nu$ is minimum number between group ranks of two combined permutation,

$$|r\rangle = |n_\uparrow\rangle\, T^j\, |m_\downarrow\rangle \tag{23}$$

with $j = 0, 1...\nu$, as in (22). The basis of each of the $L$ fixed $k$ Hilbert spaces is given by the states

$$|r_k\rangle = \frac{P_k\,|r\rangle}{\sqrt{\langle r|\, P_k\,|r\rangle}} \tag{24}$$

then the matrix elements between two states $|r_k\rangle$ and $|r'_k\rangle$ is simply given by

$$\begin{aligned} \langle r'_k|\, H\,|r_k\rangle &= \frac{\langle r'|\, P_k H P_k\,|r\rangle}{\sqrt{\langle r'|\, P_k\,|r'\rangle\,\langle r|\, P_k\,|r\rangle}}\\ &= \frac{\langle r'|\, P_k H\,|r\rangle}{\sqrt{\langle r'|\, P_k\,|r'\rangle\,\langle r|\, P_k\,|r\rangle}} \end{aligned} \tag{25}$$

Repeating the procedure for all representatives, we obtain the matrix for a given k, Th =e full matrix with fixed particle numbers $N_\uparrow$ and $N_\downarrow$ is decomposed into $L$ blockees with fixed $k$. the $24\times 24$ matrix from I is decomposed into the four $6\times 6$ matrices.

### C. A few Remarks

In III B, we encounter an easy case, $4(\uparrow)\times 6(\downarrow) \to 24 \to 4(k)\times 6$ . Let us see some more complicated case. For $L = 4$ , $N_\uparrow = 2$ and $N_\downarrow = 2$, within each blocks we have $S_4\otimes S_4 + S_4\otimes S_2 + S_2\otimes S_4 + S_2\otimes S_2 = 10$, as well as 4 states with

$$\langle r|\, H\,|r\rangle = 0 \tag{26}$$

in which $|r\rangle$ is

$$\begin{aligned} &P_1(0101 \otimes 0101)\\ &P_1(0101 \otimes 1010)\\ &P_3(0101 \otimes 0101)\\ &P_3(0101 \otimes 1010) \end{aligned} \tag{27}$$

Certainly, we need to discard the non-normalizable state, hence we find $10+8+10+8 = 36 = \binom{4}{2}\binom{4}{2}$

More difficulty appears when $L$ is larger, when $L = 6$, $N_\uparrow = 2$ and $N_\downarrow = 3$, $S_3\otimes S_2$ is supposed to have 6 states,

$$\begin{aligned} &001001 \in S_3\\ &010010 \in S_3\\ &100100 \in S_3\\ &010101 \in S_2\\ &101010 \in S_2 \end{aligned} \tag{28}$$

while 12 projected states

$$\begin{aligned} &P_k(001001 \otimes 010101)\\ &P_k(001001 \otimes 101010) \end{aligned} \tag{29}$$

have non-zero normalization. So far we have no idea how to eliminate the redundant states, hence in the implementation, we switch off the translational symmetry function when the dimension doesn't match up.

## IV. EXACT DIAGONALIZATION

In this section, we will discuss the details of solving eigenvalues of the block matrix via the Lanczos algorithm and QR reduction algorithm. The Lanczos algorithm will reduce the original matrix to a tridiagonal matrix with dimension equals to the number of iterations. Then, the eigenvalues can be obtained with QR algorithm on the tridiagonal matrix.

### A. The basic Lanczos algorithm

We first introduce the basic Lanczos algorithm which converges quickly after several dozens of iterations. Starting with a $n\times n$ matrix $A$ of interest and a random unit vector $v_1$, the recurrence of the Lanczos vectors are such that

$$\beta_{j+1}v_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}, \tag{30}$$

where $\alpha_j = v_j^* A v_j$, $\beta_{j+1} = v_{j+1}^* A v_j$, and $\beta_1 = 0$. The recurrence generates an orthonormal set of Lanczos vectors and a tridiagonal matrix defined as

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}. \tag{31}$$

This method is powerful when $m$ steps are carried with $m \ll n$. Then the following relation holds,

$$AV_m - V_m T_m = \beta_{m+1} v_{m+1} e_m^*, \tag{32}$$

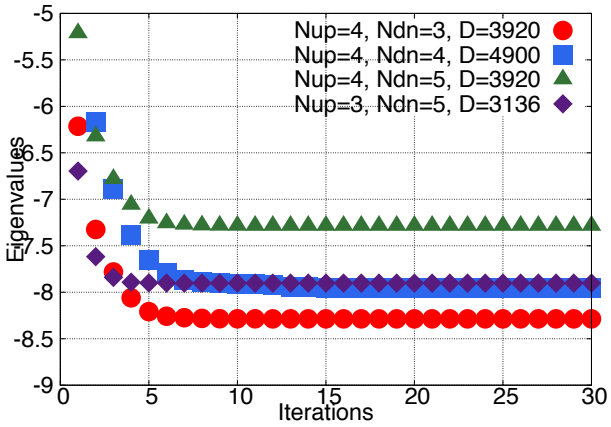



FIG. 1: Ground-state eigenvalues for $L = 8$ with different numbers of spins.

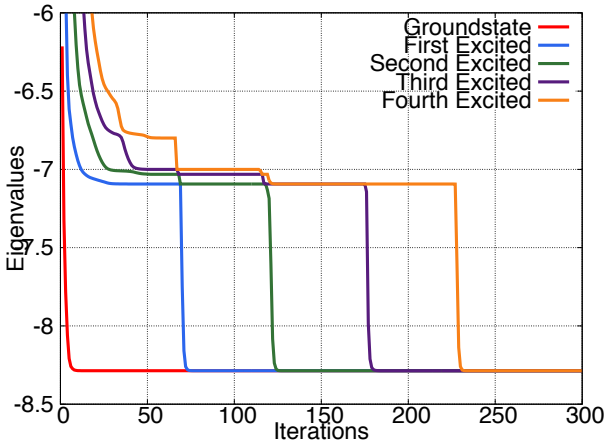



FIG. 2: Five-fold artificial degeneracy of $L = 8$, $N_\uparrow = 4$, and $N_\downarrow = 3$ using the basic Lanczos algorithm.

where $V_m = [v_1, v_2, \cdots, v_m]$ and $e_m$ is the unit vector of $m$ dimension. Once the tridiagonal matrix is obtained, the eigenvalues can be calculated with QR reduction algorithm. Practically, this method is handled by the GSL library we used. Therefore, we focus on how to obtain the tridiagonal matrix with Lanczos algorithm.

With the above algorithm, we study the ground-state eigenvalues of the largest four dimensions with the lattice size $L = 8$. We observe from Fig. 1 that for the eigenvalues converge quickly within 30 iterations for dimensions larger than 3000. This quick convergence makes the algorithm powerful. For higher lattice size, the eigenvalues will converge within one hundred iterations [1]. However, as the lattice size increases, much greater computer memory will be required to initialize the original matrix $A$. Therefore, our study provides a proof-of-principle example of the exact diagonalization method.

### B. The projection Lanczos algorithm

We discussed the basic Lanczos algorithm which shows quick convergence on the ground-state eigenvalues. With a careful examination on the next several eigenvalues of a system with given spin numbers, we find that high degeneracy in these eigenvalues as the number of iterations increases. As shown in Fig. 2, we study the first five eigenvalues of the model with $L = 8$, $N_\uparrow = 4$ and $N_\downarrow = 3$ using the basic Lanczos algorithm. With about 50 iterations, it seems that the eigenvalues have reached their respective steady values. However, as the iterations increase, the initial non-degenerate eigenvalues become degenerate. This is due to the loss of orthogonality of Lanczos vectors in as the number of iterations increases. Since we implemented in finite precision arithmetic, the Lanczos algorithm does not guarantee the orthogonality of the vectors. To clear the artificial degeneracy of the

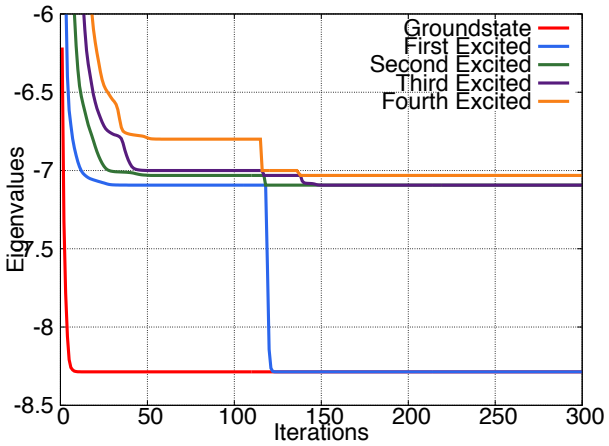



FIG. 3: Two-fold ground-state degeneracy of $L = 8$, $N_\uparrow = 4$, and $N_\downarrow = 3$ using the projection Lanczos algorithm.

basic Lanczos algorithm, we apply the projection Lanczos algorithm which orthogonalizes the vector $v_{j+1}$ with respect to all previous vectors $v_1$ to $v_j$. This method takes larger computer memory to store the vectors than the previous method, where $v_{j+1}$ is orthogonal to $v_j$ by construction.

We plot again the first five eigenvalues of the same system by using the projection Lanczos algorithm in Fig. 3. It is shown that with enough iterations the first five eigenvalues split into three different ones. The five-fold artificial degeneracy is reduced to two-fold. We compare the results of the projection Lanczos algorithm with the analytical values.

Then we use QR reduction algorithm, packaged in GSL library, to calculate the ground state respectively. We find two-fold ground-state degeneracy, $E_g =$

−8.2861179292, which matches up with the projection Lanczos algorithm.

## V. SUMMARY AND CONCLUSIONS

In this paper, we studied numerical algorithms on exact diagonalization of one-dimensional Hubbard model. We reduced the system of $L = 8$ lattice size with conservation and symmetries. Two variations of Lanczos algorithms are applied on the reduced matrix to obtain the convergence of the algorithms and correct spurious results. The basic Lanczos algorithm is fast to converge but the vectors lose their orthogonality as enough iterations are carried out. The projection algorithm is more consuming on the computer memory but provides with more reliable excited eigenvalues.

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] A. Weiße and H. Fehske, *Exact diagonalization techniques*, in *Computational Many-Particle Physics* (Springer, Berlin Heidelberg, 2008), vol. 529-544.
[2] F. H. L. Essler, H. Frahm, F. Göhmann, A. Klümper, and V. E. Korepin, *The one-dimensional Hubbard model* (Cambridge University Press, 2005).
[3] Hubbard, , J. Proc. Roy. Soc. London **A 276**, 238 (1963).
[4] E. Dagotto, , Rev. Mod. Phys. **66**, 763 (1994).
[5] D. Jaksch, C. Bruder, J. Cirac, C. Gardiner, P. Zoller, , Phys. Rev. Lett. **81**, 3108 (1998).
[6] E. H. Lieb, F. Y. Wu, , Phys. Rev. Lett. **20**, 1445 (1968).
[7] Paige,C. C., , J. Applied Mathematics **10**, 373 (1972).

# Traveling Salesman Problem Using Simulated Annealing

Caitlin Campbell
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 26, 2014)

We study the optimization problem of the traveling salesman using simulated annealing. We test the linear, logarithmic, and exponential annealing schedules along with different initial conditions to observe how the speed of the cool down along with the starting parameters affect the performance of the algorithm. The number of samples and number of Monte Carlo sweeps are varied to examine the influence these factors have on the program. We run each schedule on the 50 city tour numerous times to find the most efficient algorithm, or the approach that gives the optimal length the most often. We also test these different schedules with tours with larger numbers of cities to show that the success probability of the algorithm diminishes as city size increases.

## I. INTRODUCTION

The Traveling Salesman Problem is a widespread problem in computer science and optimization problems. The problem was defined in the 1800s by the mathematicians W. R. Hamilton and Thomas Kirkman [1]. It is a NP-hard problem where a traveling salesman needs to visit each of N cities only once and return to his original city, and he wants to find the order of cities to visit with the shortest distance. The traveling salesman problem has applications in transportation routing, scheduling, packing, production management, and placement of wires on integrated circuits. It also has applications as a subproblem in DNA sequencing finding similarity between DNA fragments [2, 3].

## II. MODEL

Simulated annealing, named for its resemblance to the thermodynamics annealing process of heating a metal and slowly cooling it for particles to reach their minimum energy state, was the optimization method used to solve this problem. The program begins with initial conditions, and it is slightly perturbed with each iteration. If the resulting solution is improved it will replace the previous solution. The downside to using simulated annealing is the algorithm is easily trapped in a metastable state with energy corresponding to a local minimum and once trapped won't reach the true minimum. This problem is corrected by using a temperature factor. For every move that results in a better solution, the move will be accepted. For every move that results in a worse solution, the move will be accepted with a probability dependent on temperature. In this manner, the solution can jump out of low minima at high temperatures to reach the true minimum. At low temperatures, the probability of moving to a worse solution decreases and it is more likely to be stuck in minima so it is best to start at a high temperature and slowly cool. In the specific example of simulated annealing applied to the traveling salesman problem, the cost function to be minimized is the length of the tour of cities. The program begins with an arbitrary initial path between the cities and a given initial temperature. At each temperature a number of Monte Carlo updates are run. At each update the length is calculated as the sum of all the distances between cities. In this case it is assumed that each city is connected to another by its Euclidean distance. A move is proposed by randomly selecting two cities to be swapped in the tour. The energy of this move is computed as the difference of the length of the existing tour from the length of the proposed new tour. If the energy is negative, or the length of the new tour is shorter than the previous tour, then the move is accepted. If the energy is positive, or the new tour is longer than the previous, the move is accepted with a probability of

$$p = \exp(-\frac{\Delta E}{T}) \tag{1}$$

After all the updates are finished, the system continues to the next temperature where more Monte Carlo updates are run, and the program persists until the temperature has decreased to 0. The physical analogy of simulated annealing is a material is heated and cooled into a solid state again. Slow cooling leads to less defects and lower energy while faster cooling yields a higher energy. If it cools too quickly and does not spend enough time near the freezing point the process may be stuck in non-equilibrium states. This is also true for our simulated annealing program. In the physical analogy, the material can be viewed as a system of particles, and our results can be interpreted like the statistical mechanics at a given temperature. These particles will change energy with temperature the probability of an increase in energy is

$$p = \exp(-\frac{\Delta E}{kT}) \tag{2}$$

where k is the Boltzmann constant. In this physical analogy the distribution of states is given by the Boltzmann distribution

$$\frac{N_j}{N} = \exp(-\frac{\Delta E}{T}) \tag{3}$$

By using the Metropolis algorithm for acceptance of worse moves, the program finds thermal equilibrium at

each temperature, or sample. Because we are using this criterion for accepting a move, our final distribution of states (in this case it will be tour lengths) will be a Boltzmann distribution. [4] A system with temperature reduced infinitely slowly will converge to the true minimum. This however is not practical, and there are three typical cooling schedules where temperature is decreased linearly, logarithmically, and exponentially. The type of cooling schedule as well as the initial conditions imposed will determine the different temperatures the system will be evaluated at as well as the amount of different temperatures. The number of Monte Carlo updates, or sweeps, is also a factor in the efficacy of the program. For example, efficient results could be found with a lower initial temperature (meaning smaller amount of different temperatures) if more Monte Carlo updates were run per temperature. It is very important to do many updates at lower temperatures because low temperatures do not do well getting out of local minima so the number of sweeps should increase as the temperature decreases.

## III. METHODS

### A. Linear Schedule

The linear schedule decreases temperature with

$$T(t) = T_{initial} - bt \tag{4}$$

where $T_{initial}$ is the initial temperature, $b$ determines the speed of cooling, and $t$ is the time step of the cooling process. Our method specifically used an initial temperature of 20 with $b = 0.1$ and therefore 200 iterations to cool to $T = 0$. The number of Monte Carlo sweeps increased linearly with the temperature decrease. The number of updates was given by

$$sweeps = 10000 * t \tag{5}$$

Therefore the number of samples was 200 with 10000 sweeps in the first sample and linearly increasing from there.

### B. Logarithmic Schedule

The logarithmic schedule decreases temperature by

$$T(t) = \frac{T_{initial}}{b + log(t)} \tag{6}$$

where $T_{initial}$ is the initial temperature, and $t$ is the time step of the cooling process. This is a slower cooling process so the number of sweeps was given by

$$sweeps = 5000 * t \tag{7}$$

With a greater number of samples, less sweeps were needed per sample. For our logarithic schedule we used $b = 1$, $T_{initial} = 50$, and 635 samples.

TABLE I: For each cooling schedule run on the 50 city tour, the number of samples and amount of sweeps per sample varied while each program ran the same number of total sweeps.

| $CoolingSchedule$ | $NumberofSamples$ | $NumberofSweeps$ |
|---|---|---|
| $Linear$ | 200 | $10000 * t$ |
| $Logarithm$ | 635 | $1000 * t$ |
| $ExponentialOne$ | 20 | $75075 * t^2$ |
| $ExponentialTwo$ | 45 | $200000 * t$ |
| $ExponentialThress$ | 21 | $1000000 * t$ |

### C. Exponential Schedule

The exponential schedule decreases temperature by

$$T(t) = T_{initial} * exp(-bt) \tag{8}$$

where $T_{intial}$ is the initial temperature, $b$ determines the speed of cooling, and $t$ is the time step in the cooling process. It was trickier to find the initial conditions of the exponential schedule that would lead to successful results than the other schedules. Three different exponential schedules were tried with varying rates of success. For exponenial set one, $T_{initial} = 80$ and $b = 0.8$. The number of of sweeps per sample was given by

$$sweeps = 75750 * t^2 \tag{9}$$

The number of sweeps was greatly increased because only 20 samples were ran. For exponential set two, $T_{initial} = 120$ and $b = 0.8$. The number of sweeps for each of the 45 samples was given by

$$sweeps = 200000 * t \tag{10}$$

For exponential set three $T_{initial} = 120$ and $b = 0.95$. The number of sweeps for each of the 21 samples was given by

$$sweeps = 1000000 * t \tag{11}$$

To ensure the comparison across all cooling schedules systematic, each schedule ran through the same total number of sweeps througout the program. However, the number of samples and increase in sweeps per sample varied greatly as seen in Table 1.

## IV. RESULTS

Starting with a 50 city random initial tour with length of 5450 steps, we computed the optimum tour length to be 1147.3151. This optimized tour is shown in Fig. 1. Each cooling schedule was run hundreds of times so that success rates of the programs could be calculated. One determination of the success of the program is the percent of runs the program yielded the exact minimum. A second measure of the success is the percent of runs the

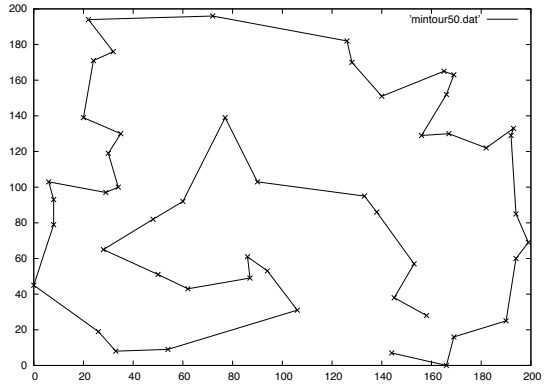



FIG. 1: This is the optimized tour of the 50 cities.

program yielded a tour that had a length within 5% of the exact minimum. The two measures of success for each of the linear, logarithmic, and exponential schedules are given in Table 1. The histograms of each cooling schedules (Fig. 2, Fig. 3, Fig. 4, Fig. 5, Fig. 6 ) show the distribution of tour lengths over several runs of the same program. The logarithmic annealing schedule was the schedule that found the true minimum the most. The exponential schedules used were not very successful at finding the minimum length of the tour compared to the linear and logarithmic schedules. In the histograms, one can see the metastable states or local minima that the program gets stuck inside. For example, the program sometimes returned the optimal length to be 1147.93 or 1447.99. The logarithmic program is more likely return the true minimum and not get stuck in a metastable state. This follows from and supports the work done by Geman and Geman in 1984 that showed the logarithimic cooling schedule works best. The logarithmic cooling schedule allows for several samples in the higher temperatures while still spending plenty of time at the temperatures close to freezing point [5]. The run time is 3 minutes and 13 seconds for the linear schedule, 3 minutes and 13 seconds for the logarithmic, 2 minutes and 56 seconds for exponential one, 3 minutes and 7 seconds for exponential two, and 3 minutes and 19 seconds for exponential three.

The minimum length converges to the optimal solution as the temperature decreases throughout the program. Fig. 7 shows the different lengths the program yieled at each step in the cooling process in the 50 city tour. Each schedule was run multiple times, and the run that led to the best solution was chosen. The logarithmic schedule has the greater number of samples with less sweeps per sample. Because it starts a much higher temperature the first few minimum lengths returned are very inaccurate, but it converges to the global minimum as the temperature continues to decrease. The linear and the

TABLE II: For each cooling schedule run on the 50 city tour, we calculated the percent of runs that the program found the true minimum, and the percent of runs that found the tour length to be within 5% of the true minimum, or less than 1204.686.

| $Cooling Schedule$ | $\%of Runs Exact$ | $\%of Runs < 1204.686$ |
|---|---|---|
| $Linear$ | 78.935 | 100 |
| $Logarithm$ | 97.540 | 100 |
| $Exponential One$ | 0.804 | 61.260 |
| $Exponential Two$ | 0.810 | 55.466 |
| $Exponential Three$ | 4.551 | 80.494 |

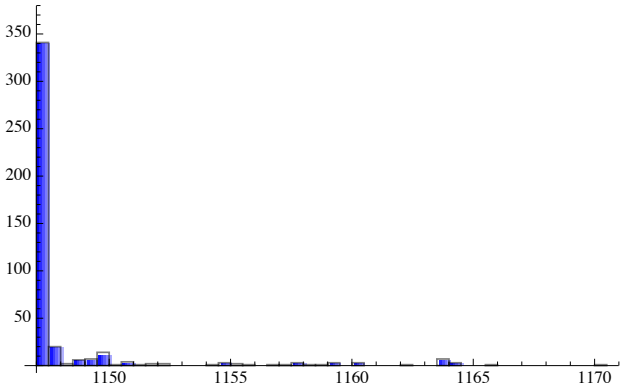



FIG. 2: Histogram of tour lengths of the 50 city tour found using the linear cooling schedule.

exponential schedules both seem to converge at around the same temperature, but the exponential schedule ends in a local minimum not the exact optimum.

In looking at the different cooling schedules one can see the strong effect initial conditions can have on the performance of simulated annealing. Higher initial temperature gives random starting points, but more samples to converge to the minimum. The best performance came from exponential set three with a high initial tempera-

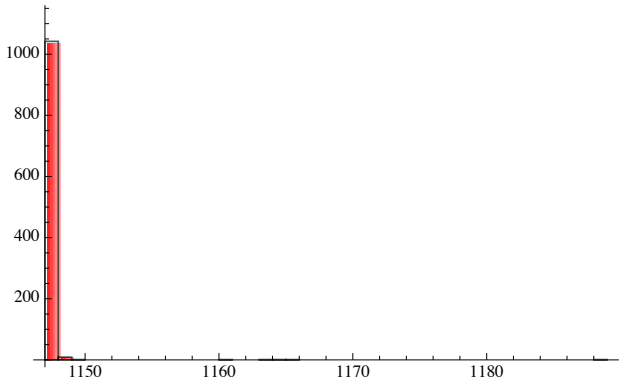



FIG. 3: Histogram of tour lengths of the 50 city tour found using the logarithmic cooling schedule.

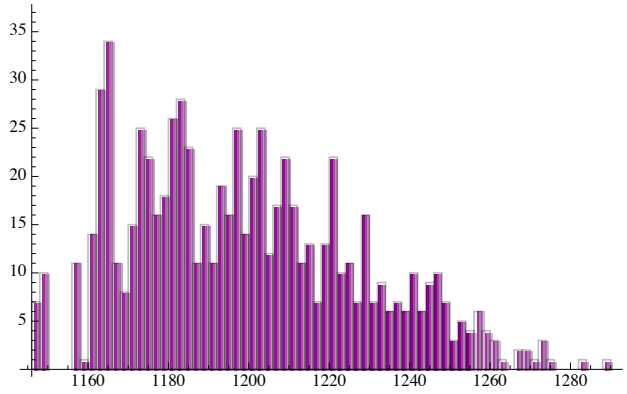



FIG. 4: Histogram of tour lengths of the 50 city tour found using the exponential one cooling schedule.

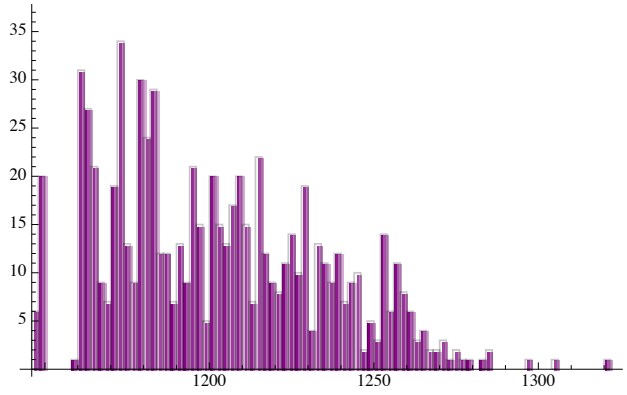



FIG. 5: Histogram of tour lengths of the 50 city tour found using the exponential two cooling schedule.

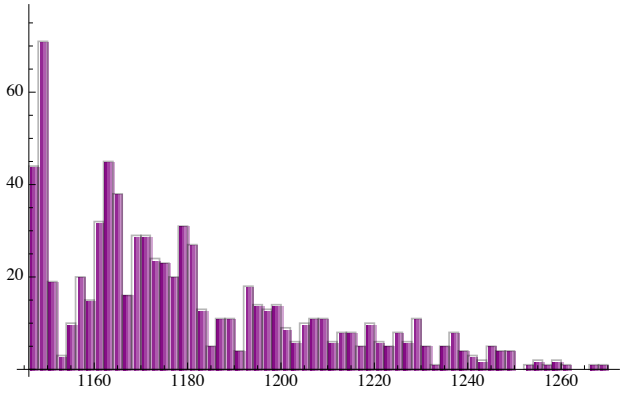



FIG. 6: Histogram of tour lengths of the 50 city tour found using the exponential three cooling schedule.

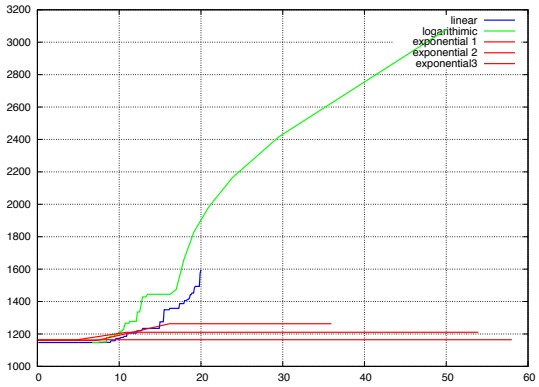



FIG. 7: This is the graph of the convergences of the minimum tour length as a function of the temperature comparing linear, logarithmic, and exponential cooling schedules.

ture, smaller number of samples, and higher number of sweeps per sample (Table II). In looking at the convergence of the minimum length as a function of temperature we see that the three exponential schedules all fell into different metastable states along the way and ended in different metastable states with no schedule finding the true minimum tour (Fig. 6).

All the previous runs were using the 50 city tour. As the number of cities on the tour to be otimized increases, the program is less successful and the probability of the algorithm returning the optimum tour diminshes to zero. Using the logarithmic schedule on a 75 city tour, the optimal tour length is still reached numerous times, but not nearly at the rate the 50 city tour was reaching (Fig. 7). As the tour length is increased to 400 cities, the algorithm certainly still optimizes the solution by taking the original tour length of about 40,000 steps to around 4900 steps. The number of runs that reaches the lowest tour length is very small compared to the other lengths found though, and the shortest tour found by this program is still not necessarily the optimal tour. In Fig. 8 the Boltzmann distribution of the density of states (tour lengths) is clearly viewed. This is due to the fact that the probability of a move out of a metastable state is given by the Boltzmann factor. The larger the number of cities, the more likely the algorithm will be stuck in one of these metastable states so the Boltzmann distribution is more visible, seen especially well in the 75 city tour distribution.

## V. SUMMARY AND CONCLUSIONS

Simulated annealing proves a suitable method to determine the shortest path between a set of cities for a reasonable number of cities. We found that the logarithmic schedule was the most optimal code for the 50 city tour, with the greatest percentage of runs finding the exact minimum. For finding the global minimum (absolute shortest route) the logarithmic and linear schedules both

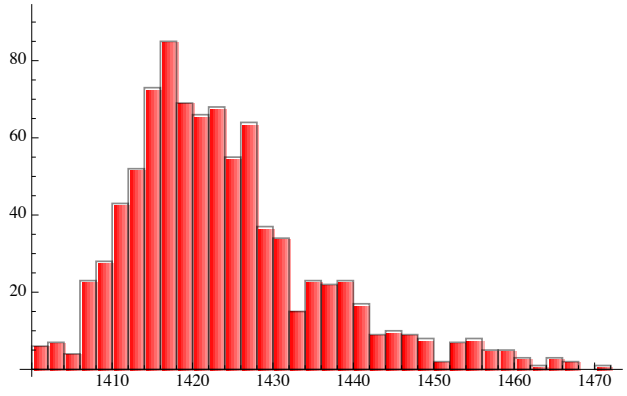



FIG. 8: Histogram of tour lengths of the 75 city tour found using the logarithmic cooling schedule.

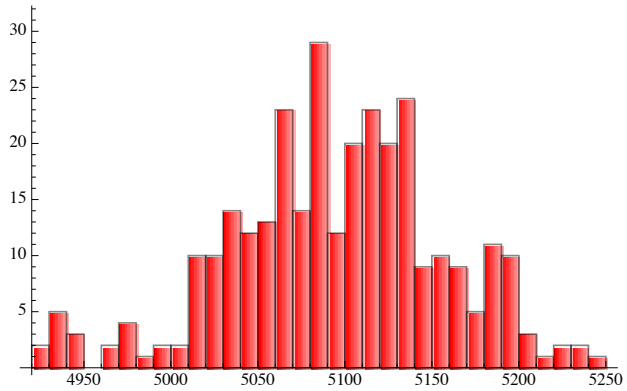



FIG. 9: Histogram of tour lengths of the 400 city tour found using the logarithmic cooling schedule.

worked reasonably well. For finding a minimized route, but not the exact shortest, the logarithmic and linear schedules both worked extremely well for 50 cities. In the tuning of the exponential schedules we found that greater sweeps with fewer samples, as in exponential scheudle three, gave better results. As the number of cities grows, the probability the program will find the optimal path decreases to zero. The calculations found here could be improved with slower annealing schedules and more sweeps per sample. A global minimum could be found by running a shorter program, one with less sweeps and samples, many times over and comparing the resulting optimized tours to find the shortest. This could be counterbalanced by running less tours that are more successful but take more running time. A balance between these two methods could be explored to find the fastest method to find the absolute minimized tour for a reasonable number of cities.

**Acknowledgments**

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] S. Nasini, Tech. Rep., Universitat Politecnica de Catalunya (2013).
[2] D. S. Johnson, Tech. Rep., ATT Labs (1995).
[3] S. Lin, Tech. Rep., The Bell System Technical Journal (1965).
[4] URL `http://www.malcolmmclean.site11.com/www/SimulatedAnnealing/SimulatedAnnealing.html`.
[5] D. G. Stuart Geman, *Stochastic relaxation, gibbs distributions, and the bayesian restoration of images*, IEEE Trans. Pattern. Analy. Mach. Intell. **PAMI-6**, 721 (1984).

# Self Organized Criticality Using 3D Random-Field Ising Model

Alex Gary
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 23, 2014)

We study the possibility and legitimacy of self-organized criticality being conveyed in a particular class of computational models. Self-organized criticality is a commonly occurring natural phenomenon that has drawn much research interest. Following research of past decades to computationally model and display self-organized criticality to understand this complexity commonly found in nature, different models have been proposed and tested. While the goal is to find a model that does not require tuning, many early models failed to produce such results. This work focuses on demonstrating whether or not the random-field Ising model will possibly fit the criteria of such a system. Nearly all research using random-field Ising Models involved tuning the standard deviation of the random field in order to display the power law which is expected to occur if a system displays self-organized criticality. Using relatively small system sizes we were able to present evidence that the random-field Ising Model does fail to produce the desired characteristics of a system that displays self-organized criticality. Using major trends and comparing them to earlier research we have shown that the R value does require tuning in order to display the power law for its distribution. Even though an exact value for the standard deviation of the critical point where the power law is displayed could not be verified due to finite-size effects, the results of previous work are confirmed by extrapolation through studying size dependence. Altogether we were able to increase the legitimacy of previous research to give further proof to the fact that the random-field Ising Model is an inadequate system to display self-organized criticality

## I. INTRODUCTION

Self-organized criticality (SOC) is the tendency of large dispersed systems to drive themselves into critical states without any required or special parameter tuning [1]. Fractal systems that display SOC are numerous in nature [2] and include but are not limited to earthquakes, the structure of dried-out river beds, the meandering of sea coasts, and the structure of galactic clusters.

With such a commonly occurring phenomenon, understanding the origin of SOC has become a major point of research in order to get to the root of the mystery of these and other natural processes. The difficulty of modeling such systems comes from the fact that in most compuational equilibrium systems critical behavior is only found at a critical value of at least parameter. This difficulty has led to decades of pioneering work in studying computational models that can be tuned to display SOC.

Beginning with dynamical systems including the sandpile [3] and forest-fire models [4], the hunt for a computational model that fits the description of a system that displays SOC has been happening since the 1980's. The sand pile model, one of the first examples found to display SOC, essentially says that if you piled sand grains randomly, one at a time, they would eventually form a pile that reached a critical slope. Once this critical slope is achieved an avalanche would occur. It has been shown that without tuning, these phenomenon occur according to the power law, or scale invariant behavior. That is, the distribution of the avalanche sizes is not limited to a characteristic size. The basic idea of the sandpile model can be seen in Figure 1.

Models of such examples were major first steps in developing questions that could later be answered. Whether the model was a feature of high-dimensional models, models with a diverging number of neighbors, long-range interactions, or is it simply a property of a large class of models all had to be answered. The work of this paper is to reaffirm some of the work related to the earliest model, the random-field Ising Model. Most of the early work done in the earliest finite-dimensional models has been shown to be inadequate when it comes to displaying SOC. Many required tuning of at least one parameter, an obvious flaw to displaying SOC. In early research with random field Ising models it was found that larger systems are crucial to extracting accurate values of the universal critical exponents and understanding important qualitative features of physics [5]. Due to limitations in code and time this work only studies the 3D Ising Model at zero temperature with dimension length up to 50 spins. Previous initial work done using the random-field Ising Model has shown that despite the attempt to use a random field to mimic some of nature's random characteristics and gaussian behavior the system did not display SOC. As the distribution of the standard deviation of the random field varied so did the distribution of the avalanche sizes. Once tuned to a specific value $R$ will cause the model to have a sclae invariant distribution of avalanches. This work tests the validity of this claim in order to see if the random-field Ising Model might still possibly serve as a system that displays SOC. This is done by providing an algorithm that flips a single neighboring spin once a first spin flips. This continues until the avalanche ends or all spins have been flipped.

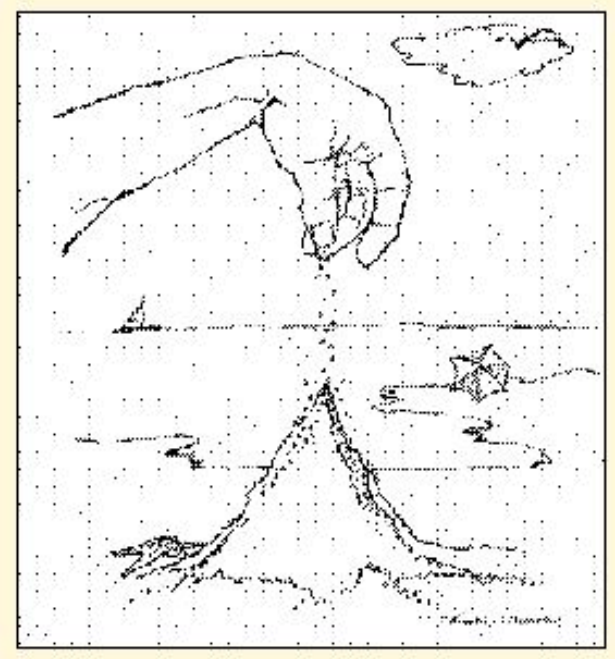


FIG. 1: Although it is a very simple example, the sandpile model is one of numerous examples of self-organized criticality found in nature. As sand is piled up randomly, certain points will eventually reach a critical slope that will then cause an avalanche. The distribution of the avalanche sizes is nearly always scale invariant. This mean that a avalanche of any size may occur. Sand on the sea shore, earthquakes, and galactic clusters all display this unique phenomenon. Such a unique phenomenon has led to a hunt for a mathematical and computational model that displays this behavior in order that the SOC character of nature can be better understood.

## II. MODEL AND ALGORITHM

The Hamiltonian of the zero temperature random-field Ising Model in an external field is given by the equation

$$\mathcal{H}(\{s_i\}) = -\sum_{ij} J_{ij} s_i\, s_j - \sum_i (f_i s_i + H s_i) \qquad (1)$$

where the Ising spins $s_i \in \{\pm 1\}$ lie on the vertices of a scale-free graph with $N$ or $L^3$ sites. For our purposes this model simply adds the ingredient of disorder to the idealized physics picture. Throughout our algorithm the external field H is changed. This change over time causes an individual spin to flip when the local field changes sign. When flipped, the local field of all neighbor spins change. Because of the field change, these neighbors now have the possibility to flip. For our purposes, only the "most extreme" spin is flipped. The process continues, forming an avalanche, until all neighbors are either previously flipped or inelligible. The local field of a spin is given by the equation

$$F_i = \sum_j J_{ij} s_j + f_i + H. \qquad (2)$$

In a d dimensional system of $d > 1$, the number of neighbors is $dx2$. The probability of a neighbor flipping for a given random field distribution is directly dependent on the number of neighbors and their spins. [5] Due to issues with our code, a dimension length of 50 spins was our maximum. While still not large enough to avoid finite-size effects it was still enough to uncover the basic trends that were needed to compare to previous work. In order to get a system to perform such flips a recursive function was used:

```
void FLIPSPIN(long int s_i, double spins[]...)
     if(LocalField[s_i]<0){
     spins[s_i]=-1;
       (*avalanchesize)++;
       (*totalflips)++;
       for(i=0; i<6; i++){
       for(q=0;q<6; q++){
           LocalF[nb[i][s_i]]=0;
           LocalF[nb[i][s_i]]+=spins[ ];
       }
           LocalF[nb[i][s_i]]+=(Rand[ ]+H);
          if(small==-1 && spins[nb[i][s_i]]>0
&& LocalField[nb[i][s_i]]<0){
              small=nb[i][s_i];
//'small' can be understood as
the 'most extreme' neighbor spin.
           }
           if(spins[nb[i][s_i]]<0)
              continue;
           if(spins[nb[i][s_i]]==1 &&
LocalF[nb[i][s_i]]<LocalF[small]){
              small=nb[i][s_i];
           }
       }
       LocalF[small]<0 && spins[small]==1){
//If the most extreme neighbor spin
is positive, that spin is flipped.
     FLIPSPIN(small, spins,....);
               }
               return;
//Once the final recursion ends or no neighbor
has a potential spin flip, the function returns.
        }
        return;
```

This recursive C function flips a qualifying spin and analyzes its neighbors. $H$, the uniform field, is reduced (based on our initial conditions) until a spin flip occurs. Once it does, the above function is called. It causes consecutive spin flips until there are no more qualifying spins. Once the avalanche ends, $H$ is reduced again until another avalanche occurs. The reduction of $H$ ends when all spins flips are negative one.

The major focus of the iterations is to see the distribution of avalanche sizes as a percentage of avalanches that occur. The calculations for percentage and avalanche sizes were summed over numerous iterations and since they were calculated exactly error bars are not relevant to this study. When the distribution of avalanches are plotted using a log scale, a straight line is expected for a relationship that follows the power-law. The presense of the power law will display scale invariance. In other words, SOC will be displayed. Each of our iterations were run with a new gaussian distributed random field. The random field strength was given by choosing two random numbers $x$ and $y$ where

$$s = x^2 + y^2 < 1 \tag{3}$$

and the values of the random numbers fall between -1 and 1. The normal random variable $n$ is produced using the Marsaglia polar method given by the equation

$$n = x\sqrt{\frac{-2\ln s}{s}} \tag{4}$$

.

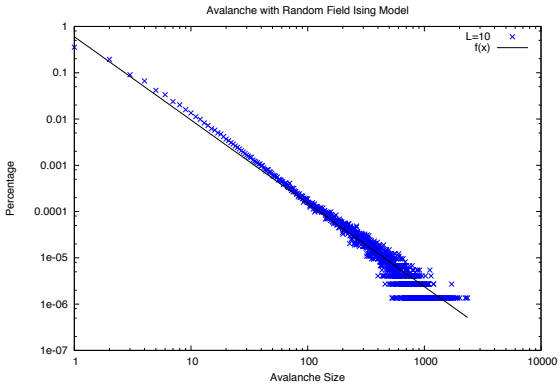



FIG. 2: Here a simulation of the random field ising model with a random field of the standard normal distribution. The dimension spin size was 20 spins or 8000 spins in the system. This curve shows a proof of concept that avalanche distributions of a random-field Ising Model can be clearly displayed, as conceptually conveyed by similar research[3].This was a crucial first step in our study's development.

## III. DATA COLLECTION AND OBSERVATIONS

Through the algorithim described above, multiple runs were made with various values for standard deviation sizes ($R$) of the random field. The random field is generated with a gaussian distribution with a mean value of zero. This section is organized in order of the progress made through the study and notes the major observations of each stage.

Once the code was completed and compiled an initial test was run. A dimension size of 20 spins and $R = 1$, or the standard normal distribution, was used initally in order to test the output of the program. The goal of this output was to test the concept of displaying the distribution of avalanche sizes. Figure 2 displays proof of concept that large numbers of avalanches can be recorded and the distribution of them displayed. It also served as an initial step for further data analysis. As seen from Figure 2 and previous research [6] it can be estimated that a value near $R = 1$ is appropriate.

Next came the need to study clear observables of avalanche distributions for various $R$ values. It was during this process that due to certain limitations the maximum spin dimension size available would be 50 spins, allowing for a maximum of 125,000 spins in the 3D model. Because of this, finite sizing effects made it difficult to get a distinctly measureable critical $R$ ($R_C$). The decision was made to try to observe trends that are expected from previous research[3] [6] in order to test their results. Three major trends can be noted. When $R_C$ is reached a scale invariant distribution of the avalanche sizes will be displayed in a straight line on our log plot. If the $R$ value is too high compared to $R_C$, there tends to be fewer large avalanches causing the plot to become much steeper. In the case where the $R$ value is too low, the plot may appear more linear initially, but there is a noticable increase in the number of smaller avalanches. This causes a noticable 'bump' in the plotted line. Figure 3 captures each of these three trends.

With the observations of the three major trends involved in locating $R_C$, it seem as if the random-field Ising Model has the capability to display a scale invariant avalanche distribution given the right amount of tuning and freedom from finite size effects but it clearly does not display SOC. As further proof that this model does not display SOC, the program was run many more times (from 540 iterations to 6000). Figure 4 shows the increase in precision that comes from increasing the number of iterations. As a final effort to get an extreme trend in avalanche distribution and prove that scale invariance is not displayed a simulation with $R = 0.5$ was ran. The result is a very interesting distribution as seen in Figure 5. The random-field Ising Model is a very good first model in tuning for a scale invariant avalanche distribution but it is far from perfect for displaying SOC. There are obvious limitations in the model itself and this has led to investigating other models in the past few decades.

In order to affirm or deny the conclusion reached in previous research that $R = 2.25$ is the critical we had to extrapolate based on size dependence. Due to code limitations our random-field Ising Model was unable to exceed $L = 50$. Previous research was able to clearly display $R_C$ with the dimension length $L = 320$. We were unable to deny these claims but by our study $R_C$ will most-likely occur at 2.25. We are able to say this

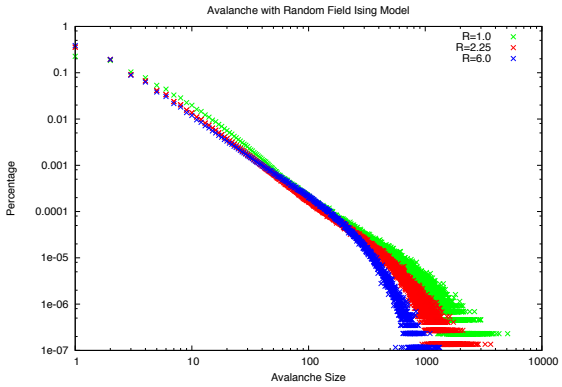



FIG. 3: Displayed are three simulations of the random field Ising model with $R = 1$ (green), $R = 2.25$ (red), and $R = 6.0$ (blue). The dimension spin size was 50, leading to 125000 spins. The simulation was run using 540 iterations in this trial. It is clearly seen that larger system sizes resulted in more distinct distribution lines compared to Figure 1. All three characteristics of an avalanche distribution could be noted here when near the $R_C$ value. First, there is a clear linear trend around $R_C$. With $R = 6.0$, the amount of large avalanche sizes decreases due to the fact that the field given by the neighbors has less effect. Initially, $R = 1.0$ seems to be the most linear, but a clear bump can be seen in the avalanches between avalanche sizes of 3 and 50. The large avalanche size linear appearance is misleading due to the early bump. In fact, a small bump is noticeable in the $R = 6$ line in the 50 to 105 avalanche size range. This figure directly correlates to the results displayed in previous work[6]. $R = 2.25$, the $R_C$ found in previous research, does not seem linear either, but due to finite sizing effects it cannot be ruled out according to this trial.

by extrapolating based off of size dependence. Figure 6 shows the trend of the avalanche distribution following the power law as $L$ increases. If this trend were to continue to infinity it is safe to say that the power law would be displayed.

## IV. SUMMARY AND CONCLUSIONS

Self-organized criticality is a characteristic of many systems found in nature. Such a commonly occurring natural phenomenon has drawn the immediate interest of computational physicists around the globe to study them in order to gain an understanding of nature's methods. Much of the earliest work in this field of research invovled finite-dimensional models such as the random-filed Ising Model. Models similar to this one were tested in order to see if they required special tuning of the standard deviation of the random field's distribution. In our research we have shown that the system fell short of dis-

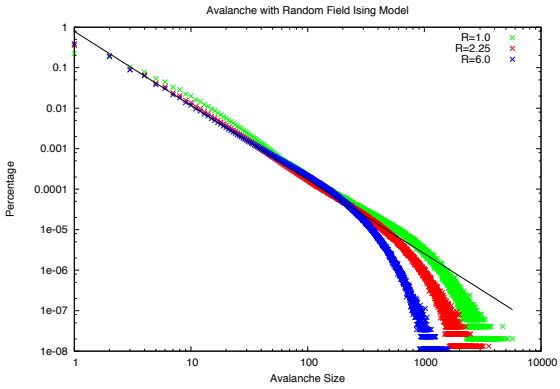



FIG. 4: The results here are from the same program of Figure 2 but with many more iterations (6000 compared to 540). The large increase of iterations led to a more defined distribution and allowed for more intense analysis. The three major features noted previously (see Figure 3 description) are even more defined in this case. A line was inserted into the graph to show the very linear trend of the $R = 2.25$ line, the bumps of the two extreme $R$ values, and the exponential curves of each distribution.

playing scale invariance for every $R$. Instead, $R$ had to be tuned which disqualifies the random-field Ising Model as a system that displays SOC. Using major trends found in early research as the $R$ value is tuned to result in a scale invariant avalanche distribution we have shown that the results of this work are remarkably similar to that of previous early research. Although a linear plot on a log scale graph could not be found due to finite-sizing effects, the major trends expected to be found in the random-field Ising Model were shown to occur which allows us to extrapolate that $R = 2.25$ would very likely display the power law if the dimension size was increased, but limitations in programming prevented us from testing our model in more detail. Finally, extreme values for $R$ displayed avalanche distributions that definitively proved this system required tuning. This work further adds to the legitimacy of early research. The random-field Ising model does not display SOC and we recognize the continued need for a better system to help scientists understand how and why SOC is so prevalent in nature.

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

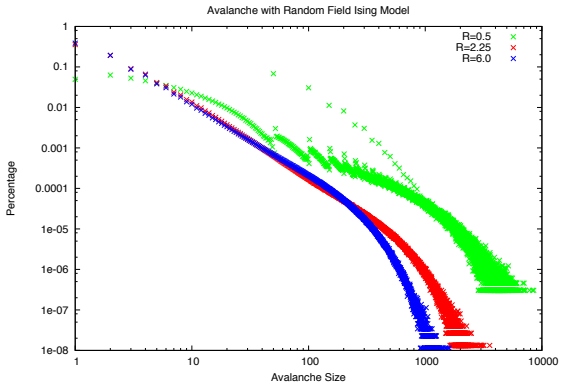



FIG. 5: The results here for $R = 2.25$ and $R = 6.0$ are the same as Figure 4 with the addition of $R = 0.5$ run with the same number of iterations. This figure shows the limitations that occur with the random-field Ising Model and its clear lack of SOC. $R = 0.5$ clearly does not display SOC. This system is not sufficient to display SOC because it requires the precise tuning of a single parameter. Because this model requires precise tuning it is clearly a model that does not display SOC and is in need of much improvement.

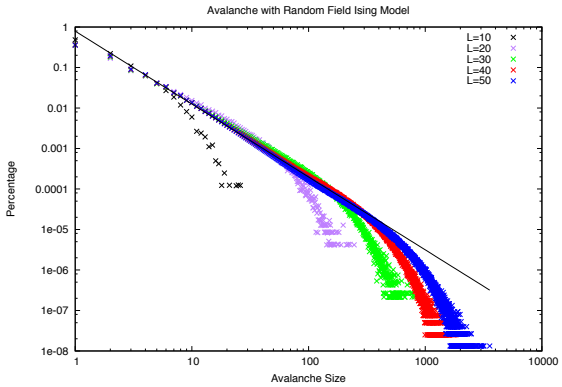



FIG. 6: In this figure the avalanche distributions for $R = 2.25$ is displayed for $L$ (dimension length) of 10, 20, 30, 40, and 50. The trend increasingly follows the power law as $L$ increases. From this we can conclude that as $L$ goes to infinity the power law would very likely be displayed for $R = 2.25$.

[1] K. Schenk, B. Drossel, and F. Schwabl, *Self-Organized Criticality in Forest-Fire Models*, in *Computational Statistical Physics*, edited by K. H. Hoffmann and M. Schreiber (Springer-Verlag, Berlin, 2002), p. 127.

[2] B. B. Mandelbrot, *The Fractal Geometry of Nature* (Freedman, New York, 1983).

[3] P. Bak, C. Tang, and K. Wiesenfeld, *Self-Organized Criticality: An Explanation of* $1/f$ *Noise*, Phys. Rev. Lett. **59**, 381 (1987).

[4] F. S. B. Drossel, *Self-organized critical forest-fire model*,

Phys. Rev. Lett. **69**, 1629 (1992).

[5] J. C. Andresen, Z. Zhu, R. S. Andrist, H. G. Katzgraber, V. Dobrosavljević, and G. T. Zimanyi, *Self-Organized Criticality in Glassy Spin Systems Requires a Diverging Number of Neighbors*, Phys. Rev. Lett. **111**, 097203 (2013).

[6] O. Perkovic, K. A. Dahmen, and J. P. Sethna, *Avalanches, barkhausen noise, and plain old criticality*, Phys. Rev. Lett. **74**, 24 (1995).

# Solitons

Weiguang Huo
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
*School of Science, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China*
(Dated: April 25, 2014)

In this work, I study the Sine-Gordon equation, which gives us several kinds of soliton solutions. I propose an explicit scheme to solve the equation numerically with no-flux boundary conditions. I present the time evolution profile for kink solitons, antikink solitons, kink-kink soliton collision and kink-antikink soliton collision. In addition, I investigate the properties of the kink solitons, demonstrating that their shape do not change over time. The collision between two solitons is elastic and each of them keeps unchanged after collison.

## I. INTRODUCTION

In physics, the behavior of the wave propagation is determined by wave equation, which builds the relation between the partial derivative of time and the partial derivative of space. The wave equation yields the wave dispersion relation, which relates the wavelength of a wave to its frequency. From this relation, the phase velocity and group velocity of a wave packet have the following expressions

$$v_p = \frac{\omega}{k}, \qquad v_g = \frac{\partial \omega}{\partial k},$$

where $v_p$ is the phase velocity, $v_g$ is the group velocity, $\omega$ is the angular frequency and $k$ is the angular wavenumber. If the phase velocity does not equal the group velocity, different frequencies will travel at different speed and the shape of the wave packet will therefore change over time, which indicates that the propagation of the wave packet is dispersive. For example, the Schrodinger equation for a free particle (with $m$ and $\hbar$ set equal to 1),

$$i\frac{\partial u}{\partial t} = -\frac{1}{2}\nabla^2 u,$$

yielding the dispersion relation,

$$\omega = \frac{1}{2}|k|^2.$$

So the group velocity of the wave packet is twice larger than the phase velocity of the wave packet. Figure 1 demonstrates one typical picture of the dispersive wave propagation. The wave packet is localized at the initial time but eventually the wave packet diffuses to an unlimited region of space.

However, in physics, there is also the non-linear Kerr effect [1]: the velocity of the wave packet also depends on its amplitude, not only its frequency. If the pulse has just the right shape, the Kerr effect will exactly cancel the dispersion effect, therefore, the shape of the pulse will not change over time, and this is the definition of a soliton. Solitons have many application in fiber optics [2], biology [3], and magnets [4]. In mathematics, the

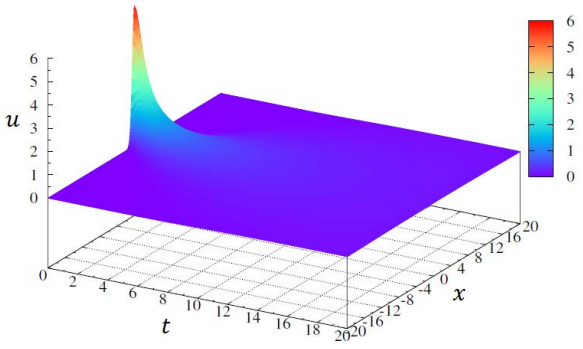



FIG. 1: Time evolution profile of the dispersive wave propagation. The wave packet is localized at the initial time but eventually the wave packet diffuses to an unlimited region of space.

Sine-Gordon equation,

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} + \sin u = 0. \tag{1}$$

can give us several kinds of soliton solutions. My work is to solve the equation numerically and investigate the properties of solitons.

## II. EXPLICIT METHOD

Consider the Sine-Gordon equation (1):

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} + \sin u = 0,$$

on the interval $x \in [a, b]$ with initial conditions,

$$u(x,0) = f(x), \qquad \frac{\partial u}{\partial t}(x,0) = g(x). \tag{2}$$

with no-flux boundary conditions,

$$\left.\frac{\partial u}{\partial x}\right|_{x=a} = 0, \qquad \left.\frac{\partial u}{\partial x}\right|_{x=b} = 0. \tag{3}$$

In the following, I propose a simple explicit scheme to solve equation (1) to (3) [5]. The discretization scheme reads

$$\frac{\partial^2 u}{\partial t^2} = \frac{1}{\Delta t^2}(u_i^{j+1} + u_i^{j-1} - 2u_i^j) + \mathcal{O}(\Delta t^2),$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{\Delta x^2}(u_{i-1}^{j} + u_{i+1}^{j} - 2u_i^j) + \mathcal{O}(\Delta x^2).$$

Plugging in equation (1), one gets,

$$\begin{aligned} u_i^{j+1} =& -u_i^{j-1} + 2(1-\alpha^2)u_i^j \\ &+ \alpha^2(u_{i-1}^j + u_{i+1}^j) - \Delta t^2 \sin(u_i^j), \end{aligned} \tag{4}$$

with $\alpha = \Delta t/\Delta x$, $i = 0, ..., M$ and $j = 0, ..., T$, where $M$ is the amount of space steps and $T$ is the amount of time steps.
To implement the second initial condition, one needs the virtual point $u_i^{-1}$,

$$\frac{\partial u}{\partial t}(x_i, 0) = g(x_i) = \frac{u_i^1 - u_i^{-1}}{2\Delta t} + \mathcal{O}(\Delta t^2).$$

Hence, one can rewrite the previous equation as,

$$u_i^{-1} = u_i^1 - 2\Delta t g(x_i) + \mathcal{O}(\Delta t^2)$$

and the second time row $u_i^1$ can be calculated as

$$\begin{aligned} u_i^1 =& \Delta t g(x_i) + (1-\alpha^2) f(x_i) \\ &+ \frac{1}{2}\alpha^2(f(x_{i-1}) + f(x_{i+1})) - \frac{\Delta t^2}{2}\sin(f(x_i)). \end{aligned} \tag{5}$$

In addition, the following two expressions can be deduced by no-flux boundary conditions for two virtual space points $u_{-1}^j$ and $u_{M+1}^j$

$$\left.\frac{\partial u}{\partial x}\right|_{x=a} = 0 \Leftrightarrow \frac{u_1^j - u_{-1}^j}{2\Delta x} = 0 \Leftrightarrow u_{-1}^j = u_1^j,$$
$$\left.\frac{\partial u}{\partial x}\right|_{x=b} = 0 \Leftrightarrow \frac{u_{M+1}^j - u_{M-1}^j}{2\Delta x} = 0 \Leftrightarrow u_{M+1}^j = u_{M-1}^j.$$

In conclusion, one can rewrite the differential scheme to a more general matrix form. In matrix notation the second time row is given by

$$\mathbf{u}^1 = \Delta t \gamma_1 + A\mathbf{u}^0 - \frac{\Delta t^2}{2}\beta_1, \tag{6}$$

where

$$\gamma_1 = (g(a), g(x_1), g(x_2), ..., g(x_{M-1}), g(b))^T$$

and

$$\beta_1 = (\sin(u_0^0), \sin(u_1^0), ..., \sin(u_{M-1}^0), \sin(u_M^0))^T$$

are $M+1$ dimensional vetors and $A$ is a tridiagonal square $(M+1)\times(M+1)$ matrix of the form

$$A = \begin{pmatrix} 1-\alpha^2 & \alpha^2 & 0 & \cdots 0 \\ \alpha^2/2 & 1-\alpha^2 & \alpha^2/2 & \cdots 0 \\ 0 & \alpha^2/2 & 1-\alpha^2 & \cdots 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \alpha^2 & 1-\alpha^2 \end{pmatrix}$$

Other time rows, which can be derived by similar way, can also be written in the matrix form as

$$\begin{aligned} &\mathbf{u}^{j+1} = -\mathbf{u}^{j-1} + B\mathbf{u}^j - \Delta t^2 \beta, \\ &\text{with} \qquad j = 1, 2, \cdots, T-1 \end{aligned} \tag{7}$$

where

$$\beta = (\sin(u_0^j), \sin(u_1^j), ..., \sin(u_{M-1}^j), \sin(u_M^j))^T$$

is a $M+1$ dimensional vector and $B$ is a square matrix, defined by $B = 2A$.

Now let us apply the explicit scheme described above to equation (1) to (3) and solve it on the interval $[-L, L]$ using the following parameters in Table 1.

TABLE I: Parameters used in the calculation

| | | |
|---|---|---|
| Amount of space steps | $M$ | 400 |
| Space interval | $L$ | 20 |
| Space discretization step | $\Delta l$ | 0.1 |
| Amount of time steps | $T$ | 1800 |
| Final time | $t_f$ | 90 |
| Time discretization step | $\Delta t$ | 0.05 |
| Velocity of the soliton | $c$ | 0.2 |

## III. RESULTS

### A. Kink Soliton

#### 1. *Profile*

The initial conditions for kink soliton solution are the following:

$$f(x) = 4\arctan\left(\exp\left(\frac{x + L/2}{\sqrt{1-c^2}}\right)\right),$$
$$g(x) = -2\frac{c}{\sqrt{1-c^2}}\operatorname{sech}\left(\frac{x + L/2}{\sqrt{1-c^2}}\right).$$

The basic profile of the kink soliton solution is shown in Figure 2(a). The width of the kink soliton solution, $\sigma$ is defined by following:

$$\sigma = x_2 - x_1,$$

where $x_1$ is the position when $u$ equals the minimum of u, $u_{min} + 0.5$, and $x_2$ is the position when $u$ equals the

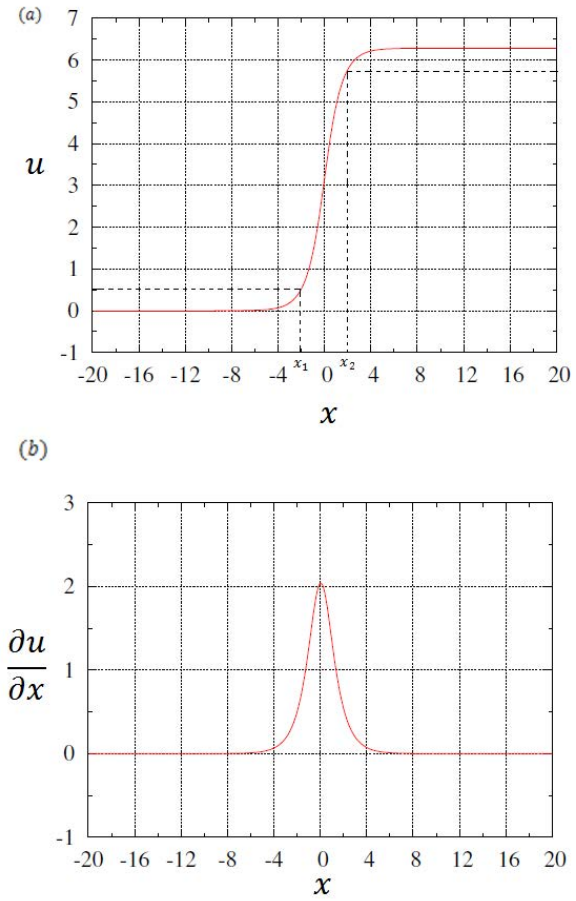



FIG. 2: (a) The profile of the kink soliton, and $x_2 - x_1$ is the width of the kink soliton. (b) The profile of the hump-shaped soliton, obtained by the partial space derivative of the kink soliton.

maximum of u, $u_{max} - 0.5$. By using the linear approximation, the definiton of the average slope of the kink soliton, $k$ is straightforward:

$$k = \frac{u(x_2) - u(x_1)}{\sigma}.$$

As shown in the figure, the shape of the kink soliton is similar to hyperbolic tangent function, which does not look like a wave packet. The solitary wave solution, named hump-shaped soliton, can be obtained by $\frac{\partial u}{\partial x}$, whose shape is shown in Figure 2(b)

Next, I show the time evolution profile for both the kink soliton in Figure 3(a) and the hump-shaped soliton in Figure 3(b). The propagation of the kink soliton and the hump-shaped soliton is stable and their shapes do not change over time.

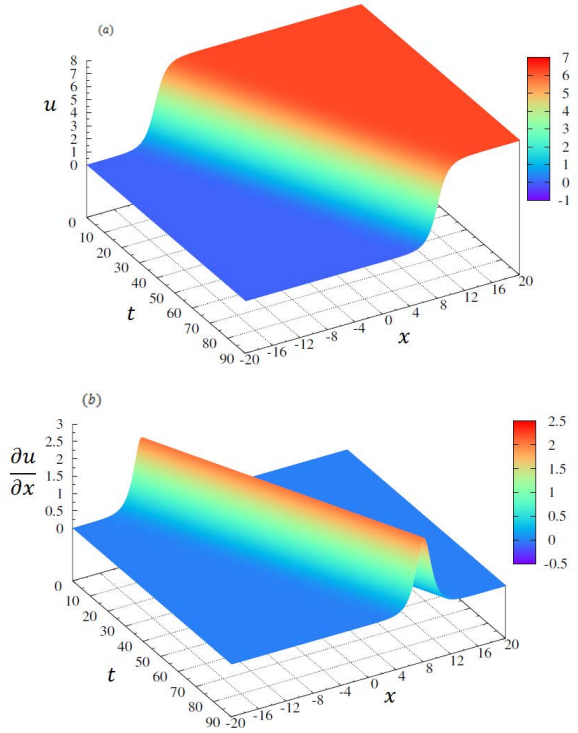



FIG. 3: (a) The time evolution profile for the kink soliton solution and (b) the hump-shaped soliton.

### 2. *Properties*

In this section, I will investigate the properties of the kink soliton solution, including the width, the average slope and the calculation error compared with the exact solution to demonstrate that the solution of Sine-Gordon equation does not change over time with acceptable error, which indicates that they are solitons.

First, I investigate how the width of a soliton evolves with time. I calculate the width of the kink soliton at arbitrary time and plot in Figure 4(a). The figure shows that the width keeps constant over time with acceptable calculation error. Then, I show the relation between the average slope of the wave packet and time in Figure 4(b). Expectedly, the average slope also keeps constant over time with acceptable calculation error.

Finally, I study the relation between the calculation error at final time $t_f$ and the space discretization step, $\Delta l$. The analytical solution of the kink soliton is given by following[6]:

$$u(x,t) = 4\arctan\left(\exp\left(\frac{x + L/2 - ct}{\sqrt{1-c^2}}\right)\right). \quad (8)$$

So the average error, $E$ at final time $t_f$ can be obtained

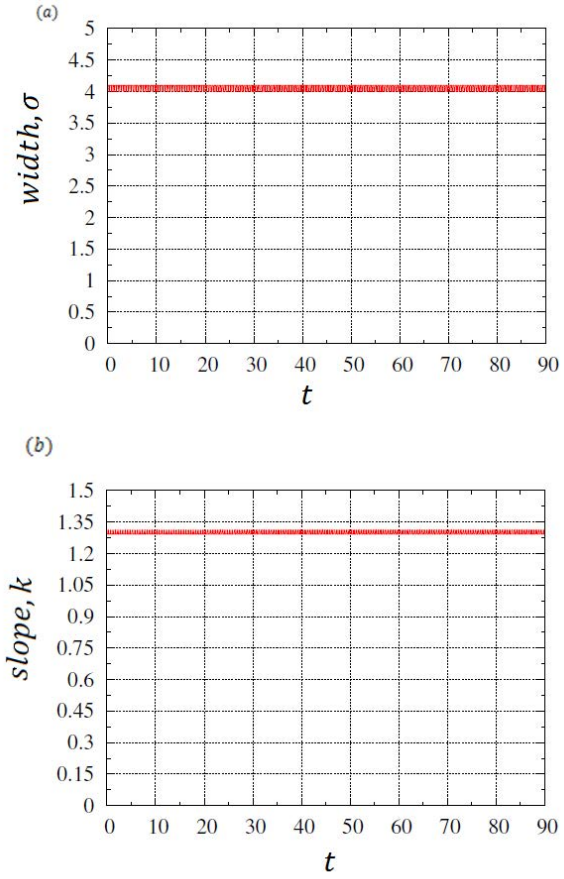



FIG. 4: (a) The width of the kink soliton versus time. (b) The average slope of the kink soliton versus time. The width and the slope keep constant over time with acceptable calculation error.

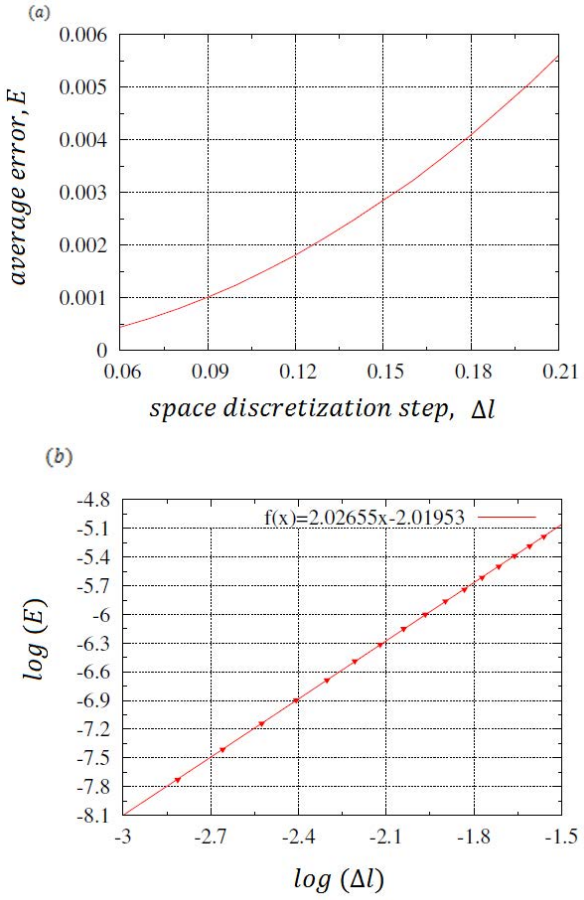



FIG. 5: (a) Error at final time versus the space discretization step and (b) in log scale. It is shown that the error increases quadratically with the space discretization step, and the error is small enough, thus, acceptable.

by

$$E = \frac{\sum_{i=0}^{M} \left| u(i, t_f) - u_i^T \right|}{M}. \tag{9}$$

Figure 5(a) demonstrates that the error increases nonlinearly with the space discretization step. In order to make it clearer, I plot in log scale and fit these data points with linear function. As shown in Figure 5(b), these points fit the function $f(x) = 2.02655x - 2.01953$ extremely well, which indicates that the error increases quadratically with the space discretization step and this result is consistent with the expicit method. In additon, I notice that the error is small enough, thus, acceptable.

### B. Antikink Soliton

The initial condition for antikink soliton solution are the following:

$$f(x) = 4 \arctan\left(\exp\left(-\frac{x - L/2}{\sqrt{1-c^2}}\right)\right),$$
$$g(x) = -2\frac{c}{\sqrt{1-c^2}} \operatorname{sech}\left(\frac{x - L/2}{\sqrt{1-c^2}}\right).$$

Then, I show the time evolution profile for both the antikink soliton in Figure 6(a) and the hump-shaped antisoliton in Figure 6(b). The propagation profile of the antikink soliton is similar to the kink soliton, but the propagation direction is opposite.

In conclusion, the kink soliton and antikink soliton solutions and their corresponding solitary wave solutions are the basic solutions of the Sine-Gordon equation. Their shapes do not change over time, which is a special

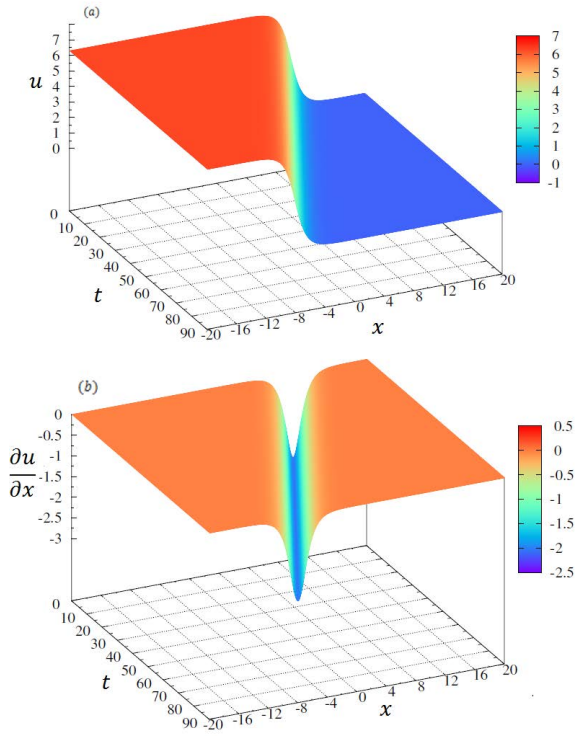



FIG. 6: (a) The time evolution profile for the antikink soliton solution and (b) the hump-shaped antisoliton.

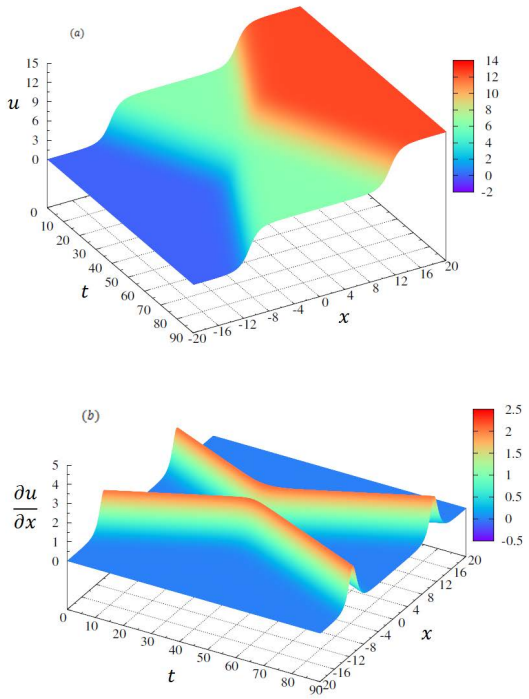



FIG. 7: (a) The time evolution profile for the kink-kink soliton collision solution and (b) the corresponding solitary wave solution. The shape of each soliton stays unchanged.

characteristic compared to other dispersive wave equation.

### C. Kink-Kink Soliton Collision

In this section, I will investigate the collision between two solitons. The first case is kink-kink soliton collision. The initial conditions for the kink-kink soliton collison solution are the following:

$$f(x) = 4\arctan\left(\exp\left(\frac{x+L/2}{\sqrt{1-c^2}}\right)\right) + 4\arctan\left(\exp\left(\frac{x-L/2}{\sqrt{1-c^2}}\right)\right)$$
$$g(x) = -2\frac{c}{\sqrt{1-c^2}}\mathrm{sech}\left(\frac{x+L/2}{\sqrt{1-c^2}}\right) + 2\frac{c}{\sqrt{1-c^2}}\mathrm{sech}\left(\frac{x-L/2}{\sqrt{1-c^2}}\right).$$

Numerical kink-kink soliton collision solution and corresponding solitary wave solution is presented in Figure 7 (a) and (b). As shown in the figure, kink solitons can interact with other solitons, and the shape of each soliton stays unchanged, which indicates that the collision is elastic. This is another characteristic of solitons.

### D. Kink-Antikink Soliton Collision

The initial conditions for the kink-antikink soliton collision solution are the following:

$$f(x) = 4\arctan\left(\exp\left(\frac{x+L/2}{\sqrt{1-c^2}}\right)\right) + 4\arctan\left(-\exp\left(\frac{x-L/2}{\sqrt{1-c^2}}\right)\right)$$
$$g(x) = -2\frac{c}{\sqrt{1-c^2}}\mathrm{sech}\left(\frac{x+L/2}{\sqrt{1-c^2}}\right) - 2\frac{c}{\sqrt{1-c^2}}\mathrm{sech}\left(\frac{x-L/2}{\sqrt{1-c^2}}\right).$$

Numerical kink-antikink soliton collision solution and corresponding solitary wave solution is presented in Figure 8 (a) and (b). As shown in the figure, the shape of each soliton stays unchanged.

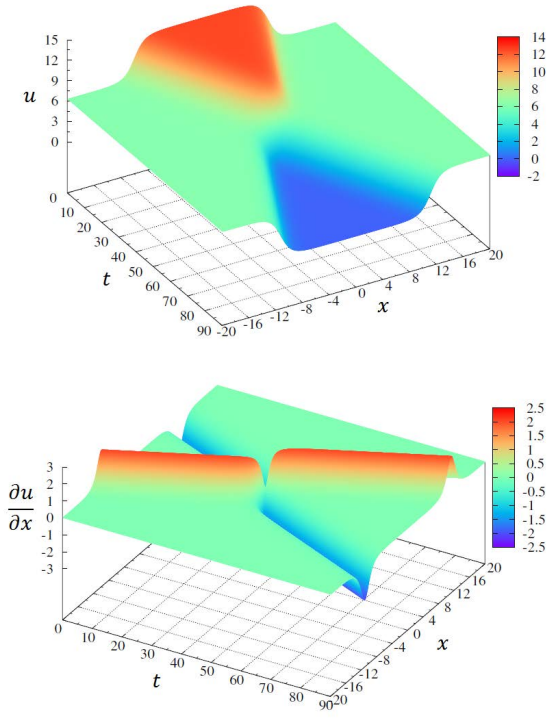



FIG. 8: (a) The time evolution profile for the kink-antikink soliton collision solution and (b) the corresponding solitary wave solution. The shape of each soliton stays unchanged.

## IV. SUMMARY AND CONCLUSIONS

In summary, I proposed an explicit scheme to solve the Sine-Gordon equation numerically with no-flux boundary conditions, which gave us several kinds of soliton solutions. I presented the time evolution profile for kink solitons, antikink solitons, kink-kink soliton collision and kink-antikink soliton collision. In addition, I investigated the properties of the kink solitons, demonstrating that their shape did not change over time with acceptable calculation error. The collision between two solitons was elastic and each of them stayed unchanged after collison.

### Acknowledgments

I acknowledge with gratitude to associate professor Helmut Katzgraber and my teaching assistant Ross McDonald, who have always been sincere and helpful for my term paper.

It is a great oppotunity for me to write a paper about "Solitons". At the time of preparing this term paper, I am gone through different books and websites which help me to get acquainted with this topics and imporve my study ability a lot. There may be shortcoming, grammar mistakes, and factual errors in this paper but I will try to make it better in future.

[1] P. Weinberger, *John Kerr and his Effects Found in 1877 and 1878*, Phil. Mag. Lett. **82**, 897 (2008).

[2] S. T. Cundiff, *Observation of Polarization-Locked Vector Solitons in an Optical Fiber*, Phys. Rev. Lett. **82**, 3988 (1999).

[3] Z. Sinkala, *Soliton/exciton transport in proteins*, Proc. Natl. Acad. Sci. **102**, 9790 (2005).

[4] Z. V. Kosevich, A. M., Gann, V. V., *Magnetic soliton motion in a nonuniform magnetic field*, JEPT **87**, 401 (1998).

[5] URL `http://pauli.uni-muenster.de/tp/fileadmin/lehre/NumMethoden/WS0910/ScriptPDE/SineGordon.pdf`.

[6] C. Inna Shingareva, *Solving Nonlinear Partial Differential Equations with Maple and Mathematica* (Springer, New York, 2011).

# Comparing the Oribts of Three Body Systems Using Dual Algorithms

Dimitrios Michaelides
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 26, 2014)

Here I have studied two different algorithms for integrating differential equations and applied them to the chaotic system of three bodies free to orbit each other. These orbits were confined to a two dimensional plane and given various initial positions and velocities. Not only that, but collision parameters were added to show added physical effects and better simulate what could be described as a developing planetary system. As the system is intrinsically chaotic, it was very sensitive to initial conditions. Due to these reasons, a symplectic algorithm was run several times under varying initial parameters for each body (mass, position, and velocity) to locate potentially stable orbits. Due to the volume of data produced, a random sampling method was used to locate stable orbits and then those parameters were run on the other algorithm. But first the algorithms need to be constructed.

## I. INTRODUCTION

Multi body orbitals are an integral part of modern astronomy. It is through almost pure gravitational interactions that celestial systems form. These systems can range from a multi star orbital system, to a standard solar system to a series of lighter bodies interacting with each other. Through observing these systems, one can make predictions about what climates may be like on certain planets and what the masses of the bodies may be, based on periodicity. This of course is all done by calculating the force using Newton's famous law of gravitation:

$$\vec{F} = G\left(\frac{m_1 m_2}{r^2}\right)\hat{r} \tag{1}$$

Where $F$ is the force between the two masses, $G$ is the gravitational constant, each $m$ denotes a mass of either body, and $r$ is the distance between the two masses. With all the above information, scientists can even determine if a planet will be habitable for humans should they visit it in the future. By judging a planet's average distance from the sun as well as its mass, they can see if the planet's climate and gravity would be suitable for terrestrial beings. [1]Not only that, but gravitational perturbations in orbits have even been used to locate new planets in the past. Neptune, for example, was discovered due to abnormalities in Uranus' predicted orbit. Because Uranus deviated from what experts were expecting, they decided that another body must be perturbing its orbit. Hence, Neptune was discovered. In more recent astronomy, orbital deviations spotted in distant stars point astronomers to information about the star's surroundings that are not readily visible to the telescope. These surrounding bodies could be anything such as exoplanets, other stars, or even black holes. By coming up with an updated model, astronomers are able to figure out exactly what it might be and where they can find it. It was through methods such as these that dark matter was first theorized to exist when bodies had perturbers that could not be seen and could not have been black holes.

## II. MODEL

For the majority of multi body problem with more than two bodies, there is will be no analytical solution. [**?** ]This was proved by Heinrich Bruns and Henri Poincare for three body problems in the 1680's and was expanded to encompass systems with higher numbers of bodies. However, over the years, beginning with Lagrange and Euler, analytical solutions have been discovered for three body systems. In 2013, two physicists Milovan uvakov and Veljko Dmitrainovi at the institute of Belgrade added 13 types of analytical three body solutions, thus bringing the grand total up to 16. This was a huge leap in discoveries. Each type of system is classified into a family based on the characteristic shape and period of its orbital. However, these families are exceptions to the rule. The vast majority of 3 body systems are unstable and have no analytical solution. Because of this, numerical integrators are of utmost important when studying such systems. One very important method of integration is the Runge-Kutta Method. This is likely one of the most famous integration methods available, especially the Runge-Kutta 4 method, often referred to as RK4. The basic formula for this method is shown below:

$$k_1 = hf(x_n, y_n) \tag{2}$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \tag{3}$$

$$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \tag{4}$$

$$k_2 = hf(x_n + h, y_n + k_3) \tag{5}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{6}$$

Where $h$ is a time step, and $f$ essentially the differential equation that you are trying to solve. [2]This is an

explicit method that uses a trial step at a midpoint to cancel out lower order error terms. As its name suggests, the RK4 eliminates error to the fourth order. While this is a generally accurate and popular method to go to, the method has some flaws. Namely, the energy of the system is not conserved. Generally, the energy changes as time goes on which can lead to errors in calculations along the way. Also, it should be noted that the method itself is not time reversible, so making calculations backwards in time can be a little tricky. However, since the early 1900's when the RK4 method was first introduced, new algorithms have come to take their place. A good example is the Forest-Ruth algorithm. The Forest-Ruth algorithm has the steps which follow:

$$x = x + \theta\frac{h}{2}v \tag{7}$$

$$v = v + \theta h F(x) \tag{8}$$

$$x = x + (1-\theta)\frac{h}{2}v \tag{9}$$

$$v = v + (1-2\theta)hF(x) \tag{10}$$

$$x = x + (1-\theta)\frac{h}{2}v \tag{11}$$

$$v = v + \theta h F(x) \tag{12}$$

$$x = x + \theta\frac{h}{2}v \tag{13}$$

Where $h$ is a given time step, $F(x)$ is the force as a function of position, $\theta$ is defined as:

$$\theta = \frac{1}{2 - 2^{\frac{1}{3}}} \tag{14}$$

And $v$ is defined as:

$$v = \frac{dx}{dt} \tag{15}$$

Like the RK4 it is a fourth order integrator, but it has a few advantages. For one thing, the Forest-Ruth algorithm is a symplectic integrator. This is an integration method that uses canonical transformations from Hamilton's Equations. This method better conserves the energy of the system over long periods of time as the method was made to conserve a slightly perturbed Hamiltonian of the system. [3]Also, because of the method's symmetry, the system is time reversible, thus allowing for time reversal invariance. Along with the two numerical integrator, I personally developed a collision model. This was coded with the following lines:

```
if (sqrt(pow(x1-x2,2.0) + pow(y1-y2,2.0)) < .01)
{
col12 = 1;
if (m1 != 0)
{
px2 = ((px1*m1) + (px2*m2))/(m1+m2);
py2 = ((py1*m1) + (py2*m2))/(m1+m2);
m2 = m1 + m2;
}
}
```

Followed by the actual integration method and later by:

```
if (col12 == 1)
{
x1 = x2;
y1 = y2;
px1 = 0;
py1 = 0;
m1 = 0;
}
```

Essentially, if the bodies got too close to each other (within a planetary radius), then they would collide and stick, thus trigger the collision parameter (col12) to switch from 0 to 1, which would be a trigger to skip calculations done on planet one (for efficiency). When done, the program would use momentum conservation to find the new velocity (px,py) of the composite body as well as add the two masses. After each iteration of the method, it would then write the coordinates of planet 2 onto planet one so that data could still be recorded. If for some reason, all three planets were to collide, the program would set all coordinates to the origin and end the simulation as there would be nothing of interest left. This model served a dual purpose. On one hand, it added a touch of realism. Granted it is simplified realism, but realism none the less. From a more practical standpoint, the collision parameters kept the bodies from getting too close to each other. Before this was implemented, there would be problems with separate bodies getting so close that the gravitational force would diverge and send the bodies spontaneously hurling into space. This model fixed that problem by making two bodies that got close enough into one single body and killing any two body forces between the old bodies thereafter.

## III. METHODS

When making the calculations, I had several obsticles to overcome. I began by looking at preceding studies into three body systems. I found certain sources that gave initial conditions which claimed would provide stability. However, as I modeled each of these, I saw that while they may have seemed stable at first (if at all) the stability did not last forever as each system broke apart. So, I was left with little clue regarding where to start when looking for a stable orbit. Hence, I decided to use a rather crude

brute force method. I originally decided that I would loop over each initial parameter (mass, two dimensional velocity, and two dimensional position) and scan the results for interesting trajectories and tweak them. At first, I was looking to have 2000 variations on each parameter. However, I soon realized that this would lead to $2000^{15}$ data sets. Not only would that be an absurd amount of data to go through, but also at least two thirds of them would be redundant. So, I decreased each parameter to have only about 20 variations as well as made unique position ranges for each body to ensure that there were no redundant data sets. To further decrease the number of data sets, I simply fixed the mass and initial position of a body. In the end, I was left with about $10^9$ data sets, a large number for sure, but incredibly smaller than what it I had originally intended. After running the Forest-Ruth algorithm over a long period of time, I had to find a way to analyze the data. As it would be nearly impossible to go through each and every set, I was inspired by the Monte Carlo method. I decided to use a random number generator to pull a number n and look at the $n^{th}$ data set produced. From the couple of hundred sets of data I looked at, most seemed to be unstable or to simply fly off with little gravitational interactions with their neighbors. However, some initial conditions yielded interesting results. I took the most interesting data sets and ran them again with a smaller time step, for better resolution, as well as a longer total time, for better checks on stability. After doing this, I ran the initial conditions again in the RK4 algorithm so that I could compare the two algorithms to see how well they matched up.

## IV. RESULTS

Below, I have included several graphs displaying the trajectories of different initial conditions. Please note that the first and second graphs are of the same initial conditions. The first is a model from the Forest-Ruth algorithm while the second was modeled with the RK4 method. As one might be able to tell, the two methods seemed to match each other fairly well. As such, the remaining graphs will only be products of the Forest-Ruth algorithm. In the captions, the initial conditions are listed in the following order: $x_1, y_2, x_2, y_2, x_3, y_3, px_1, py_1, px_2, py_2, px_3, py_3, m_1, m_2, m_3$ where the subscripts denote which body the parameter refers to. Also note that body one is modeled in red, body two in green, and body three in blue.

Bodies two and three interact with one briefly before separating and obtaining an interesting oscillatory pattern. It almost appears as if body two is leading body three in a sinusoidal orbit.

As one can see, this is identical to the Forest-Ruth result.

These initial conditions came from a given homework problem early in the course. It was supposed to be a stable three body system, however, one can clearly see

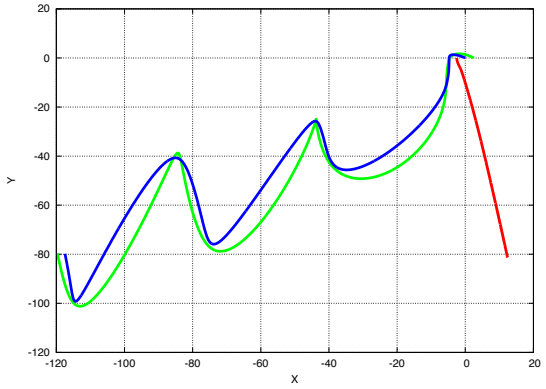



FIG. 1: Forest-Ruth modeled with initial parameters: -2.5, 0, 2.5, 0, 0, 0, 0, -1.25, -1, 0.75, -1.25, 0.75, 1, 1, 1

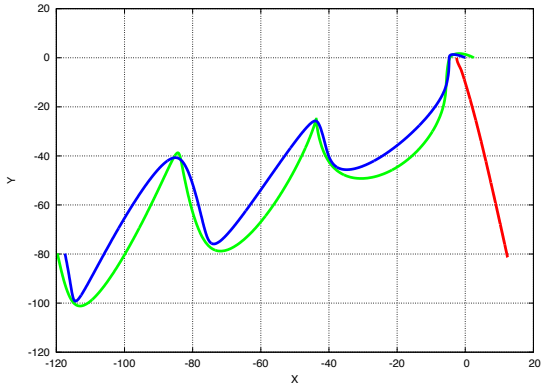



FIG. 2: RK4 modeled with initial parameters: -2.5, 0, 2.5, 0, 0, 0, 0, -1.25, -1, 0.75, -1.25, 0.75, 1, 1, 1

that the orbits fall apart after a certain period of time.

This is the Butterfly 1 pattern as described by Milovan uvakov and Veljko Dmitrainovi. For reasons I do not know, the orbit was not stable as all as the three bodies end up colliding and did not behave a bit as the publication said it would. However, this could be due to my collision simulation which may have stopped the orbits from reaching stability due to becoming too close and colliding.

Shortly after the modeling begins, bodies one and two collide and enter an orbit with body three. This orbit ends up being unstable, however, as body three eventually collides with the composite body and the simulation ends.

Shortly after the modeling begins, bodies one and two collide and enter an orbit with body three. This time,

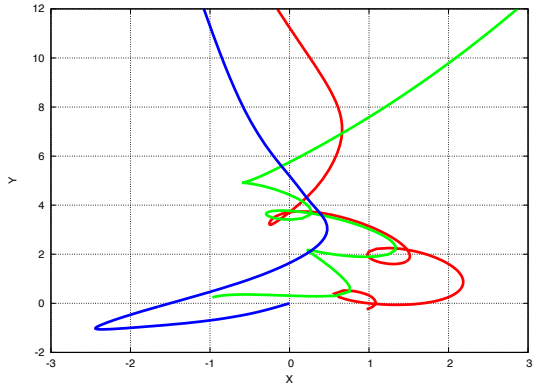



FIG. 3: Forest-Ruth modeled with initial parameters: 0.970004, -0.243088, -0.970004, 0.243088, 0, 0, 0.466204, 0.432366, 0.466204, 0.432366, -0.932407, -0.864731 1, 1, 1

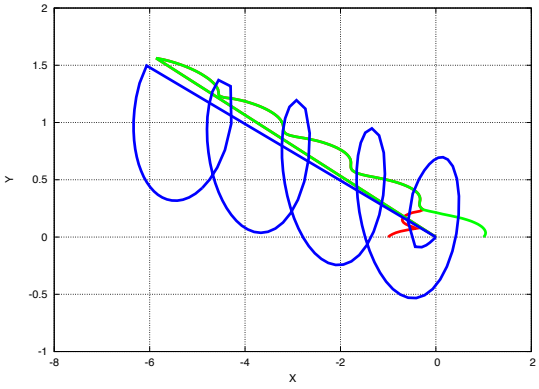



FIG. 4: Forest-Ruth modeled with initial parameters: -1, 0, 1, 0, 0, 0, 0.306893, 0.125507, 0.306893, 0.125507, -0.613786, -0.251014, 1, 1, 1

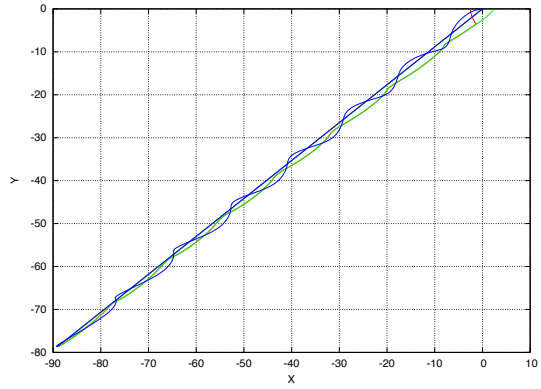



FIG. 5: Forest-Ruth modeled with initial parameters: -2.5, 0, 2.5, 0, 0, 0, 0, -1.25, -1, -1.25, -1.25, -0.25, 1, 1, 1

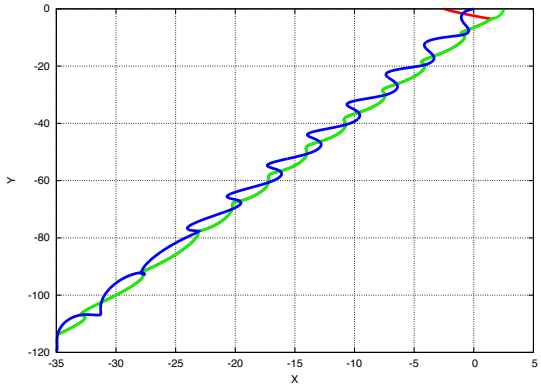



FIG. 6: Forest-Ruth modeled with initial parameters: -2.5, 0, 2.5, 0, 0, 0, 0, -1.25, -1, 0.75, -1.25, 0.75, 1, 1, 1

the orbit appears to be stable, albeit chaotic.

No collisions in this one. However, bodies two and three interact with wild chaotic behavior before flying off into space while body one has essentially no interactions.

## V. SUMMARY AND CONCLUSIONS

Regrettably, I was unable to find any truly stable orbits with my methods. Even initial conditions which were given as being stable seemed to diverge at the end. At least, that is true from what I know. I only looked at a random sample of about a hundred models. That is less than a thousadnth of a percent of the total data sets that were calculated. It is very possible that I had a perfectly stable three body system, but did not see it. But, upon looking at the literature of what have been found to be stable initial conditions, It became obvious that by my methods, I would have needed much more than $2000^{15}$ data sets to find them for sure. This is because the initial conditions of the published results have several decimal digits of accuracy whereas my method only went in intervals of $\frac{1}{4}$. However, I was able to see some truly interesting orbital behavior. Not only that, but I was able to model collisions, which gave a much more accurate picture as to what might occur in the formation of a celestial system. And with the collision system, I was able to obtain somewhat stable orbitals with a collided and uncollided body. My methods in programing were rather straight forward and could be generalized from

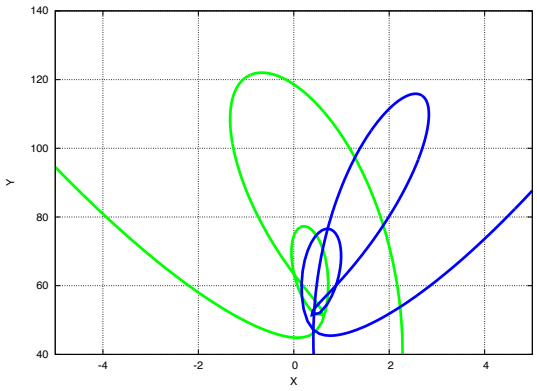



FIG. 7: Forest-Ruth modeled with initial parameters: -2.5, 0, 2.5, 0, 0, 0, 1, -1.25, 0, -0.25, 0.75, -1.25, 1, 1, 1

two to three dimensions fairly easily for future modeling. In the end, I gained a deeper understanding of numerical integrators and was able to directly compare them and the results they would give.

**Acknowledgments**

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] W. H. Smyth, *Account of some circumstances historically connected with the discovery of the planet exterior to uranus*, Royal Astronomical Society **VII**, 23 (1846).

[2] (2014), URL `http://tinyurl.com/y27h2r`.

[3] P. Young, *The leapfrog method and other "symplectic" algorithms for integrating newton's laws of motion* (2013).

# Finite Temperature Coherence in the Two-Dimensional Edwards-Anderson Model

Andrew Ochoa
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 26, 2014)

The most coherent state of a two-dimensional ferromagnet with disorder is not at the ground state but at a finite temperature known as the Nishimori Temperature. Nishmori has proven the result analytically. [1] This work seeks to prove the result numerically.

## I. INTRODUCTION

The Ising model is a simple tool to study a multitude of problems in statistical physics, and with some creativity, problems in biology, sociology, and computer science, to name a few. One in particular is the minimization of errors in the transmission of information through noisy media. When transmitting information, one encodes information to be fed into a medium. In a noisy medium, errors are introduced that obscure the real information. The receiver then decodes the noisy signal and possible errors are detected and corrected.

The Ising spin glass is a collection of anisotropic spins $S_i = \pm 1$, arranged in a lattice (square, hexagonal, etc.) with nearest neighbor interactions $J_{ij} = \pm 1$ (Figure 1). The Hamiltonian, or the total energy of the system, is given by

$$\mathcal{H} = \sum_{\langle i,j \rangle} J_{ij} S_i \, S_j. \tag{1}$$

Suppose the information to be sent is a configuration of Ising spins $\{\epsilon_i = \pm 1\}_{i=1,\ldots,N}$. A simple way is to send the interactions $J_{ij} = \epsilon_i \epsilon_j$. The receiver recieves the noisy interactions $\tilde{J}_{ij} = -J_{ij}$ and then finds the ground state of the disordered Ising spin glass Hamiltonian. It is important to note that even if a small percentage $q$ of the interactions are incorrect, it is still possible to retrieve the true configuration as long as errors are isolated from each other because isolated frustration does not change the ground-state configuration.

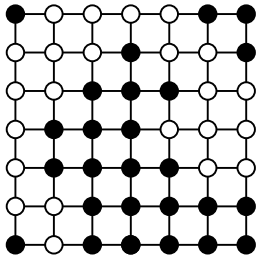


FIG. 1: Illustration of the two-dimensional Ising model with nearest neighbor interactions. Filled [open] circles represent $S_i = +1$ [$S_i = -1$]. The spins only interact with their nearest neighbors (lines connecting the circles). *Printed with permission from the author.* [2]

Nishimori proved analytically the proposed rule to assign the decoded signal $S_i(T_N) = \frac{\langle \sigma_i \rangle_{T_N}}{|\langle \sigma_i \rangle_{T_N}|}$.[3] This observable is studied in detail to reproduce the analytic conclusions of Nishimori.

## II. MODEL, OBSERVABLES, ALGORITHM

The Hamiltonian of the Edwards-Anderson Ising spin glass is

$$\mathcal{H}(\sigma_i) = -\sum_{ij}^{N} J_{ij} \sigma_i \, \sigma_j, \tag{2}$$

with Ising spins $\sigma_i \in \{\pm 1\}$ on the vertices of a square lattice. We impose periodic boundary conditions that deform the square lattice into a toriodal shape. The toroidal square lattice has $L \times L = N$ sites. The interactions are chosen from a bimodal ($J_{ij} = \pm 1$) distribution given by

$$\mathcal{P}(J_{ij}) = p\delta\left(J_{ij} - J\right) + (1-p)\delta\left(J_{ij} + J\right). \tag{3}$$

$p$ is the probability of having a ferromagnetic bond, $q = 1 - p$ the probability of having an antiferromagnetic bond. The pure Ising model is recovered for $p \to 1$. Figure 2 shows the phase diagram for the two-dimensional $\pm J$ Ising spin glass. The exact expectation value of the energy can be calculated at the Nishimori Temperature $N_T$ for values of p via

$$[\langle E \rangle_T]_p = \frac{-zN(2p-1)}{2}, \tag{4}$$

where $N$ is the number of sites and $z$ is the number of neighbors. The Nishimori Temperature can also be calculated[4] from the percentage of antiferromagnetic bonds $q$ by

$$T_N = \left(\frac{1}{2} \ln \frac{q}{1-q}\right)^{-1}. \tag{5}$$

The observable $S_i(T_N) = \frac{\langle \sigma_i \rangle_{T_N}}{|\langle \sigma_i \rangle_{T_N}|}$ has the following motivation. The $J_{ij} = \epsilon_i \epsilon_j$ interactions are transmitted

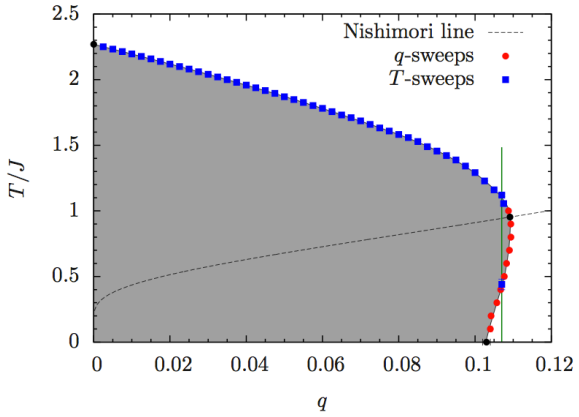



FIG. 2: Phase diagram of the two-dimensional Ising spin glass with $\pm J$ interactions. The shaded region is ferromagnetic, while the white region is paramagnetic. *Printed with permission from the author.* [5]

in a noisy channel. Due to the noise it is possible for errors to be introduced in the form of flipped interactions $\bar{J}_{ij} = -J_{ij}$ with a probability $q = 1-p$. The receiver then calculates the ground state of the new noisey Hamiltonian $\bar{H} = -\sum_{ij}^{N} \bar{J}_{ij}\sigma_i\,\sigma_j$. Let $S_i = \pm 1$ be the retreived configuration. It is then possible to calculate an error rate via

$$m = \frac{1}{N}\sum_{i}^{N} \epsilon_i\, S_i, \tag{6}$$

$$p_{error} = \frac{1-m}{2}, \tag{7}$$

where m is the Mattis magnetization.[6] If the original configuration is correctly retrieved, in other words, $\epsilon_i = S_i$ then $p_{error} = 0$. The goal is to minimize $p_{error}$ or maximize m. From maximum-likelihood decoding, a function is chosen for for $S_i$ so that $\bar{H}$ can be retrieved.

$$S_i(T_N) = \frac{\langle\sigma_i\rangle_{T_N}}{|\langle\sigma_i\rangle_{T_N}|} \tag{8}$$

Angular brackets denote a thermal average and square brackets denote an average over disorder. (8) is then averaged over disorder to reach the final inequality, the analytical proof of which is refered to Nishimori.[1]

$$\bar{m}(T,p) \equiv [\epsilon_i S_i(T)]_p \leq \bar{m}(T_N,p) \tag{9}$$

It is important to note that $\bar{m}(T,p)$ is not the thermodynamic magnetization $[\langle\sigma_i\rangle_T]_p$. In other words, the overall spin allignment is largest at the Nishimori Temperature. The spin configuration at $T_N$ may be more coherent than at $T \neq T_N$.

The calculations of the Hamiltonian $\bar{H}$ are done using the Monte Carlo method. The Monte Carlo method uses repeated random sampling of the observable (8). The choice of simulation parameters given in Table I is based on the phase diagram of the $\pm J$ Ising spin glass given in Figure 2. The three values of $q$ illustrate the movement of $\bar{m}_{max}$ along the Nishimori line. $q = 0.8$ was chosen to explore the ferromagnetic phase. $q = 0.109$ was chosen near the multicritical point, the largest value of $q$ for which a ferromagnetic phase may exist. $q = 0.12$ was chosen to explore $\bar{m}_{max}$ in the abscence of a ferromagnetic phase.

TABLE I: Parameters of the simulation: For each system size $N$ we perform $N_{sa}$ averages over a specific amount of disorder $q$. $N_{sw} = 2^b$ is the number of Monte Carlo equilibration and thermalization sweeps, $T_{min}[T_{max}]$ is the lowest [highest] temperature simulated, and $T_N$ is the Nishimori Temperature for a given value of $p$.

| $q$ | $N$ | $N_{\rm sa}$ | $b$ | $T_{\rm min}$ | $T_{\rm max}$ | $T_N$ |
|---|---|---|---|---|---|---|
| 0.08 | 1024 | 1000 | 16 | 0.5 | 2.0000 | 0.818884 |
| 0.08 | 2304 | 1000 | 16 | 0.5 | 2.0000 | 0.818884 |
| 0.08 | 4096 | 500 | 16 | 0.5 | 2.0000 | 0.818884 |
| 0.109 | 1024 | 1000 | 16 | 0.5 | 2.0000 | 0.951929 |
| 0.109 | 2304 | 1000 | 16 | 0.5 | 2.0000 | 0.951929 |
| 0.109 | 4096 | 500 | 16 | 0.5 | 2.0000 | 0.951929 |
| 0.12 | 1024 | 1000 | 16 | 0.5 | 2.0000 | 1.003800 |
| 0.12 | 2304 | 1000 | 16 | 0.5 | 2.0000 | 1.003800 |
| 0.12 | 4096 | 500 | 16 | 0.5 | 2.0000 | 1.003800 |

A detailed discussion of the algorithm applied to all system sizes and values of q follows. Without loss of generality, the configuration to be sent is $\epsilon_i = 1\,\forall\, i$, thus $J = 1$. This allows for a simplification in the calculation of (9) to a disorder average of $S_i$. A random initial configuration is generated. A random set of interactions $\bar{J}_{ij}$ is chosen according to the desired probability of error. The system is allowed to thermalize for $2^{15}$ Monte Carlo sweeps and measurements of the observable are recorded for an additional $2^{15}$.

One problem that arises is the spin reversal symmetry of the problem. If the original configuration is $\epsilon_i = 1$, it is possible for the receiver to find a ground state of $\epsilon_i = -1$, due to the fact that both configurations have the same energy. A list of potential solutions follows. The first is making the choice of labeling one spin as correct, for example $\epsilon_0 = 1$, and if the receiver calculates $\sigma_0 = -1$ then the entire configuration is inverted. This solution is undesirable because errors are introduced when the receiver correctly calculates $\sigma_0 = 1$ but the remainder of the configuration is still inverted. A second solution is including an external field in the Hamiltonian in order to influence the spins to point in a preferred direction. This solution was not chosen due to the large long range interactions and extra computation involved in computing the external field terms. A third solution is to fix certain to influence neighbor spins and the configuration as a whole. This method was tested and but due to finite size effects the desired trend in the data is not apperent. The method chosen for this study is to measure the thermodynamic magnetization and simply multiply

the configuration, magnetization, and local fields of each spin by $-1$, essentially inverting the system. $\sum_i^N \sigma_i < 0$ is evaluated after each sweep and before the measurement of observables. The inversion of the system only takes place a few times before settling in the preffered configuration.

After each Monte Carlo sweep, the thermal average of each spin, $\langle \sigma_i \rangle_T$ is calculated by summing the value of each spin. The thermal average is then divided by its magnitude to obtain its sign. Finally the signs of all $N$ spins are averaged. For each sample the average sign of spins is then averaged over disorder using jackknife statistics to obtain errorbars.

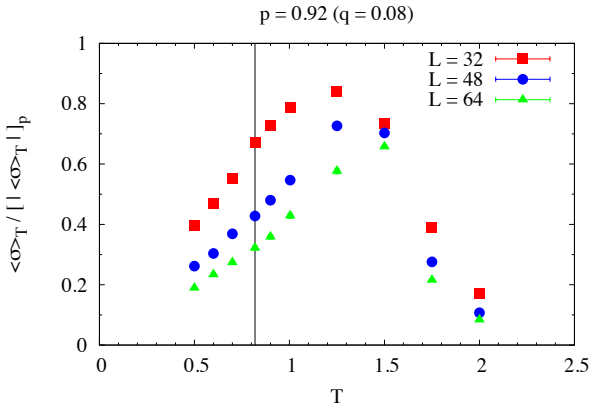



FIG. 3: For $q = 0.08$, the Nishimori Temperature is indicated by a vertical line. Error bars are smaller than the symbols.

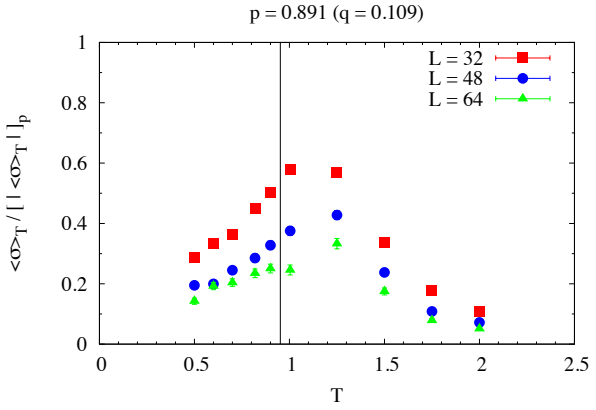



FIG. 4: For $q = 0.109$, the Nishimori Temperature is indicated by a vertical line. Error bars are smaller than the symbols.

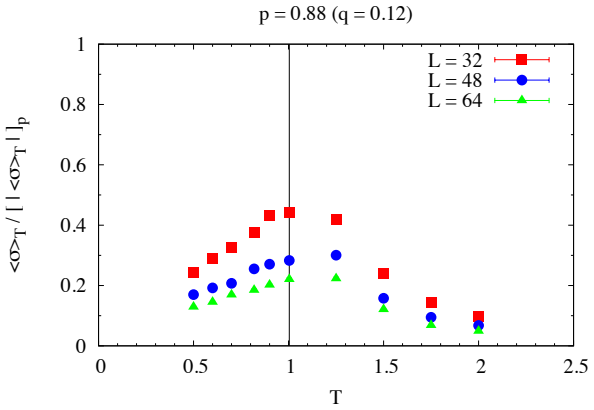



FIG. 5: For $q = 0.12$, the Nishimori Temperature is indicated by a vertical line. Errorbars are smaller than the symbols.

## III. SUMMARY AND CONCLUSIONS

Figures 3, 4, and 5 show the coherence as a function of temperature T for different system sizes. The data show a peak, however it is away from the Nishimori Temperature, indicated by a vertical line. This implies that there does exist a finite temperature at which the system is more coherent but current results show this may not be the Nishimori Temperature for a finite size lattice. Note that the peak does not shift towards the Nishimori Temperature, indicating that larger system sizes than the ones simulated will not improve the location of the peak on the temperature axis.

Future work includes applications to higher dimensions and chimeral topologies similar to those found in the D-Wave One and D-Wave Two quantum annealer. The D-Wave suffers from calibration errors in the interactions between qubits. The obvious application is to determine the best temperature to achieve cohernece despite these calibration errors.

### Acknowledgments

The author would like to thank Zheng Zhu, Ross McDonald, Ruben Andrist, and Helmut Katzgraber for their insightful conversations. We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] H. Nishimori, *Optimum Decoding Temperature for Error-Correcting Codes*, J. Phys. Soc. Jpn. **62**, 2973 (1993).
[2] H. G. Katzgraber, *Introduction to Monte Carlo Methods* (2009), (arXiv:0905.1629).
[3] P. Rujan, *Finite temperature error-correcting codes*, Phys. Rev. Letters **70**, 2968 (1993).
[4] H. Nishimori, *Internal Energy, Specific Heat and Correlation Function of the Bond-Random Ising Model*, Prog. Theor. Phys. **66**, 1169 (1981).
[5] C. K. Thomas and H. G. Katzgraber, *Simplest model to study reentrance in physical systems*, Phys. Rev. E **84**, 040101(R) (2011).
[6] D. Mattis, *Solvable spin systems with random interactions*, Physics Letters **56A**, 421 (1976).

# Solving the Traveling Salesman Problem with the Genetic Algorithm

Casey W. Perkins
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 24, 2014)

The genetic algorithm is extremely versatile and widely used. This study attempts to pose a method for using a genetic algorithm to solve the traveling salesman problem. Multiple methods are posed and discussed. After a single method is selected a study is done to determine the optimal value for the population size and mutation rate for different tour sizes. Genetic Algorithms and simulated annealing are briefly compared at the end of the paper.

## I. INTRODUCTION

The Genetic algorithm is an optimization algorithm that mimics natural selection in order to find the ideal solution of a problem. Genetic algorithms have a wide range of applications including electromagnetics problems and schedule optimization. An initial set of solutions called a population is randomly generated. The traits of each solution are referred to as genes. This population of solutions is allowed to interact with each other via user defined interactions in order to produce new solutions. Admirable traits are preferentially passed on to the children until all the solutions converge to a single solution that hopefully represents the optimal solution. There are three main functions carried out in a genetic algorithm: selection, crossover, and mutation. A flow chart of how the genetic algorithm works is shown in figure 1. The selection process determines which solutions will have an opportunity to pass their genes on to the next generation. Selection can be accomplished by either killing the least fit solutions or by preferentially allowing fit solutions to reproduce themselves more often into the next generation. The crossover process employs some method to take two parent solutions and creates a set of children solutions. A mutation changes the genes of a single solution to create a new solution. There is a limitless number of possible ways to define mutations or crossovers.

In this paper I will propose an algorithm that optimizes the Traveling Salesman Problem (TSP). The TSP is an optimization problem that poses a set number of coordinate pairs that we will call "cities" from here on out. The TSP produces a list of these cities called a tour, and requires a person to visit every single one of these cities while traveling the shortest distance possible and ending in the city where the person started. A person can easily solve the TSP for a small set of tours, but the solution set of the traveling salesman scales as $(N-1)!$ (where $N$ is the size of the tour). That means that while a tour of five cities only has 24 possible solutions, a number that can easily be sifted through individually by a computer, a tour of 30 cities has a solution set of over $10^{30}$. Cycling through every possible permutation of cities would be a nightmare even with a supercomputer. If your computer could compare one trillion solutions per second, it would still take over one trillion years to work your way through the entire solution space. That's where our genetic algorithm comes in. By calculating and crafting solutions based on the fitness of their predecessors, we can skip investigating all of the solutions and hone in on our most likely candidates. Defining a mutation strategy is simple: switch two cities. The real struggle in implementing a genetic algorithm lies in defining an effective crossover method and deciding how to calculate the fitness of the population. After defining how the blocks of the genetic algorithm will execute, there are two parameters to set: the mutation rate and the population set. The mutation rate is the percentage of solutions that will undergo a mutation after they are produced by the crossover operation. The population size determines how many solutions you allow to interact at any given time. In this paper I analyze a few different methods for solving the TSP with the genetic algorithm and compare their merit. I will then discuss the effect on the convergence rate of the algorithm due to changing the population size and mutation rate. Finally I will compare my algorithm to another strategy, simulated annealing, for solving the TSP.

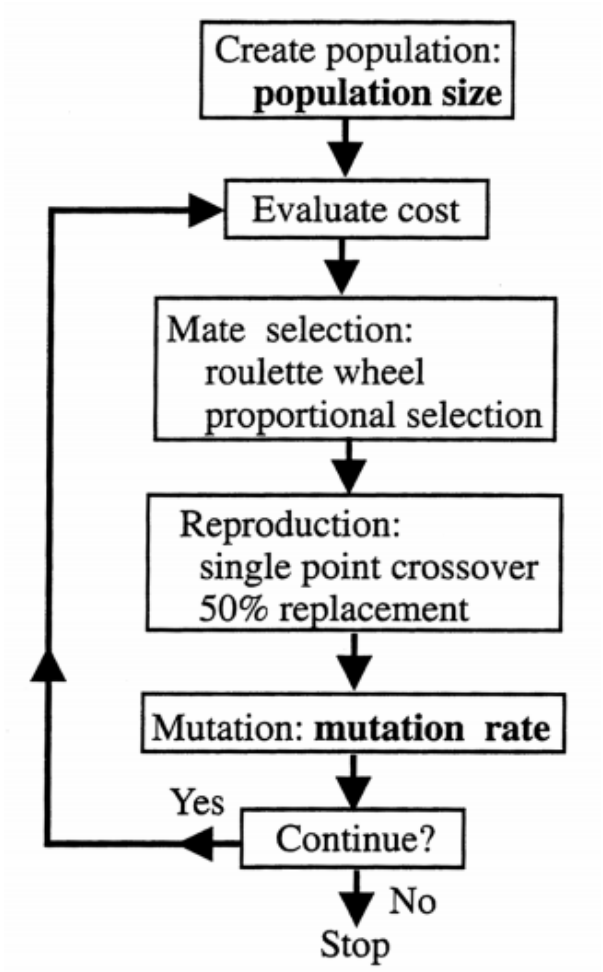



FIG. 1: The genetic algorithm starts with calculating the fitness and then based on that fitness picks mates which are then used to produce children. This process continues until a satisfactory solution is found. [1]

## II. SETUP

The first thing I had to decide was how I was going to go about calculating the fitness of each solution. The fitness must be maximized in the optimal solution, so just taking the length will not work. The initial method I used for determining the fitness was by simply taking the inverse of the length.

$$fitness = \frac{1}{Length} \tag{1}$$

After a few runs I realized that this method was not differentiating strongly enough between solutions during the selection process. The second method I tried, which converged much more quickly, was to pick a value higher than the longest possible path and subtract the length from that value.

$$fitness = 10000 - Length \tag{2}$$

Although this seems less rigorous because the possibility of having a fitness that is negative, this definition actually converges much faster, as long as the value is large compared to the fittest solution the any negative fitness values fall out within the first generation and cease to be an issue. the selected parents for all of my algorithms using a fitness dependent probability. [2]

```
roll = total_fitness * drand();
accumulator = 0;
for(i=0; accumulator <= roll; i++)
{
    accumulator += fitness(population[i])
       - min_fitness;
}
parent = i-1;
```

Each of my algorithms defined the mutation method to switch two cities in the tour. I proposed three different crossover methods.

## III. PROPOSED CROSSOVER METHODS

For the first method I represented each city by a number $0-(N-1)$. I initialize an array the size of my population and fill that array with another set of arrays that are the size of my tour. Each of those nested arrays are filled with each number in the range of $0-(N-1)$. The lower arrays each represent a single member of the populations and the index of those arrays represented when the city that is stored in the element is visited. i.e. the 0th element is the first city visited, while the $N-1$ element is the last city visited. So if I store the number 5 in the $6^{th}$ element of the $3^{rd}$ array, then that means that in the $3^{rd}$ solution the city that I chose to be represented by the number 5 is visited $6^{th}$. This representation makes executing a mutation very simple. Two cities are switched by swapping the values stored in those array elements.

```
int s1 = rand() % TourSize, s2 = rand() % TourSize;
swap child[s1], child[s2];
```

As long as each number appears in each solution only once then you can be certain that you have a valid and complete tour. This representation ended up being non-ideal because it makes searching for links rather difficult. To know if two solutions share a common link must be done by searching through the array $N$ times, which is slow and impractical. This issue arises from the fact that identical tours can be shifted by one element and appear by initial inspection to be completely different. The crossover method for the first method I wrote took two parents and selected a city at random. It compared the

distance between the randomly selected city and the next stop in both solutions and placed the shorter connection in both parents and passed the two resultant solutions on to the next generation.

```
/*randomly select a city*/
int site = rand() % TourSize;
for i = 1 ... N do: /*find the selected city falls
    if ( parent1[i] == site){ index1 = i;}
    if ( parent2[i] == site){ index2 = i;}
child1 = parent1;
child2 = parent2;
if ( Length ( parent1[index1], parent1[index1+1])
< Length ( parent2[index2], parent2[index2 +1]))
    swap parent2[index2+1] with parent1[index1+1];
endif
else
    swap parent1[index1+1] with parent2[index2+1];
endelse
```

This crossover does not converge to the optimal path consistently. This method does not take into consideration that sometimes the optimal solution does not require each city to be connected solely with the neighboring cities that are closest. Also one of the solutions produced only differs from one of the parents by one connection and the other is an identical copy of the other. This causes the algorithm to converge more slowly especially if the initial population doesn't contain optimal connections. The second method uses the same representation for each solution as the first. The second method I created took two parents and creates a child by randomly selecting a segment of cities from one parent and copies that segment into the child. It then fills the remaining cities in the order that they appear in the second parent [3]. The second crossover method is demonstrated graphically in figure 2. This method did not consistently produce children whose fitness exceeds their parents, and therefore converged much too slowly. Taking random segments from each parent does not discriminate between good traits and bad traits in parents. As stated above this does not test for favorable links, so it heavily relies on randomness to arrive at the solution. This fault causes the solution to converge extremely slowly.

The final and most efficient algorithm was also the most complicated to implement. Like above each solution is represented by an array of size $N$. The index of the array represents the city that is assigned to that number. The value that is stored in the array represents the city that will be visited next; i.e., if the number 5 is assigned to the $8^{th}$ element of the array, then that means that the city assigned to number 8 will be followed by the city assigned to number 5. This representation is favorable in that it is easy to see common links. All you have to do to see a common link between two solutions is compare each element of the array. This only requires you to loop through the array once. The down side is this representation was much less intuitive. For example to switch two cities it is necessary to switch the array

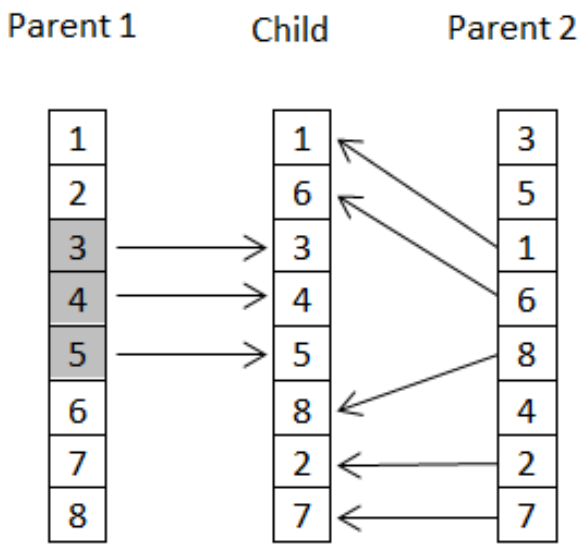



FIG. 2: this diagram demonstrates the crossover methodology of my second algorithm.

elements associated to that city, but you also must swap the numbers each time they occur in the solutions.

```
int s1 = rand() % TourSize, s2 = rand() % TourSize;
for i = 1 ... TourSize do:
    if (child[i] == s1)
        child[i] == s2
    endif
    else if child[i] == s2
        child[i] = s1
    endif
done
swap (child[s1], child[s2])
```

Another complication is that testing for a valid tour is no longer trivial. Because the stops are not listed sequentially, an invalid tour may arise when miniature, independent circuits arise. For the crossover, this method compares two parents and copies each connection that is shared by both parents into the child, and then it alternates between the two parents and fills in empty connections in the child. After that, all empty elements are filled at random by cities pulled from the pool of unassigned cities [4]. With this representation it is possible that an invalid tour of cities will be generated. It is possible that two complete but independent tours will be created. To prevent this every insertion is checked before it is accepted to ensure that a valid tour is created.

```
\*take every link found in both parents*\
for i = 1 ... TourSize do:
    if ( parent1[i]==parent2[i])
      child[i]=parent2[i];
    endif
```

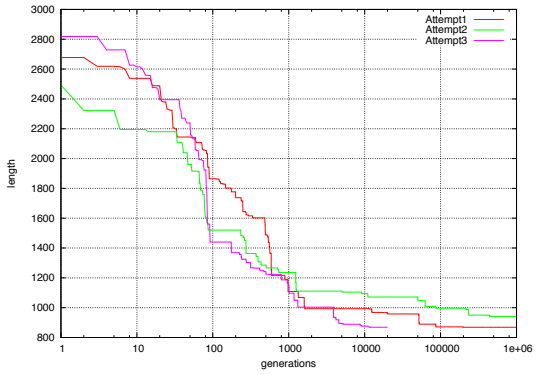



FIG. 3: The length of the optimal solutions of the three methods vs. the number of generations that have passed.

```
done
for i = 1 ...TourSize do:
  if(child[i] isn't filled && switch is valid)
    every odd iteration child[i] = parent1[i];
    every even iteration child[i] = parent2[i];
  endif
done
for i = 1 ... TourSize do:
  while ( child[i] is empty )
    int site = rand() % TourSize;
    if (child[i] = site makes a valid tour)
      child[i]=site;
    endif
  endwhile
done
```

This algorithm preferentially passes traits that are found in both parents. This ensures that the next generation will possess traits that are commonly held by the fittest solutions of the previous generation. The down side to this new algorithm is that new connections are only generated by mutation. This makes the genetic algorithm unlikely to work its way out of metastable states once stuck. To compare the three different methods I attempted to optimize the same thirty city tour with each method. I set the population size to be ten and the mutation rate to be 20%. The third method converges the fastest by far. Method 3 converged in only 10,000 generations, method 1 took almost ten times as long to converge, and method 2 was not able to find the optimal solution in the first 1,000,000 generations. The first and third method both converged to the minimized value of 868.61. The second didn't even breach a final path length of 941.

## IV. OPTIMAL MUTATION RATE AND POPULATION SIZE

Setting the mutation rate and population size has a large effect on the convergence rate and picking a bad combination of mutation rate and population size can actually prevent a good algorithm at all. A larger population brings larger diversity to the initial population. This trait makes the algorithm less likely to get stuck in a metastable state, but makes the algorithm take longer before it settles into equilibrium. A high mutation rate allows solutions to jump out of metastable states by injecting jolts of entropy into the genome, but a mutation rate that is too high will overwhelm the crossover procedure and cause the algorithm to diverge or just converge too slowly to be useful. I tried to optimize a small tour of twenty different cities using my third algorithm for ten different mutation rates in the range of 5%-50% for each of the ten different population sizes in the range of ten to one hundred solutions. I took the average over ten runs for each combination of the mutation rate and population size in those ranges. In my data I considered that each crossover executed as one unit of time. I decided not to count generations, because a generation for a population size of one hundred takes substantially longer than for a population of ten. This convention considers the difference in time spent executing mutations and selecting parents is negligible for different population sizes and mutation rates compared to the time spent performing crossovers. The results are displayed as a heat map in figure 4. The value of the color gradient is determined by the amount of time the algorithm takes to arrive at a near optimal solution I defined this as within fifty length units of the actual optimal solution. The algorithm converged most quickly for a population size of twenty and a mutation rate of 10%. For small population sizes the optimal mutation rate was slightly higher than that for larger population sizes. This makes sense because large population sizes start with more diversity and for that reason already take more time to shake down to an optimal solution. I thought that larger populations might converge more efficiently for problem sizes with a larger solution space, so I ran a select combination of population sizes and mutation rates on the seventy-two city tour we solved previously by using simulated annealing. None of the algorithms converged within the time allocated to the optimal solution, but contrary to my hypothesis the smaller population sizes and mutation rates still converged more quickly. In fact the parameters that converged the most quickly of those that I tested were a population size of ten and a mutation rate of 10%. This result is supported by a paper by Haupt [1]. Ultimately the best values for the parameters are highly dependent on the problem that is being addressed. Compared to simulated annealing the genetic algorithms I created converge much more slowly. My genetic algorithms ran for over twelve hours of real time and still did not arrive to the optimal solution that simulated annealing found in

about fifteen minutes. This is largely due to the genetic algorithms propensity to stall out in metastable states. Whereas simulated annealing is meant to occasionally jump up to a worse solution in order to break free of metastable states, the genetic algorithm relies only on random mutation and solutions that undergo a mutation that make them a worse solution are often immediately killed off by our selection function. When a metastable stable is deep it will often take several mutations that actually decrease the fitness of the solution before it is able to get free. This issue in my algorithm may be fixed by giving the solutions a larger jolt by each mutation by defining the mutation to switch multiple cities.

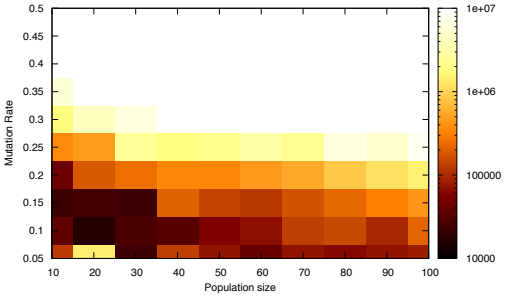



FIG. 4: Heat map that shows the convergence rate of the genetic algorithm for different values of the population size and mutation rate.

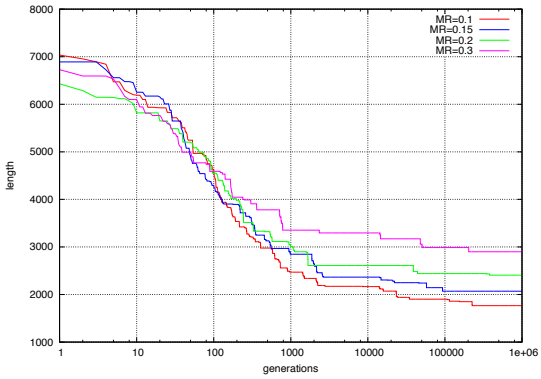



FIG. 5: tracks the best solution for four different iterations of the genetic algorithm optimizing the 72 city tour for four different mutation rate and a population size of 100.

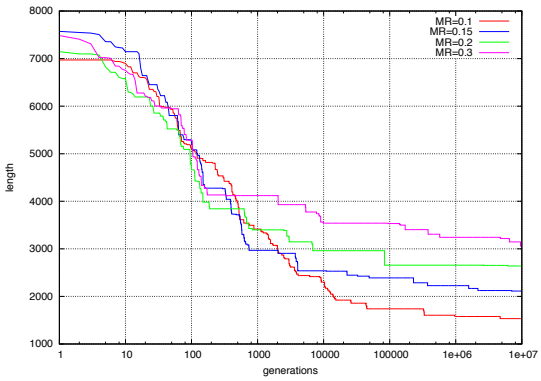



FIG. 6: tracks the best solution for four different iterations of the genetic algorithm optimizing the 72 city tour for four different mutation rate and a population size of 10.

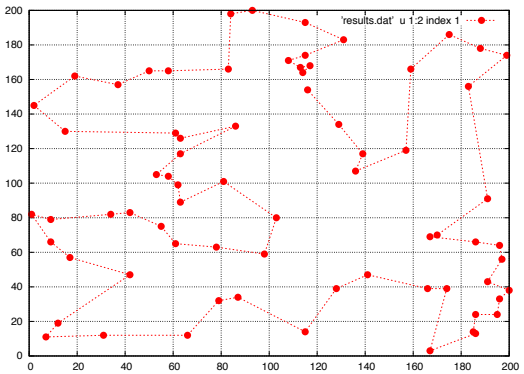



FIG. 7: The optimal path for the 72 city tour as found by simulated annealing.

## V. SUMMARY AND CONCLUSIONS

In this paper I proposed and compared multiple methods of applying the genetic algorithm to the TSP. the genetic algorithm is best used for finding near ideal solutions quickly. The genetic algorithm scales especially poorly for problems with a large number of deep metastable states. I found data supporting that smaller population sizes and medium mutation rates are superior to large population sizes and small mutation rates. Possible ways that I could have improved upon my algorithms is by changing the definition of fitness to better highlight positive traits and change my definition of the mutation operation in order to give the solutions larger jolts in order to help dislodge them from metastable states. In

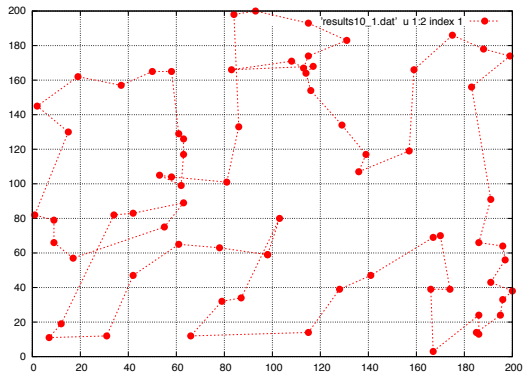



FIG. 8: The best path found by the genetic algorithm. Ran with a population size of 10 and a Mutation rate of

conclusion, I have shown that while there are potential methods that can theoretically optimize the TSP and even very effectively optimize short tours, there are better approaches to the TSP, such as simulated annealing. The genetic algorithm works well to quickly find near optimal solutions to problems, but when the problem requires the absolute best optimization, other algorithms are probably better suited.

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] R. Haupt and S. Haupt, *Optimum Population Size and Mutation Rate for a Simple Real Genetic Algorithm That Optimizes Array Factors*, ACES Journal **15** (2000).

[2] H. Katzgraber, *Optimization and complexity @ONLINE* (2014), URL `katzgraber.org/teaching/SS14/`.

[3] L. Jacobson, *Applying a genetic algorithm to the traveling salesman problem@ONLINE* (2012), URL `http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-5`.

[4] M. LaLena, *Traveling salesman problem using genetic algorithms @ONLINE* (1996), URL `http://www.lalena.com/AI/Tsp/`.

# Wang -Landau algorithm for the 2D Potts model

Laziz K. Saribaev
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 26, 2014)

Wang - Landau algorithm is presented and applied for two-dimensional Potts model. Algorithm uses random walk in energy space to obtain the density of states for the Potts model. Density of states converges to true value using iterations of modification factor $f$ which first is taken as value close to $e$ and then for subsequent iteration accepts smaller value. Direct calculation of the density of states allows to use formulas from statistical mechanics to find out average values for thermodynamic quantities such as internal energy, magnetization ets. Algorithm is then applied to calculate the density of states of $Q = 8$ and $Q = 10$ Potts model where contrary to the Ising model spins are allowed to have $Q$ values from 1 to $Q$.

## I. INTRODUCTION

Computer simulations have become very interesting topic in condensed matter physics because of their versatile applications. One of them is to study phase transitions and critical phenomena which happen at low temperatures. Metropolis algorithm appears to be the first algorithm to do this job [1]. During experiment Metropolis algorithm creates representative small ensemble and uses it to find thermodynamic averages. Appearance of metastable states at first order transitions and critical slowing down at continuous transitions do not let Metropolis algorithm to sample energy states. This happens because of Boltzmann criteria used in the algorithm to become very small and it becomes locked for long time in one of the multiple probability maxima states, not giving possibility to sample other states. Multicanonical ensemble method [2] was suggested to overcome the tunneling barrier between coexisting phases at first-order transitions. Cluster methods [3, 4] were implemented to reduce critical slowing down. In order to reach proper accuracy re-weighting techniques [5] were used.

All these algorithms gave good results but they suffer from systematical errors growing up and not letting to properly calculate probabilities for large scale systems. Wang-Landau algorithm solved this problem and gave good performance for large-scale systems. In their paper [6] Wang and Landau were able to calculate the density of states for up to 256x256 2D Ising and 200x200 2D Potts systems whereas Lee's results for entropic modelling gave good results up to 24x24 Q=10 2D Potts system and 4x4x4 3D Ising model [7]. After the density of states is known it is simple matter to find thermodynamic quantities. In this report Wang-Landau algorithm is implemented using C programming language.

## II. DESCRIPTION OF WANG-LANDAU ALGORITHM

Wang-Landau algorithm uses random walk in energy space with a "flat histogram". This is implemented by choosing at random spin and changing its value according to the probability which is proportional to the reciprocal of the density of states corresponding to the energy $E$ associated with resulting spin configuration. This condition makes algorithm quickly sample all energy levels which usually takes long time in case of unbiased random walk in energy space. First, $g(E)$ of each energy level is set equal to 1. Then at random choose spin and choose at random spin value, calculate energy of whole system associated with this spin change giving for example $E_2$. Then the transition probability from state $E_1$ to $E_2$ is defined as

$$p(E_1 \to E_2) = min(\frac{g(E_1)}{g(E_2}, 1).$$

This implies that spin configuration with $E_2$ is accepted if $g(E_2) \leq g(E_1)$, otherwise it is accepted with probability $\frac{g(E_2)}{g(E_1)}$. This means we generate random number between 0 and 1 and compare it with $\frac{g(E_2)}{g(E_1)}$, if the random number is smaller, then $E_2$ state is accepted. Per each visit $g(E)$ is increased by multiplying by the predefined value $f$ which is usually accepted as $e$. $g(E)$ is thus continuously changing its value during the experiment as $g(E_2) \to f \times g(E_2)$. If move from $E_1$ to $E_2$ is not allowed then $g(E_1) \to f \times g(E_1)$ is done. During the random walk the a number of visits of each energy level is recorded as a histogram $H(E)$, meaning $H(E) \to H(E)+1$. Number of visits per energy level must be not less then predefined percentage of histogram average which is usually between 0.8 and 1. After each energy histogram bin reaches the histogram average, modification factor $f$ is changed as $f \to \sqrt{f}$. After each modification of $f$ all histogram values are zeroed. But we still keep record of $\ln(g(E))$. This makes one iteration of the density of states. We keep going this until $f$ reaches value 1.00000001. Therefore, we will have altogether 27 iterations during which density of states $g(E)$ slowly converges to its true value. Usually the density of states is very large number, reaching for example for 10x10 Ising spin system value of $2^{100} = 1.3 \times 10^{30}$. Because of this it is convenient to work with $\ln(g(E))$ giving convenient double numbers operations for any programming language. The check on the flatness of histogram is done after 10000 sweeps as it was done in original paper [6]. Here one sweep means $N \times N$ random

choices of spin and changing its value and doing above explained probability transition check. Here $N \times N$ is the number of spins.

The resultant density of states must be normalized which is usually done by knowing the number of states for the ground level. In case of Q=10 Potts model it is equal to Q. Then normalized density of states becomes $\ln(g_n(E)) = \ln(g(E)) - \ln(g(E = -2 \times N \times N)) + \ln(Q)$.

After the density of states is found it becomes simple matter to calculate thermodynamic quantities. First, we find internal energy by using

$$U(T) = \frac{\sum_E E \times g(E) e^{-E/k_B T}}{\sum_E g(E) e^{-E/k_B T}}$$

Then specific heat is found from formula for the thermal fluctuations

$$C_v = \frac{< E^2 > - < E >^2}{kT^2}$$

Then by using

$$F = -kT \ln(Z)$$

where $F$ is the free energy of the system, entropy can be calculated using

$$S(T) = \frac{U(T) - F(T)}{T}$$

Here

$$Z = \sum_E g(E) \exp(-\beta E)$$

is the partition function of the system.

## III. APPLICATION OF THE WANG-LANDAU ALGORITHM TO TWO DIMENSIONAL POTTS MODEL

The problem originally proposed to Potts by Domb was to regard the Ising model as a system of interacting spins. Then the generalization was to consider a system of spins confined in a plane, with each spin having $q$ equally spaced directions specified by angles. Potts was able to find the critical point of this model on square lattice for $q = 2, 3, 4$. But together with this he included the critical point for all $q$ of the model $H = -\sum_{i,j} \varepsilon \delta_{i,j}$. This model has the name of Potts model. This model showed afterwards wide application to the grain structure formation because of the similarities of space-filling arrays of cells that evolve to minimize boundary area [8]. This finds applications in the simulations of microstructural evolution. In our case the hamiltonian of the system is similar to above with $\varepsilon$ =1 corresponding to ferromagnetic case

$$H = -\sum_{i,j} \delta(q_i, q_j).$$

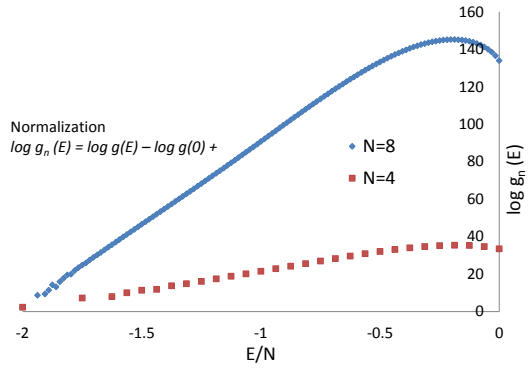



FIG. 1: Logarithm of the density of states $g(E)$ for the Q=10 Potts system for N=4, 8 as function of $E$ per lattice site.

Here q= 1,2, ..Q.

The concrete implementation of the Wang-Landau algorithm becomes as following:

1) initialize spin system as each spin having random Q values from 1 to Q,

2) choose at random spin from 1 to $N \times N$ creating trial state,

3) measure energy of the trial state,

4) decide to choose or not the trial state with transition probabiltiy $p(E_1 \to E_2) = min(\frac{g(E_1)}{g(E_2}, 1)$,

5) compute $\ln(g(E))$, $H(E)$ of the resulting trial state,

6) repeat steps (2) -(5).

During the course of this simulation modification factor $f$ and histogram $H(E)$ are monitored as it was described earlier. We also bin the energy values since energy levels are enumerated for these systems and equal to $2 \times N \times N$. In order to have positive indices we lift energy bins to that value.

2D problem is better handled by transforming it to 1D by enumerating all spins from left to right and continuing enumeration on next rows. In this case so called neighbour matrix must be generated which is 2D array in the form $nb[j][i]$, where $j$ is the number between 1 and 4 showing the neghbour ( 1 is "up", 3 is "down", 2 and 4 are "right" and "left" neighbours), and $i$ denotes site number in 1D line.

In Fig.1 $\ln g(E)$ vs $E/(N \times N)$ dependence for $Q = 10, N = 4, 8$ is shown to be well in accordance with low energies corresponding to low $g(E)$ and high energies to high $g(E)$.

The maximum value for case of $N = 8$ is well in accordance with results in [9] being equal to $\exp(145.16) = 10^{64}$, as it should be for $Q = 10, N = 8$ system giving maximal density of states equal to $Q^{N \times N} = 10^{64}$. This correct value of the maximal density of states tells that the implementation of the program is good. As it was described above, we used $\ln(g_n(E)) = \ln(g(E)) - \ln(g(E =$

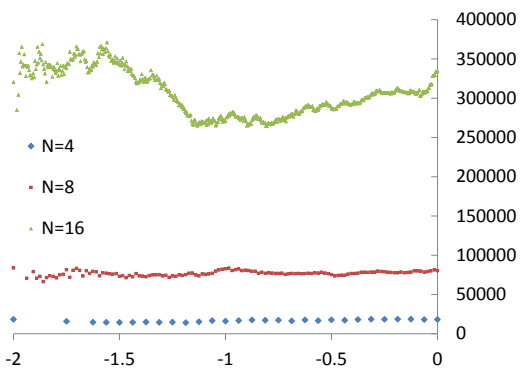



FIG. 2: Final histogram for the case of $Q = 8, N = 4, 8, 16$.

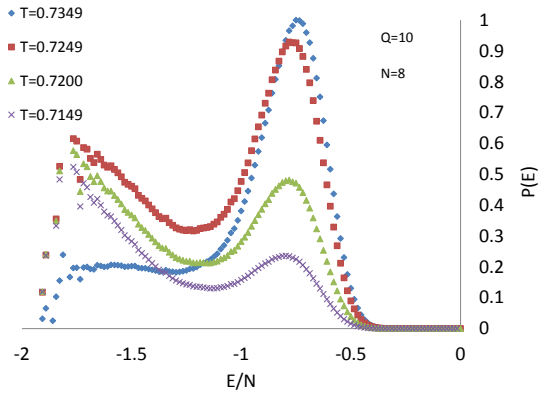



FIG. 3: Canonical probability distribution evolution through $T_c$ for $Q = 10, N = 8$.

$-2 \times N \times N)) + \ln(Q)$ normalization for the density of states.

In Fig.2 final histogram of energy for the last iteration of random walks for the case of $Q = 8, N = 4, 8, 16$ is shown. It describes total number of visits per each energy level in the final iteration of the $g(E)$. Usually it is not possible to keep all histogram values uniform and algorithm allows to keep at 0.8 value, that is each histogram value shouldn't be smaller than 0.8 of average histogram value.

Potts model is known to have double-peaked canonical probability distribution at the transition temperature $T_c$ for the 1st order transition. We compute it at the end of simulation having known all the density of states $g(E)$ using formula

$$P = g(E)exp(-\beta E)$$

It is interesting to look at the evolution of the double-peaked canonical probability distribution starting from a above transition temperature $T_c$ to temperatures lower than $T_c$. Fig.3 shows this evolution for the case of $Q = 10, N = 8$ case. It can be seen from figure that at T=0.7249 second peak is starting to appear. Probability distribution for $T = 0.7200$ appears to be very close to the transition temperature, whereas $T = 0.7149$ has passed through $T_c$. Usually, at transition temperature two peaks are of same height. This can be seen in the Fig.4 where we simulated canonical probability distribution for the case of $Q = 8, N = 16$ with $T_c = 0.7519$. Double-peaked behavior of the probability distribution usually signals about coexistence of two phases near the transition temperature - ordered and disordered ones. As the system's scale gets higher, that is $N$ gets higher, more these two peaks become pronounced and the dip in between them gets lower and lower. It is exactly this feature of the canonical probability distribution function which doesn't allow good sampling in other than Wang-Landau algorithms because extremely long time is required for

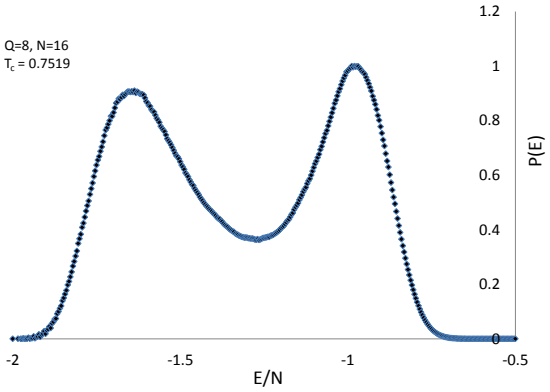



FIG. 4: Canonical probability distribution for $Q = 8, N = 16$.

the system to travel from one peak to the other in energy space. With Wang-Landau experiment this is overcome because of the random walk in energy space biasing visiting all energy levels uniformly.

Fig.5 shows the temperature dependence of the internal energy in vicinity of the transition temperature. For Potts model the transition temperature is found from relation

$$T_c = \frac{1}{\ln(1 + \sqrt{Q})},$$

which for the case of $Q = 8$ is equal to 0.7449 depending in addition also to the value of $N$. it is seen that transition temperature has correct positioning. This step - like behavior of the internal energy at the transition temperature shows first-order phase transition from ordered phase to disordered phase.

Likewise, temperature dependence of the specific heat

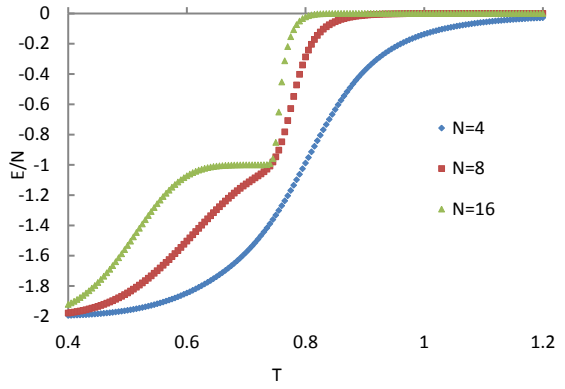



FIG. 5: Internal energy per spin vs T for the case of $Q = 8, N = 4, 8, 16$.

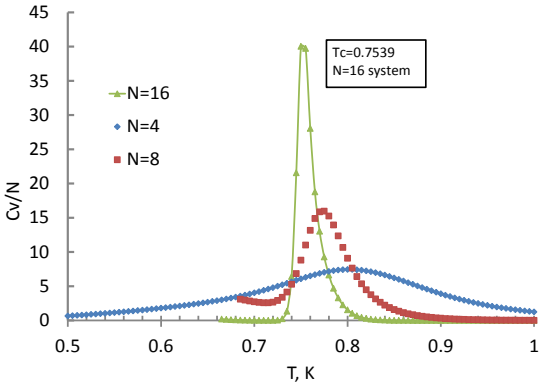



FIG. 6: Specific heat per spin of the $Q = 8$ Potts system for $N = 4, 8, 16$.

also shows as the peaking at the transition temperature which is shown in Fig.6. The additional bump at lower than transition temperature doesn't have to be there. It may happen because it was derived from derivative of the internal energy, not from the fluctuations formula. This can be solved by changing in the program which involves putting several Markov chains and thus evaluating mean $< E >$ and $< E^2 >$ and calculating the specific heat from fluctuations.

Finally, the temperature dependence of entropy per spin for $Q = 10, N = 4, 8, 16$ is shown in Fig.7. Like the internal energy, it has step-like change at $T_c$.

For both cases, in temperature dependence of the internal energy and entropy, step-like changes will get sharper if $N$ gets larger thus approaching the thermodynamic limit behaviour. It is possible to find latent heat from these figures.

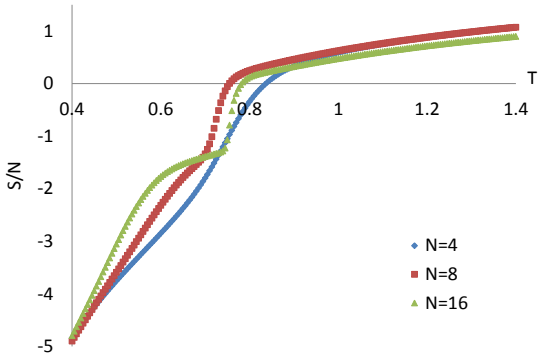



FIG. 7: Temperature dependence of the entropy of the Q=10 Potts system for N=4, 8, 16.

## IV. SUMMARY AND CONCLUSIONS

Efficient Wang-Landau algorithm is described and implemented using C programming language. Because of its specific transition probability in choosing trial site which is proportional to reciprocal of the density of states, algorithm allows quickly sample states and give good results for even large systems. Modification factor $f$ allows to iterate the density of the states so that it slowly converges to its true value, as we saw it in case of maximal density of states for the case of $Q = 10, N = 8$ where we found good agreement between computer experiment and theoretical value of maximal density of states. Energy values for the trial states have integer values, this allows to use these energy levels as the bins with respect to which we can enumerate energy levels even for large systems using the possibilities of computer programming. Correctly found transition temperatures show that the algorithm works fine.

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] N.Metropolis, A.W.Rosenbluth, M.N.Rosenbluth, A.H.Teller, and E.Teller *"Equations of state calculations by fast computing machines"*, J.Chem.Phys. 21,

1087-1092, (1953).

[2] B.A.Berg and T.Neuhaus *"Multicanonical ensemble: A new approach to simulate first-order phase transitions"*, Phys.Rev.Lett. 68, 86-88, (1987).

[3] R.H.Swendsen, J-S.Wang, *"Nonuniversal critical dynamics in Monte Carlo simulations"*, Phys.Rev.Lett. 58, 86-88, (1987).

[4] U.Wolff, *"Collective Monte Carlo updating for spin systems"*, Phys.Rev.Lett., 62, 361-364, (1989).

[5] A.M.Ferrenberg and R.H.Swendsen, *"New Monte Carlo technique for studying phase transitions"*, Phys.Rev.Lett. 61, 2635, 1988. Also *"Optimized Monte Carlo data analysis"*, 63, 1195, 1989.

[6] F.Wang and D.P.Landau, *"Efficient multiple-range random walk algorithm to calculate the density of states*, Phys.Rev.Lett. 86, 2050, (2001).

[7] J.Lee, Phys.Rev.Lett. 71, 211, (1993).

[8] M.P.Anderson et.al. Acta Metall., 32 (5), 783, (1984).

[9] D.P.Landau, S-H Tsai, and M.Exler, Am.J.Phys. 72, (10), 1294, (2004).

# Molecular Dynamics: Breathing Modes

S. Upadhyayula and P. Zhokhov
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 26, 2014)

We studied the breathing modes of various types of molecules under a harmonic potential employing molecular dynamics. We start by studying simple systems that contain two, three and four atoms, as well as a $C_{60}$ buckminsterfullerene (shortened to fullerene) that has a truncated icosahedral structure [1]. We reveal the breathing modes and the energy conservation of these systems with respect to time.

## I. INTRODUCTION

Carbon-based nanostructures, such as graphene, nanotubes and fullerenes have gained a lot of attention due to their unique properties and potential for nanotechnological applications [2–5]. In fact, an entire journal was devoted to fullerenes, nanotubes and carbon nanostructures. While graphene has been discovered relatively recently [2], the discovery of buckminsterfullerene $C_{60}$ goes back to 1985 [6], with its existence suggested in 1966.

Calculating vibrational modes with experimental data plays an important role in determining the structure of $C_{60}$ and $C_{70}$, which were originally discovered as peaks in the mass spectrum analysis of carbon clusters. Furthermore, fullerenes can be viewed as building blocks for nanomaterials, and as such, determination of their vibrational spectrum is an important step in the studies of more complicated nanomaterials.

On the other hand, fullerenes are highly symmetric molecules, thus studying their spectrum is aesthetically pleasing. Fullerenes provide a great playground for group theory techniques and for molecular dynamics of different levels of complexity for the same reason [7–9].

Finally, it has recently been claimed that buckminsterfullerene $C_{60}$ can be an elixir of life - according to [10], it can increase the lifespan of rats by about 90%.

To study the interaction/breathing dynamics of these molecules, we used a couple of standard algorithms available to us. We also started off with far simpler molecules as a proof of concept and then scaled up to the fullerenes.

## II. ALGORITHMS

### A. Dynamics

There are a few symplectic integrators available that we can use. We used the Velocity Verlet algorithm as well as the Forest-Ruth algorithm to compute the time evolution of the coordinates. We did not pursue Runge-Kutta (RK4) algorithm because it does not conserve energy, and we needed something that conserves energy in the system.

We started off with the harmonic potential,

$$\mathcal{U}(\vec{r}_1, \vec{r}_2, \vec{r}_3...) = \frac{1}{2}\sum_{i,j} k_{ij}(|\vec{r}_i - \vec{r}_j| - l_{ij})^2 \tag{1}$$

where we sum $j$ over neighbors of $i$. We define $r_i$ as the position of atom $i$, $k_{ij}$ as the bond strength between atoms $i$ and $j$, and $l_{ij}$ as the bond length between atoms $i$ and $j$.

#### 1. Velocity Verlet

Velocity Verlet is a second order integrator. This method is similar to the leapfrog method with the difference being velocity and position are not calculated at the same time step for leapfrog. The algorithm is as follows:

$$\begin{aligned}
\text{Let } \vec{r}_i{}^{n+1} &= \vec{r}_i((n+1)\Delta t) \\
\vec{v}_i{}^{n+1} &= \vec{v}_i((n+1)\Delta t) \\
\text{then, } \vec{v}_i{}^{n+1/2} &= \vec{v}_i{}^{n} + \frac{\Delta t}{2m_i}\vec{F}_i(\{r_j^n\}) \\
\vec{r}_i{}^{n+1} &= \vec{r}_i{}^{n} + \Delta t\ \vec{v}_i{}^{n+1/2} \\
\vec{v}_i{}^{n+1} &= \vec{v}_i{}^{n+1/2} + \frac{\Delta t}{2m_i}\vec{F}_i(\{r_j^{n+1}\})
\end{aligned} \tag{2}$$

where,

$$\vec{F}_i = -\frac{\partial \mathcal{U}(\{r_j^n\})}{\partial \vec{r}_i} \tag{3}$$

Velocity Verlet has an error in each position step of $\mathcal{O}(\Delta t^4)$, and each velocity step of $\mathcal{O}(\Delta t^2)$. The global error present in both position and velocity is $\mathcal{O}(\Delta t^2)$. In a molecular dynamics situation like the one here, global error is far more relevant than local error. Hence, Velocity Verlet is a second order symplectic integrator.

In principle, it would've been possible to perform this integration using a basic Strömer Verlet integration (without velocities) since all we need are the time evolution steps of coordinates. However, we used Velocity Verlet because knowing the velocities at those steps can be useful in calculating the energies.

#### 2. Forest-Ruth

Forest-Ruth is a symplectic algorithm of the fourth order. It is described (along with a more general approach) in [11, 12]. Let us subdivide a temporal step into four sub-steps and write

$$\begin{aligned} \vec{v}_i{}^{n+1/4} &= \vec{v}_i{}^{n} + c_1 \Delta t \frac{\vec{F}_i(\{\vec{r}_j{}^{n}\})}{m_i} \\ \vec{r}_i{}^{n+1/4} &= \vec{r}_i{}^{n} + d_1 \Delta t \vec{v}_i{}^{n+1/4} \\ \vec{v}_i{}^{n+2/4} &= \vec{v}_i{}^{n+1/4} + c_2 \Delta t \frac{\vec{F}_i(\{\vec{r}_j{}^{n+1/4}\})}{m_i} \\ &\dots \\ \vec{r}_i{}^{n+1} &= \vec{r}_i{}^{n+3/4} + d_4 \Delta t \vec{v}_i{}^{n+3/4} \end{aligned} \tag{4}$$

The crucial step is the derivation of the coefficients $c_1 \dots c_4$ and $d_1 \dots d_4$. In the original paper [11] it has been done numerically and then checked analytically, later more elegant approaches based on the Baker-Campbell-Hausdorff formula have been developed [13]. We use the following coefficients

$$\begin{array}{ll} c_1 = x + \frac{1}{2}, & d_1 = 2x + 1 \\ c_2 = -x, & d_2 = -4x - 1 \\ c_3 = -x, & d_3 = 2x + 1 \\ c_3 = x + \frac{1}{2}, & d_4 = 0 \end{array} \tag{5}$$

where $x = \frac{1}{6}\left(2^{1/3} + 2^{-1/3} - 1\right)$.

### B. Mode Analysis

Let us assume all atoms have unit mass and all bonds have unit stiffness. In order to find modes semi-analytically, we represent potential energy function $U(\{\vec{r}_j\})$ (1) as a bi-linear form in the displacements $\delta\vec{r}_j$ from the equilibrium positions $\vec{r}_{j,0}$. In other words we introduce displacement vector with $3N$ components (where $N$ is number of atoms in the molecule)

$$R = \begin{pmatrix} \vec{r}_1 \\ \vec{r}_2 \\ \vdots \\ \vec{r}_N \end{pmatrix} - \begin{pmatrix} \vec{r}_{1,0} \\ \vec{r}_{2,0} \\ \vdots \\ \vec{r}_{N,0} \end{pmatrix}, \tag{6}$$

and find a $3N \times 3N$ matrix $\mathcal{U}$ such that,

$$\mathcal{U}(\{\vec{r}_j\}) = R^T U R + \mathcal{O}(||R||^4). \tag{7}$$

In principle, $U$ can be defined as

$$U_{ij} = \left(\frac{\partial^2 \mathcal{U}}{\partial R_i \partial R_j}\right)_{\text{equilibrium}} \tag{8}$$

The problem of finding the vibration modes of the molecule is then equivalent to the problem of finding eigenvalues and eigenvectors of the matrix $U$ [14]. Matrix $U$ is positively defined, thus all its eigenvalues $\lambda_m \geq 0$. Mode frequencies are given by $\omega_m = \sqrt{\lambda_m}$, while eigenvectors correspond to the displacement of atoms within the given mode. The mode analysis is performed using MATLAB software package, due to its ready-to-use linear algebra routines for eigenvalue search problems and it's graphics capabilities. The key part of the mode search algorithm is the generation of the matrix $U$:

```
%atomsX(i,:) contains Cartesian coordinates of
%atom i, and bonds(i,j) contains the number of
%j-th neighboor of atom i.
for i = 1:Natoms
    for b=1:Nbonds;
        j=bonds(i,b)+1;
        dX = (atomsX(i,:)-atomsX(j,:));
        eX = dX./norm(dX);
        for nu=1:3
          for mu=1:3;
            %i and j enumerate atoms,
            %nu and mu - Cartesian coordinates
            p = (eX(nu)*eX(mu));
            M(nu,i,mu,i) = M(nu,i,mu,i)+p;
            M(nu,i,mu,j) = M(nu,i,mu,j)-p;
          end;
        end;
    end;
end;

M = M/2;
U = (reshape(M, D*Natoms, D*Natoms));
[V,lambda] = eig(U);
```

### C. Visualization

The Visualization subroutine was implemented in MATLAB as well. The atoms are represented as spheres of color 0 (in a standard MATLAB color scheme 0 stands for dark blue color), while bonds are represented by cylinders of color 1. The radius of the spheres and cylinders as well as number of constituent polygons can be varied easily. When it comes to rendering the animations, we can incorporate slow rotation of the entire scene, allowing us to observe the molecule from different angles.

## III. RESULTS

We started with the trivial case of the two atom system. Here we have six degrees of freedom. Five of these are zero frequency modes that can either be transnational or rotational. Oscillations of the only remaining degree of freedom and the energy conservation are consistent with what is expected from a harmonic potential like the one used here (1).

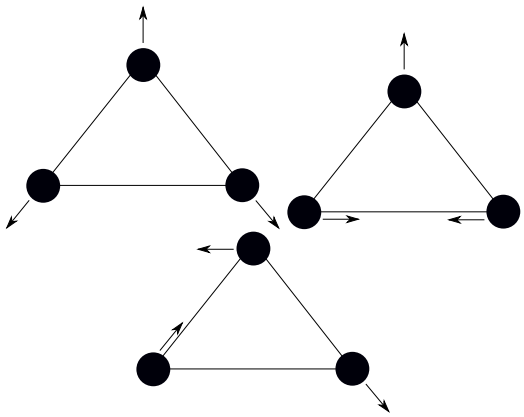


FIG. 1: Here, we can see the way three modes that were excited behave. The first mode has a frequency of $\sqrt{1.5}$. The second and third modes have a frequency of $\sqrt{0.75}$. Since the second and third modes are degenerate, any linear combination of those two modes is possible.

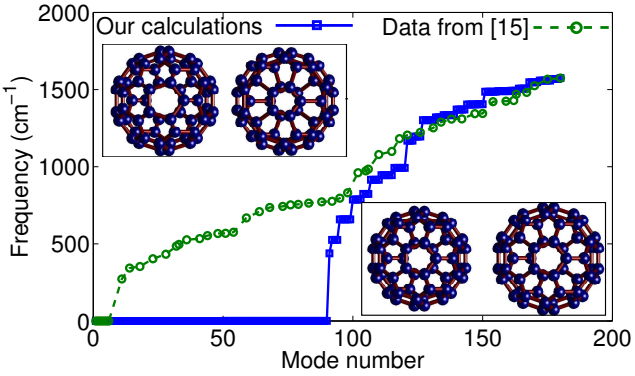



FIG. 2: Vibrational spectrum of the Buckminsterfullerene $C_{60}$ with bond strength and atom masses set to unity. The inset shows the lowest-frequency mode (lower right corner and the highest frequency mode (upper left corner). The solid blue line with squares shows our results, the dashed green line with circles shows the results by [15].

The three atom system is no longer an exercise in triviality. We now have three modes we can excite. Fig. 1 shows the different possible modes of the molecule.

The four atom system has twelve degrees of freedom and six of those correspond to translations and rotations of the system as a whole. Four of these modes can be represented as two adjacent atoms oscillating away from each other, and these have a frequency of 1. Since we do not have a torsion potential, there is no returning force when atoms go in and out of the plane which gives us two more zero frequency modes.

Let us now turn to the study of the buckminsterfullerene $C_{60}$. The mode spectrum is presented in Fig. 2, along with the shapes of the lowest (radial breathing mode) and the highest modes. We note that some of the modes are not directly accessible to standard excitation methods, such as infrared or Raman spectroscopy [4]

Having established the vibrational spectrum, we can use it as a benchmark for the integration schemes. First, let us excite the lowest-order mode and take a look at the energy as a function of time for different time steps. Theoretically the energy should stay constant; we can see that both algorithms provide a constant energy on average. However, Forest-Ruth algorithm provides much smaller deviations of the total energy within the cycle [Fig. 3]. Additionally, we plot relative energy deviation as a function of time step size $\Delta t$ [Fig. 6] to make sure that the Forest-Ruth method error decreases as $\Delta t^4$ and Velocity Verlet method error scales as $\Delta t^2$.

Energy conservation is an intrinsic property and the strong side of the symplectic algorithms, such behavior is to be expected. However, the algorithms do not have to *a priori* respect the mode structure, i.e. the energy distribution among the modes does not have to be conserved. To test this aspect, let us excite a mode of the molecule and look at the energy in the other modes. The-

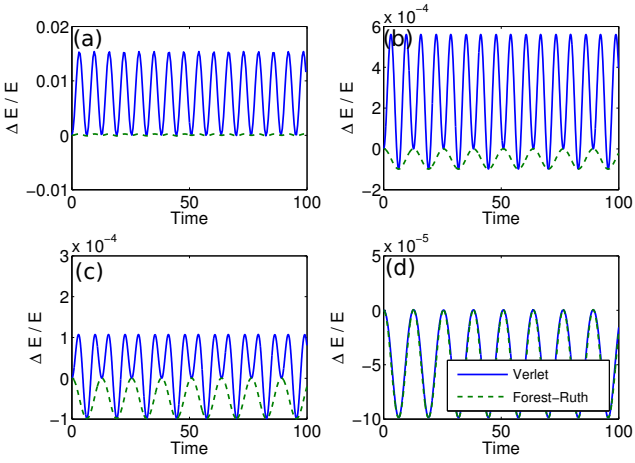



FIG. 3: Relative deviation of the energy of the lowest frequency mode of the $C_{60}$ from the initial value as a function of time for different integration time steps: (a) $\Delta t = 0.5$, (b) $\Delta t = 0.1$, (c) $\Delta t = 0.05$, $\Delta t = 0.01$ obtained with velocity Verlet method (solid blue line) and Forest-Ruth method (dashed green line).

oretically, energy should stay in the same mode while in the numerical approach it can get dispersed into other modes (in principle, for high amplitudes, non-linearity related to the $\mathcal{O}(||R||^4)$ term in the Eq. (7) sets in, and the modes become actually coupled). We see that the portion of energy leaking in other modes is on the order of $10^{-4}$ for the highest-frequency mode, and is on the order of $10^{-3}$ for the lowest-frequency mode [Fig. 5]. With decreasing step-size both methods converge onto a single specific temporal profile of the energy leak, suggesting the nonlinear coupling that follows from the Eq. (1).

Having established the limits of applicability of our numerical integration scheme, let us proceed to some more

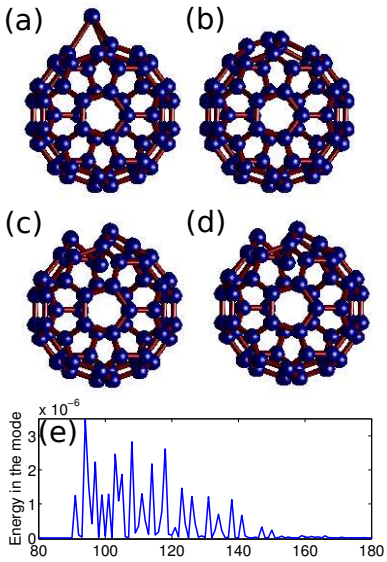



FIG. 4: Dynamics of $C_{60}$ after one atom being pulled out (a) and released (b)–(d), and the corresponding spectrum of such vibration.

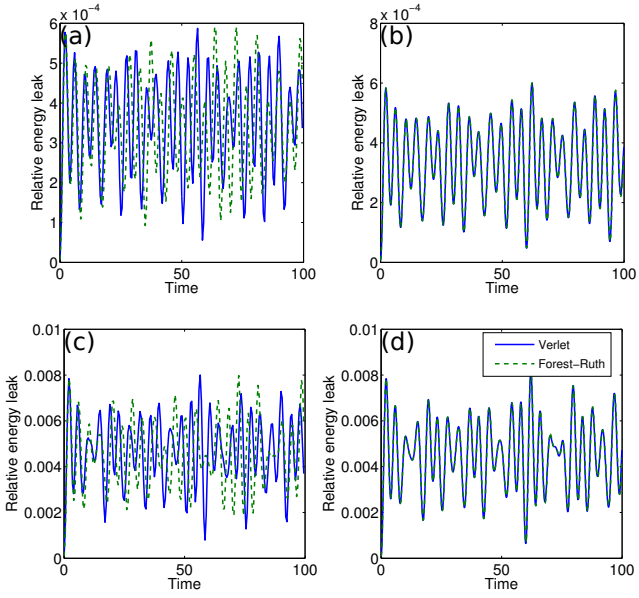



FIG. 5: A portion of energy contained in the modes besides the excited one in the initial condition when calculated with velocity Verlet method (solid blue line) and Forest-Ruth method (dashed green line). Initial condition is (a),(b) highest frequency mode and (c),(d) lowest frequency mode; integration time step is (a),(c) 0.5 and (b),(d) 0.1. The convergence of both methods to the same curve at the smaller time step is indicative of the nonlinear mode coupling, taken into account in the dynamics model, but neglected in the mode analysis.

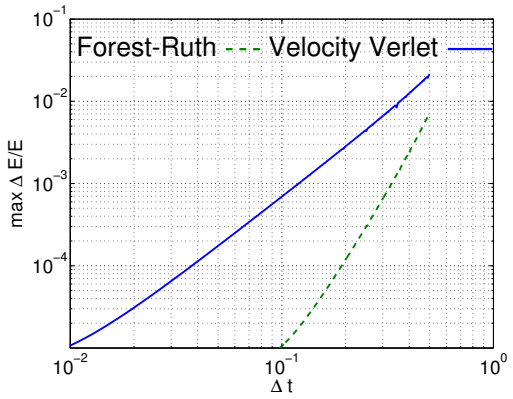



FIG. 6: Relative deviation in energy as a function of time step. Velocity Verlet method (dashed green line) provides second-order convergence, while Forest-Ruth method (solid blue line) provides fourth order convergence.

interesting matters. Our numerical codes integrate the equations starting from any initial condition, and as such it is tempting to try some exotic vibrational regimes. For example, one can take one atom, pull it somewhat away [Fig. 4(a) ]and let go [Fig. 4(b)–(d)]. The spectrum of such vibartion, maintained accurately by the simulations, is shown in Fig. 4.

## IV. SUMMARY AND CONCLUSIONS

As expected, the Forest-Ruth algorithm gave us more precise results than the Velocity Verlet algorithm. With all the different types of molecules, we generated the bonds and initial positions data of the atoms using MATLAB. This data was then fed into the C++ program that contained the algorithms to compute the dynamics. The output from this C++ program contained the time evolution of the coordinates and velocities. This output was then imported into MATLAB to perform visualizations of the respective molecules. We also performed mode analysis to compare our results using MATLAB

For the simple systems we studied, such as molecules with 2-4 atoms, the results of the integration coincided well with the mode analysis. For complex systems like the Buckminsterfullerene, the results of the integration did coincide well with the mode analysis only at the small perturbation amplitudes. The larger was the perturbation, the faster the system tends to deviate from the linear regime described by the mode analysis, suggesting the role of nonlinear coupling effects in the potential (1).

In principle our code is suited for studying vibrational dynamics of carbon systems of any finite size, and, with very few modifications, of any molecule with given equilibrium atom positions and bonding potential.

**Acknowledgments**

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster. We thank Helmut G. Katzgraber for furnishing the resources, background information and affording us the opportunity to work on the project. We also thank Ross McDonald for providing feedback and introducing us to other symplectic algorithms besides Verlet algorithm.

[1] H. W. Kroto, A. W. Allaf, and S. P. Balm, Chem. Rev. **91**, 1213 (1991), ISSN 0009-2665, URL http://pubs.acs.org/doi/abs/10.1021/cr00006a005.

[2] K. S. Novoselov, a. K. Geim, S. V. Morozov, D. Jiang, Y. Zhang, S. V. Dubonos, I. V. Grigorieva, and a. a. Firsov, Science **306**, 666 (2004), ISSN 1095-9203, URL http://www.ncbi.nlm.nih.gov/pubmed/15499015.

[3] E. W. Billups and M. A. Ciufolini, *Buckminsterfullerenes* (John Wiley & Sons, Inc., New York, NY, 1993).

[4] M. S. Amer, *Raman Spectroscopy, Fullerenes and Nanotechnology* (Royal Society of Chemistry, Cambridge, UK, 2010).

[5] H. W. Kroto, J. E. Fischer, and D. E. Cox, *The Fullerenes* (Pergamon Press, Oxford, 1993).

[6] H. W. Kroto, J. R. Heath, S. C. O'Brien, R. F. Curl, and R. E. Smalley, Nature **318**, 162 (1985), ISSN 0028-0836, URL http://www.nature.com/doifinder/10.1038/318162a0.

[7] B. Adams, J. B. Page, O. F. Sankey, and M. O. Keeffe, Phys. Rev. B **50**, 17471 (1994).

[8] D. Jing and Z. Pan, Eur. J. Mech. - A/Solids **28**, 948 (2009), ISSN 09977538, URL http://linkinghub.elsevier.com/retrieve/pii/S099775380900028X.

[9] Z. Cao, Y. Peng, S. Li, L. Liu, and T. Yan, **10**, 3096 (2009).

[10] T. Baati, F. Bourasset, N. Gharbi, L. Njim, M. Abderrabba, A. Kerkeni, H. Szwarc, and F. Moussa, Biomaterials **33**, 4936 (2012), ISSN 1878-5905, URL http://www.ncbi.nlm.nih.gov/pubmed/22498298.

[11] E. Forest and R. D. Ruth, Phys. D **43**, 105 (1990).

[12] E. Forest, J. Phys. A. Math. Gen. **39**, 5321 (2006), ISSN 0305-4470, URL http://stacks.iop.org/0305-4470/39/i=19/a=S03?key=crossref.4272a1ed7fcbbdc214ce5477ce8427d6.

[13] H. Yoshida, Phys. Lett. A **150**, 262 (1990), ISSN 03759601, URL http://linkinghub.elsevier.com/retrieve/pii/0375960190900923.

[14] H. Goldstein, C. Poole, and J. Safko, *Classical Mechanics* (Addison Wesley, San Francisco, 2001), 3rd ed.

[15] J. B. Page and J. Menendez, in *Light Scatt. Solids VIII*, edited by M. Cardona and G. Güntherodt (Springer, Berlin, 2000).

# Parallelization of fractal generation using graphics processing units

Richard M. Vega
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 15, 2014)

Over the past decade there has been significant advancements in the capabilities of graphics processing units (GPU's) to perform tasks that were once relegated to large clusters of central processing units (CPU's). Parallelization of tasks such as matrix multiplication, sorting, and sequence matching are just a few examples of this. With the increase in the capabilities of hardware came a similar increase in the capabilities of software to achieve this general purpose computation on graphics processing units (GPGPU). Programming languages such as OpenCL, OpenGL, and CUDA have been at the forefront of this field. This report focuses on the use of OpenCL to generate images of fractal geometries using a single GPU and compares the run time to the same calculation performed on a single CPU. Parallelization of complex number fractals and iterated function system fractals are compared. A speed up by a factor of nearly 40 is observed.

## I. INTRODUCTION TO FRACTALS

It would be nice to start any discussion of fractals and the theory behind them with a definition. Unfortunately, there is no consistent definition in the literature. Mandelbrot gave his original definition of a fractal as a set of points with a Hausdorff dimension strictly greater than its topological dimension [1]. Other authors may use a different type of dimension such as box dimension or packing dimension to describe a fractal, and some variations of dimension may be more appropriate for certain types of fractals. It is appropriate for the scope of this research to loosely define a fractal as a set of points that has:

- fine structure at all scales
- self similarity
- a "fractal dimension" greater than its topological dimension

The first of these requirements is satisfied if the structure of the fractal has the same level of complexity at all magnifications. The second requirement states that the structure of the fractal should be replicated (although perhaps not exactly) at higher magnifications. The third requirement is simply a re-statement of Mandelbrot's definition, with the freedom to select a valid definition of dimension for the fractal being considered. The dimension of a fractal can be thought of as the amount of space it fills. A classic example of this is the Koch curve which has a dimension of roughly 1.262 [2] and is shown in Figure 1. The Koch curve is constructed by taking a line segment and breaking it in three, replacing the middle segment by two copies of itself, and connecting the two copies to each other and the first and third segments. This is repeated for each resulting segment ad infinitum. In this way, what started as a one-dimensional line segment with finite length, will now have infinite length and be enclosed in a finite area. Clearly, the Koch curve will not fill the two-dimensional space, but it can no longer be considered a one-dimensional object either. It's dimension is somewhere in between one and two.

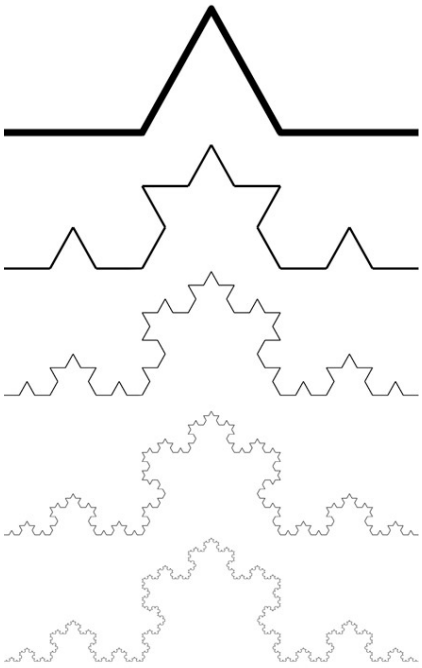


FIG. 1: Koch curve.

The Koch curve is constructed by a very simple set of geometric instructions. A brief Google search will reveal hundreds of other fractals, each with their own simple definitions. Two categories into which many of these fractals fall into are complex number fractals and iterated function system (IFS) fractals. While these two categories do not encompass every fractal in existence, they do contain many of the most widely known fractals. This report will focus on the parallel calculation of paradigm examples of each of these two categories. The two fractals chosen for this purpose are the Mandelbrot

set and Seirpinski's triangle.

The Mandelbrot set is probably the most widely known complex number fractal. A complex number fractal is generated by performing an iterative operation on points in the complex plane. Images of these fractals are then generated by treating each point as a pixel and coloring it according to the result of the iterative operation. The Mandelbrot set is generated by the following recursive formula:

$$z_{n+1} = z_n^2 + c \tag{1}$$

For each point $c$ in the complex plane, Equation 1 is applied recursively starting with $z = 0$, and if $z$ remains bounded, the point $c$ is determined to be part of the Mandelbrot set. It can be shown that if $|z| \geq 2$, the point will diverge [3]. Thus, if at any point in the iteration $|z|$ exceeds this value, there is no longer any reason to continue. If the purpose is to generate an awe inspiring image however, simply coloring pixels depending upon whether or not they diverge will produce a rather dull result. For this reason, points are typically colored based on how many iterations it takes for $|z|$ to exceed 2, giving some information about how quickly each point diverges. Points that do not diverge within the maximum number of iterations are colored black and are determined to be part of the Mandelbrot set. A maximum number of iterations must be chosen and will determine the accuracy of the generated image. For example, a point may diverge very slowly, and if the number of iterations performed is too low, the point will be mistakenly identified as part of the set. The number of iterations necessary to produce an accurate image depends strongly on the magnification of the image. The Mandelbrot set can be seen in Figure 2 at various magnifications showing the fine structure and self similarity at increasing magnifications. The effect of the maximum number of iterations can be seen in Figure 3.

Complex number fractals like the Mandelbrot set lend very nicely to parallelization due to the fact that each point's orbit is independent of any other's. Instead of looping through columns and rows of pixels, and performing the above iteration for one pixel at a time, a batch of pixels could be spread across as many compute units as are available, with each compute unit calculating the orbit of a point or group of points. As can be seen in Figure 2, the complexity of the image grows as the magnification becomes larger. This poses a problem when the scale of the image is comparable to the precision of the machine. This can be especially significant for GPU calculations where double precision floating point values are typically not available on all but the most expensive models.

Where complex number fractals perform an iterative function on all points of interest, IFS fractals repeatedly transform a single point, and plot its position after each transformation. The initial point is chosen at random on a bi-unit square. This process is made more interesting by what is known as the "chaos game." This is

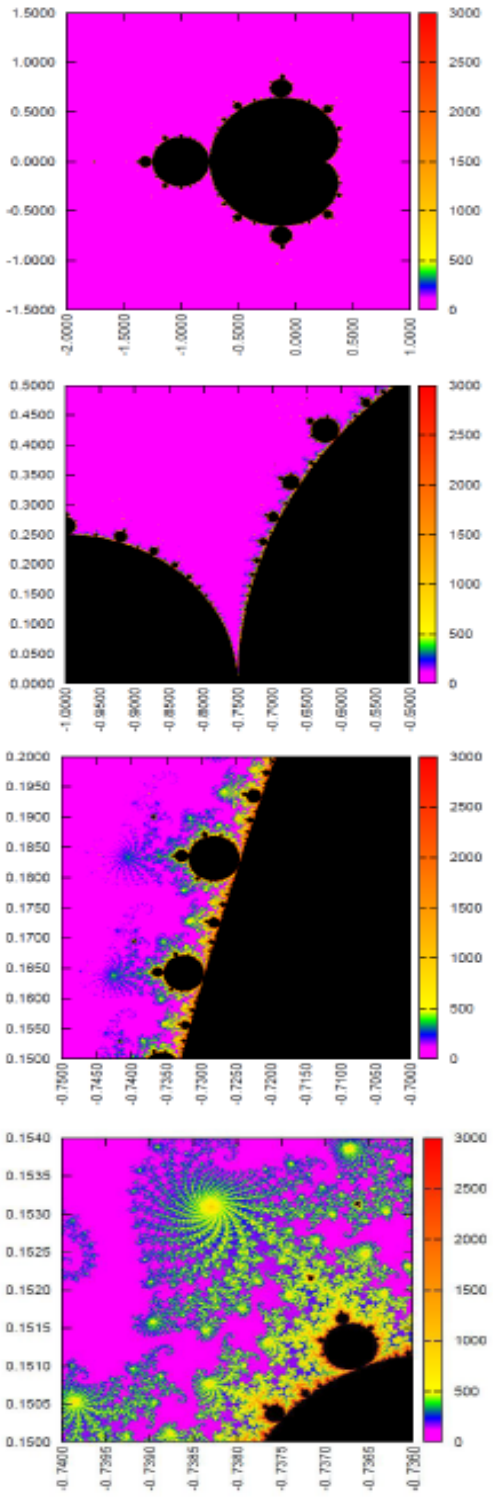


FIG. 2: Mandelbrot set at various magnifications.

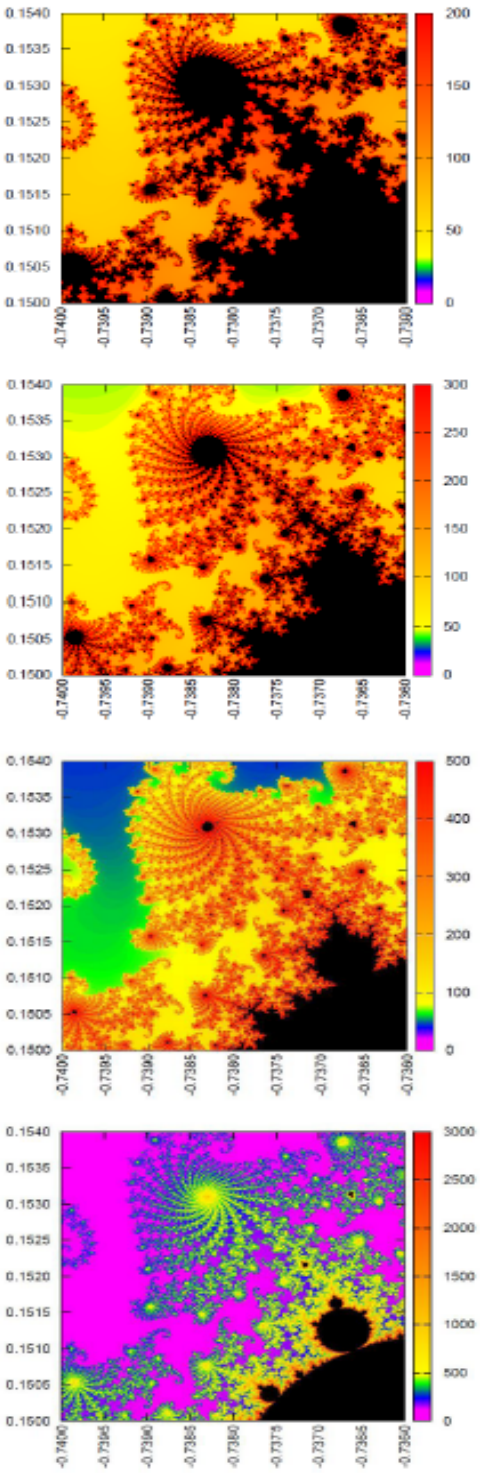


FIG. 3: Mandelbrot set with various maximum numbers of iterations.

carried out by selecting a transform at random, from a list of transforms with set probabilities, for each iteration. One of the most famous examples of this is Seirpinski's triangle shown in Figure 4. To generate Seirpinski's triangle, one chooses three points as the vertices of the outer triangle. Three transformations are then created that transform any random point to the halfway mark between the original location and one of the three vertices. For instance, if the three points chosen are (0,0), (1,0), and (0,1) as in Figure 4, the three transformations will be:

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = 0.5 \begin{pmatrix} x_n \\ y_n \end{pmatrix} \tag{2}$$

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = 0.5 \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.0 \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = 0.5 \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.5 \end{pmatrix} \tag{4}$$

where each transformation is chosen with equal probability. Color can be added as in Figure 4 by associating a color with each transformation, and coloring each point depending upon which transformation the pixel was a result of. Clearly, Seirpinski's triangle has perfect self similarity and fine structure at all scales. The fractal dimension of Seirpinski's triangle is 1.5829 [2] which is between one and two as expected for a set of infinite connected points which does not fill the area that it is enclosed in. The quality of the image of an IFS fractal is determined by the resolution and number of points plotted where a finer resolution requires more plotted points.

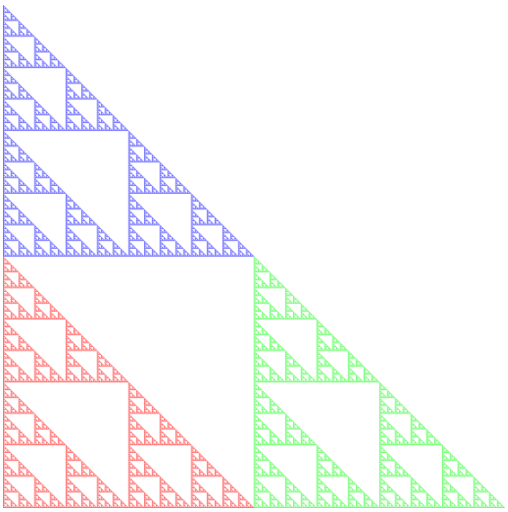


FIG. 4: Seirpinski's triangle.

## II. GPU PARALLEL CALCULATION

As advancements in hardware development produced faster and faster CPU's, it was realized that this increase would not be without limit. Serious impediments loomed in the horizon which included the absolute limit on transmission speed, the size limit of components, and the overall increasing cost of making CPU's faster [4]. The obvious solution was to ditch quality and focus on quantity. Why not find a way to run on more than just a single CPU? At the time, parallelizing programs written in C, C++, and FORTRAN was very complicated, and depended upon interfaces provided by operating systems. Newer standards such as the Message Passing Interface (MPI) and OpenMP for multi-core CPU's, solved this problem by allowing programmers to easily write codes that could run on multiple processing units at once. The two primary methods of code parallelization are data parallelism and task parallelism. Data parallelism is where groups of a large set of data, such as a matrix, are sent to different processing units and each group undergoes the same operation. For the purpose of parallelizing the computation of fractals, this is the type of parallelization that will be utilized. Task parallelism on the other hand is where different tasks are performed at the same time.

Eventually, it was realized that GPU's could be used to speed up data parallelization algorithms. This is because GPU's consist of hundreds of cores that are designed to perform the same task on large numbers of pixels. While there are certainly some things that GPU's cannot do, repetitive mathematics is not one of them. While the potential to use GPU's for more than simply graphics was realized, it took the development of the OpenCL and CUDA platforms to take full advantage of this potential. Typically, OpenGL was used to program GPU's, which acted as a one-way pipeline from the host program to the screen. Although OpenGL has advanced significantly alongside OpenCL and CUDA, it remains suited for graphical applications more so than truly general purpose computing. When deciding between OpenCL and CUDA, the general perception is that CUDA is easier to learn, but is limited in that it only works for NVIDIA GPU's. OpenCL on the other hand works on AMD and NVIDIA GPU's as well as CPU's made by ATI, IBM, and Intel. For this reason, this research uses OpenCL.

When deciding how to parallelize the calculation of the Mandelbrot set, the first inclination was to simply send each point to a compute unit on the GPU. The kernel program that the GPU uses is then simply the same as the code in the inner loop of the serial calculation. Thus, the kernel is written to receive a single point and perform its orbit via Equation 1. If the orbit exceeds the maximum radius, the loop is stopped and the iteration number is recorded. A color is assigned to that point depending upon how many iterations it underwent. One might naively think that if 1000 compute units are working in parallel on individual points, there would be a speed up by a factor of 1000. As usual, things are never this simple. While a GPU can crunch numbers and perform trigonometric operations as fast or faster than a CPU, it is significantly slower at performing logical operations and comparisons, such as those existing in typical flow control statements like loops and branches. Still, the speed gained by utilizing many cores on the GPU is expected to make up for this loss in looping efficiency.

A further gain in speed is realized by utilizing vector data types. A vector in OpenCL is similar to an array in C. The biggest difference between the two is that when an operation is performed on a vector in an OpenCL kernel, every element of the vector is operated upon at the same time. While it is not impossible for a CPU to perform vector operations, built in vector data types do not exist in C, C++, or FORTRAN; they must be defined by the user. The other significant difference between a C array and an OpenCL vector is that the OpenCL vector can only contain a specific number of bytes. The number of bytes that a vector can hold is dependent upon the device being used. Most GPU's will have a limit of 16 bytes per vector [5], which equates to four floating point scalars. This opens up the possibility of operating on four points at once per compute unit. We now have two strategies for parallelizing the Mandelbrot set calculation:

**Algorithm 1** Scalar Mandelbrot algorithm

```
get indices for pixel
transform indices to coordinates x and y
c ← x + iy
z ← 0 + 0i
for k = 1 to MaxIterations do
    z = z^2 + c
    if |z| ≥ 2 then
        break out of the loop
    end if
end for
store the value of k for pixel coloring
```

**Algorithm 2** Vector Mandelbrot algorithm

```
get indices for first pixel of group
transform indices to coordinates x_j and y_j for j = 0, 1, 2, 3
create an integer vector mask of length 4
create an integer vector iter of length 4 set to (0,0,0,0)
c_j ← x_j + iy_j
z_j ← 0 + 0i
for k = 1 to MaxIterations do
    z_j = z_j^2 + c
    if |z_j| ≤ 2 then
        mask_j = 1
    else
        mask_j = 0
    end if
    iter_j = iter_j + mask_j
    if mask = (0,0,0,0) then
        break out of the loop
    end if
end for
store the values of iter for pixel coloring
```

The parallelization of an IFS fractal is much more flexible by comparison. For instance, if the goal is to plot 100 points, one could run ten initial random points for ten iterations, or two initial random points for 50 iterations. In each case, the iterations are performed in parallel on the GPU. One minor drawback is that when starting with a random point, the point must undergo a certain number of unrecorded iterations to allow the system to settle and begin plotting points that are actually part of the fractal [6]. Typically this is taken to be 20 iterations; however if 1000 random points are started, this is 20,000 unrecorded iterations that are not required in the serial calculation. Different combinations of random starting points and numbers of iterations per starting point can be used to investigate the effect of the parallelization on the run time. The pseudo code for the generation of Seirpinski's triangle is as follows:

**Algorithm 3** Seirpinski algorithm

```
for each iteration thread:
get random starting point coordinates x and y
for k = 1 to 20 do
    s ← random number (0,1)
    if s < 1/3 then
        x = (1/2)x
        y = (1/2)y
    else if s < 2/3 then
        x = (1/2)x + 1/2
        y = (1/2)y
    else
        x = (1/2)x
        y = (1/2)y + 1/2
    end if
end for
for k = 1 to MaxIterationsPerThread do
    s ← random number (0,1)
    if s < 1/3 then
        x = (1/2)x
        y = (1/2)y
        transform x and y to image array indices
        plot point in image as color red
    else if s < 2/3 then
        x = (1/2)x + 1/2
        y = (1/2)y
        transform x and y to image array indices
        plot point in image as color green
    else
        x = (1/2)x
        y = (1/2)y + 1/2
        transform x and y to image array indices
        plot point in image as color blue
    end if
end for
```

When parallelizing the calculation of an IFS fractal, one important question to ask is where are the random numbers coming from? In order to make a fair comparison between the serial and parallel calculations, both codes should use the same random number generator (RNG); however there is no built in RNG in the OpenCL kernel language, so there are two options:

1. Generate the random numbers within each kernel as needed by writing a simple function.
2. Generate a buffer of random numbers from a proven RNG in the host C program and send them to the kernel for use.

The second option introduces a loss in speed due to the fact that memory transfer between the host program and the kernel is an added burden that the first option would not require. For this reason, the first option is chosen and a simple linear congruential generator (LCG) is implemented due to its high speed and low memory use [7]. A LCG is defined as a RNG that produces random numbers according to the following formula:

$$x_{n+1} = (ax_n + c) \bmod m \tag{5}$$

The values of $a$, $c$, and $m$ are chosen to be the same as those suggested by Press *et al* [8]. With a maximum period of $2^{32}$, this should work well enough for an IFS fractal with less than a billion plotted points.

## III. RESULTS

This is not the first study comparing a single CPU calculation to a single GPU calculation and it will certainly not be the last. When performing such a comparison, one question is typically debated: do you sacrifice efficiency in the serial algorithm for the sake of making the two codes perform nearly the same exact operations? For instance, in the serial calculation of the Mandelbrot set, one could simply adjust the coordinates inside the outer loops and write the integer iteration values to an output file for image processing, without storing these values in large, memory intensive arrays. For the parallel calculation however, the array is necessary because the GPU does not have the capability to write the values to file, and even if it did, many compute units would be trying to write to the same file at the same time.

For the parallelization of the Mandelbrot set, several cases are considered to perform the comparison as fairly as possible. Three separate serial strategies are considered alongside three separate parallel strategies. The three serial cases are:

1. Perform the calculation of the integer iteration values per pixel without the use of arrays. The output of the code is simply a text file that is plotted with gnuplot to check that the code is producing the correct output; however once the output is confirmed, the code is timed without the data being written to file, because writing this data to a file is time consuming and not relevant to the performance of the code.
2. Perform the calculation of the integer iteration values per pixel with the use of arrays. Other than this, the handling of the output is identical to the first serial case.

3. Perform the calculation of the integer iteration values per pixel with the use of arrays, and convert these integer values to RGB values. Once an array of RGB values is obtained, an output file is written in binary .bmp format, which is significantly faster than writing a text file, and eliminates the need for any external plotting capabilities.

The three parallel cases are:

1. Perform the calculation of the integer iteration values per pixel with the use of arrays, calculating the orbit of a single point per kernel. Once again, the output of the code is simply a text file plotted with gnuplot only to check that the code is producing the correct results. The code is then timed with no output being generated.

2. Perform the calculation of the integer iteration values per pixel with the use of arrays, calculating the orbit of a single point per kernel, and convert these integer values to RGB values. Once an array of RGB values is obtained, an output file is written in binary .bmp format.

3. Perform the calculation of the integer iteration values per pixel with the use of arrays, calculating the orbit of four points per kernel. Other than this, the handling of the output is identical to the first parallel case.

The use of these six cases should provide quite a fair comparison. A comparison between serial case 2 and parallel case 1 or serial case 3 and parallel case 2 shows the difference in speed between a single CPU and a single GPU performing the same exact operations, whereas a comparison between serial case 1 and parallel case 3 shows the difference in speed between the most economical methods in each category. The run time of each case given above is shown in Table I. For each case considered, the time reported in Table I is the real time required to calculate the Mandelbrot set of resolution 2048 × 2048 with a maximum of 3000 iterations, with or without output, depending on the case. All reported run times in this report are the averages of the run times for a sample of five identical runs.

TABLE I: Run times for the six Mandelbrot set cases considered.

| Case | Run time (s) |
|---|---|
| Serial 1 | 16.709 |
| Serial 2 | 16.829 |
| Serial 3 | 17.085 |
| Parallel 1 | 0.810 |
| Parallel 2 | 0.968 |
| Parallel 3 | 0.458 |

The parallelization of the IFS fractal provides much more flexibility. Unfortunately, the IFS fractal is not quite as easy to parallelize because of the added data that must be transferred from the host program to the kernel. For this reason, the speed up factor is expected to be less significant than for the complex number fractal which only needs an image buffer to write its values to. To parallelize the IFS fractal, a batch of random starting points and random seeds are produced, and each kernel code receives one element of each batch, along with the number of batches being run. The total number of points plotted is then divided by the number of batches, and the kernel performs the resulting number of iterations. For instance, if a total of 100,000,000 points are plotted with 100 batches, a set of 100 random starting points are generated along with 100 random number seeds, and each kernel performs 1,000,000 iterations from its own starting point. The ratio of the run times for the serial to the parallel calculations is given in Figure 5. All run times presented for the IFS fractal generation used a resolution of 2048 × 2048.

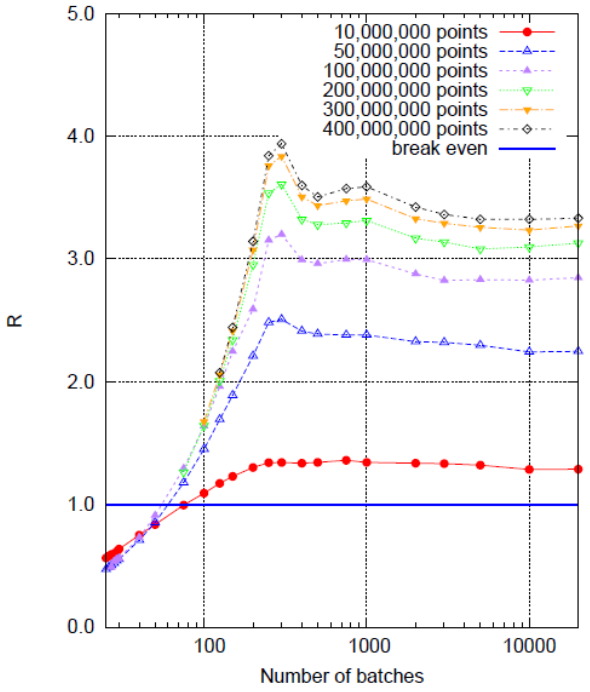



FIG. 5: Ratio of serial to parallel run times as a function of the number of batches.

Up to this point, all of the data presented has been generated using an AMD Radeon HD 6770M GPU. It would be interesting to compare these results to those generated using a different GPU. The second GPU chosen for this purpose is a ATI Mobility Radeon HD 5870. A comparison of the specifications for these two GPU's it given in Table II. As can be seen in Table II, The two GPU's are of comparable quality with the most significant difference being the number of processing units in each. Although the 5870 is from an older generation

of the same manufacturer, it is also from a higher class within that generation, and therefore still outperforms the 6770M.

TABLE II: Comparison of specifications for the two GPU's used. Values courtesy of game-debate.com [9].

| | ATI Mobility Radeon HD 5870 | AMD Radeon HD 6770M |
|---|---|---|
| Processing units | 800 | 480 |
| Memory | 1024 MB | 1024 MB |
| Memory speed | 1000 MHz | 800 MHz |
| Clock speed | 700 MHz | 725 MHz |

A comparison of the performance of each of these GPU's can be seen in Figure 6 where the ratios of serial to parallel run times are plotted as a function of the number of batches used in the IFS fractal generation. In this figure, the number of plotted points is 400,000,000. The vertical lines in the figure are the number of processing units for each GPU, and seem to line up with the local minimum of each set of data.

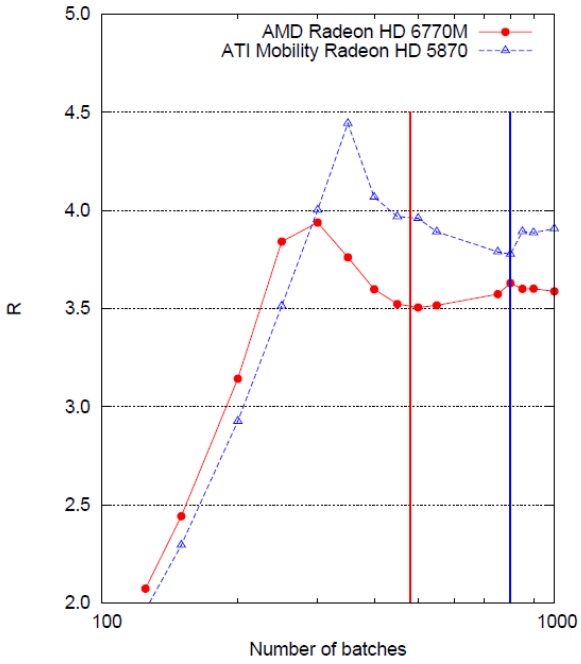



FIG. 6: Run time ratio comparison for the two GPU's considered.

## IV. SUMMARY AND CONCLUSIONS

When deciding how to parallelize an algorithm, the programmer must identify whether the algorithm is best suited for data parallelism or task parallelism. In the case of data parallel algorithms, such as the calculation and generation of fractals, the repetitive calculations should be performed on a GPU if possible. This is supported by the overall speed up factor of 36.48 for the Mandelbrot set and 4.44 for Seirpinski's triangle. In addition, these speed up factors were found for GPU's that are no where near top of the line by today's standards. It is not uncommon for a modern GPU to have over 2,000 processing units and significantly more memory. For instance, the top of the line GPU from AMD to date is the AMD Radeon HD 8970 which has 2048 processing units along with 3 GB of memory and a 1 GHz clock speed [10].

When examining Figure 6, it may seem strange that the run time decreases slightly as the number of batches approaches the number of processing elements on the GPU. One would naively think that if all of the processing units are in use, the GPU is being used to its fullest extent, and hence most efficiently. However, since both GPU's have the same memory and comparable memory speeds, the dip is most likely due to the exorbitant amount of data being handled by the GPU when all cores are being used. More batches means longer starting point arrays and longer random number seed arrays for the GPU to store in its global memory.

While IFS fractals allow the programmer to experiment more with the parallelization of the algorithm, it is clear that complex number fractals are likely to have much more significant speed up factors when using the GPU. This is because they require less memory transfer between the host program and the GPU, but also because the loops performed on the GPU can be significantly shorter than those needed for the IFS fractal. Of course, this statement will not always be true, depending upon the length scale of the complex number fractal and the number of points plotted in the IFS fractal; however in many cases complex number fractals can be generated accurately using a maximum number of iterations in the thousands, whereas the IFS fractal may require a million or even a billion points to be plotted depending upon the resolution of the image.

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

[1] B. B. Mandelbrot, *The Fractal Geometry of Nature* (W. H. Freeman and Company, San Francisco, 1982).

[2] P. W. Addison, *Fractals and Chaos: An illustrated course* (Institute of Physics Publishing, Bristol, 1997).

[3] G. W. Flake, *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation* (MIT Press, Cambridge, 1998).

[4] B. Barney, *Introduction to Parallel Computing* (2013), (computing.llnl.gov).

[5] M. Scarpino, *OpenCL in Action* (Manning Publications, Greenwich, 2011).

[6] M. Semeniuk, M. Znoj, N. Mejia, and S. Robertson, *Accelerated rendering of fractal flames* (2011), (eecs.ucf.edu).

[7] H. G. Katzgraber, *Week 05: Numerical integration and Random numbers* (2014), (PHYS-401 Lecture Slides).

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran 77; The Art of Scientific Computing 2nd Ed.* (Cambridge University Press, Cambridge, 1992).

[9] Game Debate, *PC Hardware Game Requirements* (2014), URL `http://www.game-debate.com/hardware/`.

[10] AMD, *AMD Radeon HD 8000 Series Graphics (OEM)* (2014), URL `http://www.amd.com/us/products/desktop/graphics/8000/`.

# How does the minimum depend on the different parameters

Wang Jizhou
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: Apri 17, 2014)

In this work, I study the finding the minimum of function $f(x) = 11\sin(6x) + 7\cos(5x)$ in range of $(0, 2*\pi)$ by using genetic algorithms. Previous studies at class we learned several ways to find the minimum number of a function. The advantage of using genetic algorithms than other optimizations is that it can easily find the minimum number in the whole range rather than tracked in part of the range.

## I. INTRODUCTION

In the computer science field of artificial intelligence, genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search problems.[1] Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution. In nature, a group contains amount of individuals which have their own gene. The individuals which contain the gene can suit the environment better will have more chance to survive and create next generation. When the better individuals give the good gene to their babies, mutation and crossover will happens. In this program, we will simulate what happens in nature and using this way to do optimization.

## II. MODEL

In general, if we want to using GA to do optimization, we should follow several steps. Firstly, make sure what you are going to optimize and abstract a mathematics model; Secondly, change the parameters in the model into chromosomal which contains all the information of one individual's gene; Thirdly, create the first generation of a group; Fourthly, evaluate new generations with selection, crossover and mutation for around 50 generations; Finally, get the answer and make a check, and adjust your code to see if can make it better. In this time, I will use GA trying to find the minimum number of function 1 in range of(0, 2Pi).

$$f(x) = 11\sin(6x) + 7\cos(5x) \tag{1}$$

I tried to us Mathematica to plot this function, and it is showed by the picture. And find minimum value of the function related to x equals to 1.8486. We can use this solution to check if GA can optimize it well.

For the fist step, it is already a easy model which computer can deal with.

For the second step, I separate the range of $2\pi$ to $2^D$ parts and using binary numbers like from 0 000 000 000 000 000 to 1 111 111 111 111 111 to represent of them. So that they can be write in the form of chromosomal.

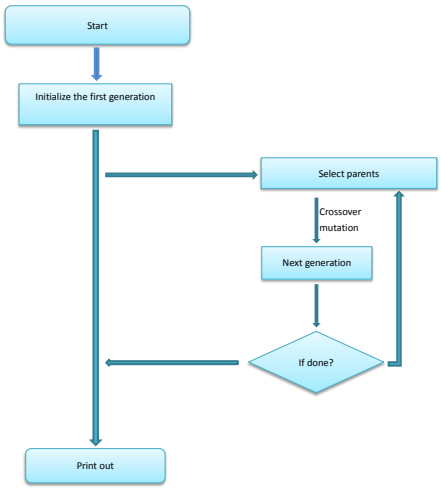



FIG. 1: GA

For the third step, I create a three-dimensional array x[T][M][D], which T is the number of generations, M(to be convenient of the rest calculation, I choose M is even) is the number of the individuals. When T=0, it present the first generation, and for each individual, I using random number generator to create random number of 0 and 1, and then put them into first generation a[0][M][16] one by one. Then the first generation was created.

For the forth step, the most important part is how to select.

1.Firstly we need to find out how to determine which individual is better than others, and the way to do it is

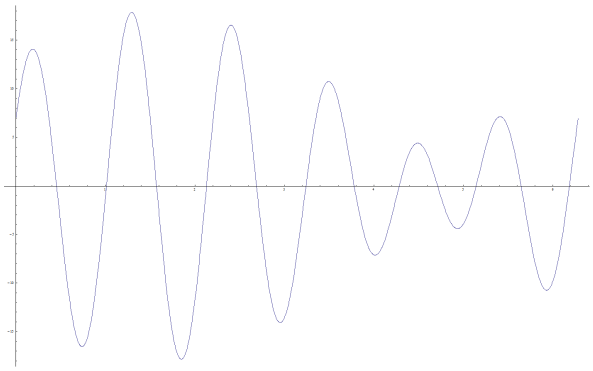


FIG. 2: original function

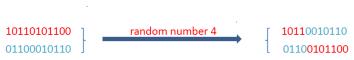



FIG. 3: example for crossover

to found a fitness function to measure each individuals, just to make sure that the bigger the answer you get from the fitness function, the better the individual is. In my case, firstly I change the 16 digits 'chromosomal' to the x in 1 by using 2,

$$x = \sum_{i=0}^{D-1} x_i \cdot 2^i \tag{2}$$

then I set fit function as 3.

$$fit(x) = 18 - 11\sin(6x) - \cos(5x) \tag{3}$$

2. Principal the better individuals are, the more chance they would be parents to generate next generation. Using a good way to select parents can save your time to get the answer you want. Usually, the the probability to be chosen is proportional to the value of fitness function like 4.

$$Pi = fit(x_i) / \sum fit(x_i) \tag{4}$$

Since this problem I am trying to solve is easy, I choose a simple way to deal with this: every time before creating a new generation, I using bubble sorb to sorb the old generation according the answer of fitness function, then I give up some of the worst individuals and then pick the couples randomly in the rest with equal probability, and at end I copy the top two individuals' chromosomal in the old generation directly (this way is Hybrid Genetic Algorithm which can accelerate to close to the optimality).

3.for the crossover part, it is easy to achieve. After picking a couple, get a random number in range of 1 to 16 to determine the point to cross. it showed in picture 1 I randomly get two number, and cross over the chromosomal between them, and I thought this way can be better which was wrong.

4. After cross over to generate a new generation, you still need to make mutation on them. To do this, you need to set a probability of mutation. Here is my code and I set a function for mutation called mut(i, a[][][D]).

For the last step, I try to do some adjust some parameter and some change of the code to see how the progress will changed. In first time I running the code, I set number of individuals as 1000, D=16 and p=0.06, and I print out the best individual of each generate. The program is solved immediately, and it gets the exact answer. So GA do can solve this problem well and with little time.

Let us try to do further research of GA, so we can understand it better.

1. How to choose the parameter D? I think it at least depends on two things.The first thing you should consider is the form of the fitness function. If the fitness function is "sensitive" to the change of x which means if you change x a little, the value of the function changes a lot, you need to set a large D. It is easy to understand. Because the number you find will have a little difference with the exact value, and from 5 (A is the range of x)

$$error_{\max} = \frac{A}{2^D} \tag{5}$$

the error is in range of [0,*error*], if the function is sensitive in the area near the minimum values part, this small error can cause a big difference in related function value. So if this things happen, you need to make D lager to reduce the error; Another thing determines the value of D is how many significant digits you want. Like in this problem I solve, when D equals 16, from 5 the maximum

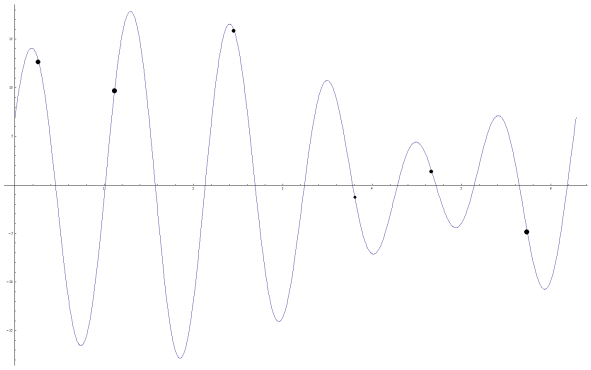


FIG. 4: 2

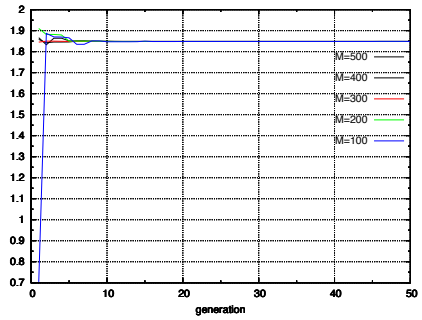



FIG. 5: I set different M, and plot the best individual's value change with the number of generation

error is 0.0055, so the significant digits is 4. If you need more significant digits, then you need to set D bigger until the maximum error is smaller than the digits you need. But for some problems, D is already setted, like traveling salesman problems, D should equal to the numbers of cities. 2.How to choose a proper parameter M? These fact we can conclude that the number of individuals you choose can't be very small, otherwise it can't find the right answer; And also it not necessary to be very big, if so it will need to run a long time when solving a hard problem. So choose a proper M is important. And what should you consider when you are trying to set M? I think there are at least two things you need to consider. One is the function's form, in fact in general, should be the fitness function's form, if it has a lot of partial optimized points, then you need plenty of individuals, on the contrary, if your function don't have many partial points optimized points, you don't need to set a lot of individuals. I think this is not hard to understand, like if you choose small group of the first generation, sometimes they would separate like this, and it will hard for them to approximate to the minimum number. So seems the more peaks the function have, the more individuals are needed. I run this program several times with M as 1000, D as 16 and p=00.6, and every time it can get the same answer and the final value show up in around 16 generation. It seems the number of individuals may be too much for this problem. So I reduce the number of individuals from 1000 to 500, and it still can find the exact answer in few generations. Then I set M as 100, then sometimes I even can't get the exact solution but a number close to the real answer. Later, I tried 30, and the final answer trapped in 0.7335 which the related value of function 1 is not the minimum number in the whole range but a partial minimum number.

Is there have a chance that I set a small M, but make a big number of generations? Then I set M equals 20, and run it for 500 generations, and I got a plot. The answer in

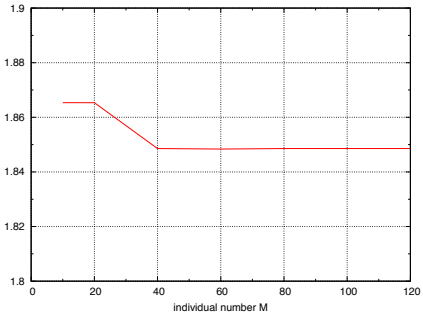



FIG. 6: Change M and plot the change of the best individual in 50th generation

500 generation is same with it in 100th genenration, but to make the begining visible, I only plot 100 generations. And we can see that it is trapped and hard to go to the right value in big number genenrations. The other thing you should consider is how many parts you divide the range to, in other wordz, it depends on the D you set, if D get larger, then the number of individuals also should be more. To prove this, I change D to 25 and set M as 1000, then I find that it always can't find the right answer in 50 generations. 3.How to choose the number of generations T? In fact, it is releated to parameter M, If you set a small M, then you need to evolve many generations to get the final answer. Of course it can't be too small, otherwise it asks for a huge number of individuals, which is not neccessary. But we also can't set it too big and set M small, cause then it will be hard for you to distinguish

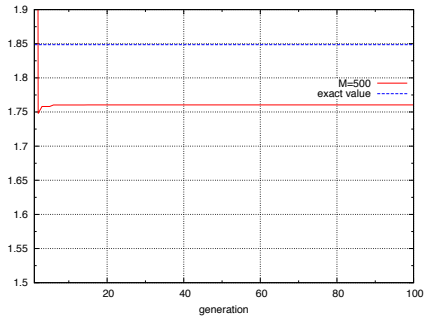



FIG. 7: set M=20, and make more generations

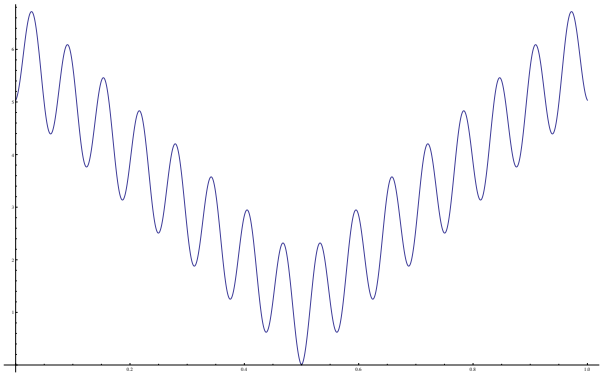


FIG. 8: New function

if you get a right answer, like the example above, I set M equals to 20 and evolve for 500 generations, and I get the same value for over 400 generations, you might think it is the final answer, but the fact is it is wrong. So usually we set T around 50. But notice that some problems do need a big number of T, and it can't be solved in few generation, like traveling salesman problem.

To prove what I find above are right, I will use them to predict another finding minimum problem. If the prediction are right with the fact, then it means what I find is right. I will try to find the minimum of function 6 in range of (0,1). And the minimum number is x equals 0.5.

$$f(x) = 10|x - 0.5| - \cos(100(x - 0.5)) + 1 \qquad (6)$$

I know the plot of the fuction is and the number of peaks is almost same to the front one. First, I can still set D as 16; Second, the range of new function is almost 6 times smaller than the front one, but I set the same D, so I predict that the number of individuals M can be several times smaller than it in front one, like set M equals 20 can still get the right answer And I make a plot of this;

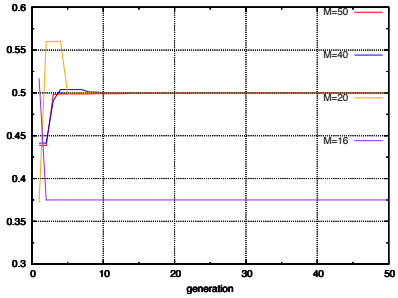



FIG. 9: Change M of new function

Third, I set T as 50. The plot are exact same with my prediction. So my views above seems are right.

## III. SUMMARY AND CONCLUSIONS

Genetic Algorithms do can solve optimizations, and can solve some problems well. It has some advantages than other algorithms. Like we can make it parallel to save the running time; also, it can find peak without knowing the derivative of the function, so it can solve some optimize problem in life not only in mathematics field.

But we can see there are some limit for GA, like it related to the number of parameters which can't be found exactly, also, if using GA to solve problems have difficult fitness functions like have a lot of partial peaks, then you need to set a big number of individuals which can make the program run slow, like Traveling Sales Man problem. And sometimes it will be difficult to make a chromosomal or abstract a fitness function.

So usually when we use GA to solve some difficult problems, we usually add some algorithms to GA or combine them together to speed it up. And people already developed a lot of algorithms based on GA so that it can solve some special problems quickly like Hybrid Genetic Algorithms.

# Variational Quantum Monte Carlo

Zhikun Xing, Yunsong Wu
*Department of Physics and Astronomy, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 26, 2014)

We used the Variational Quantum Monte Carlo methods to find the ground state energy of several systems, including the 1D harmonic oscillator, 3D harmonic oscillator, hydrogen atom, helium atom, hydrogen molecule, and trihydrogen cation. The values we got in our experiments agree with the theoretical ones very well.

## I. INTRODUCTION

### A. The variational theorem

Quantum mechanical systems of particles can be described by the time-dependent Schrödinger equation

$$i\hbar\frac{\partial}{\partial t}\Psi = \hat{H}\Psi \tag{1}$$

For stationary states where the Hamiltonian is not dependent on time, we have the time-independent Schrödinger equation

$$\hat{H}\Psi = E\Psi \tag{2}$$

where the Hamiltonian is

$$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{x}) \tag{3}$$

Exact solutions have been found only for a very small number of problems which can essentially be reduced to one-dimensional ordinary differential equations. Good examples include the 1-D and 3-D quantum harmonic oscillator, as well as the hydrogen atom which consists of a proton and an electron interacting through a Coulomb force. For most quantum systems which involve more than two particles, i.e., atomic nuclei and electrons, numerical methods must be used.

For a sytem, the eigenfuctions which can describe the system are $\psi_n(\mathbf{x})$. If they are complete, then any wavefunction $\Psi(\mathbf{x})$ can be expressed as a linear superposition of these eigenfunctions

$$\Psi(\mathbf{x}) = \sum_n c_n\psi_n(\mathbf{x}) \tag{4}$$

where $c_n$ are any complex numbers. The energy corresponding to this wavefunction is

$$E = \frac{\int \Psi^*(\mathbf{x})\hat{H}\Psi(\mathbf{x})d\mathbf{x}}{\int \Psi^*(\mathbf{x})\Psi(\mathbf{x})d\mathbf{x}} \tag{5}$$

According to the variational theorem, $E \geqslant E_{ground}$, where $E_{ground}$ is the ground state energy of the system. The equal sign "=" holds if and only if $\Psi(\mathbf{x}) = c_0\psi_0$, where $\psi_0$ is the eigenfunction of the ground state.

Given the variational theorem, in order to compute the ground state energy of a sytem, we can guess a general wavefunction which contains some parameters. Then, we calculate the energy corresponding to that wave function, and the energy is also a function of the parameters in the wavefunction. Next, we can vary the parameters to minimize the energy, which will give us a good upper bound of the exact ground state energy.

### B. Variational Monte Carlo (VMC)

In the Variational Monte Carlo method, we choose a wave function $\Psi(\mathbf{x})$ depending a set of parameters, $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$. We calculate the energy according to

$$E = \frac{\int \Psi^*(\mathbf{x})\hat{H}\Psi(\mathbf{x})d\mathbf{x}}{\int \Psi^*(\mathbf{x})\Psi(\mathbf{x})d\mathbf{x}}$$

using the Hamiltonian of the system, $\hat{H}$. In order to compute the multi-dimensional integrals in this formula efficiently, Monte Carlo methods will be used. We will use the Metropolis algorithm to sample the important regions of the multi-dimensional space. In the VMC method, the distribution of our sample points is

$$\rho(\mathbf{x}) = \frac{|\Psi(\mathbf{x})|^2}{\int \Psi^*(\mathbf{x})\Psi(\mathbf{x})d\mathbf{x}} \tag{6}$$

Then, the energy is

$$\begin{aligned} E &= \frac{\int \Psi^*(\mathbf{x})\hat{H}\Psi(\mathbf{x})d\mathbf{x}}{\int \Psi^*(\mathbf{x})\Psi(\mathbf{x})d\mathbf{x}} = \frac{\int |\Psi(\mathbf{x})|^2 \frac{\hat{H}\Psi(\mathbf{x})}{\Psi(\mathbf{x})}d\mathbf{x}}{\int \Psi^*(\mathbf{x})\Psi(\mathbf{x})d\mathbf{x}} \\ &\equiv \int d\mathbf{x}\rho(\mathbf{x})E_L(\mathbf{x}) \end{aligned} \tag{7}$$

where

$$E_L(\mathbf{x}) = \frac{\hat{H}\Psi(\mathbf{x})}{\Psi(\mathbf{x})} \tag{8}$$

is called "local energy". The wavefunction $\Psi(\mathbf{x})$ can generally be chosen to be real and positive definite when evaluating the ground state energy.

The VMC strategy is to choose a set of random points, $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$, in the configuration space acoording to the distribution function $\rho(\mathbf{x})$, and evaluate $E_L$. The estimated ground state energy will be

$$E = \frac{1}{M}\sum_i^M E_L(\mathbf{x}) \tag{9}$$

## II. MODEL

### A. VMC for the Harmonic Oscillator

We consider the 1-D harmonic oscillator first, as it is the simplest quantum mechanical system of particles which has bound states. The Hamiltonian operator for a 1-D harmonic oscillator is

$$\hat{H} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{1}{2}m\omega^2 x^2 \tag{10}$$

Choosing units such that $m = 1$, $\hbar = 1$, and $\omega = 1$, the Hamiltonian in theses units is

$$\hat{H} = -\frac{1}{2}\frac{d^2}{dx^2} + \frac{1}{2}x^2 \tag{11}$$

Using a simple variational trial function $\Psi(x) = e^{-\alpha x^2}$, the local energy is calculated as

$$E_L(x) = \alpha + x^2(\frac{1}{2} - 2\alpha^2) \tag{12}$$

The distribution function $\rho(x) \sim e^{-2\alpha x^2}$. It can be seen that when $\alpha = 0.5$, the exact ground state energy is reached.

As FIG. 1 shows, after executing the program, we find that when $\alpha = 0.5$, we arrive at the minimal energy, 0.5, with the minimal error, 0. This is the same as the exact result, which means the trial function we choose is exactly the eigenfunction of the ground state.

Then, we consider the 3-D harmonic oscillator. The Hamiltonian is

$$\hat{H} = -\frac{d^2}{dx^2} + \frac{1}{2}x^2 - \frac{d^2}{dy^2} + \frac{1}{2}y^2 - \frac{d^2}{dz^2} + \frac{1}{2}z^2 \tag{13}$$

Using the trial function as a product of the wave functions of the three coordinates, $\Psi(\mathbf{x}) = e^{-\alpha(x^2+y^2+z^2)}$, the local energy is

$$E_L(x) = 3\alpha + r^2(\frac{1}{2} - 2\alpha^2) \tag{14}$$

where $r = \sqrt{x^2+y^2+z^2}$. The distribution function $\rho(\mathbf{x}) \sim e^{-2\alpha r^2}$. It can be seen that when $\alpha = 0.5$, the exact ground state energy is reached.

As FIG. 2 shows, after executing the program, we find that when $\alpha = 0.5$, we arrive at the minimal energy, 1.5, with the minimal error, 0. This is the same as the exact result.

### B. VMC for the Hydrogen Atom

The Hamiltonian operator for the hydrogen atom is

$$H = -\frac{\hbar^2}{2m}\nabla^2 - \frac{ke^2}{r} \tag{15}$$

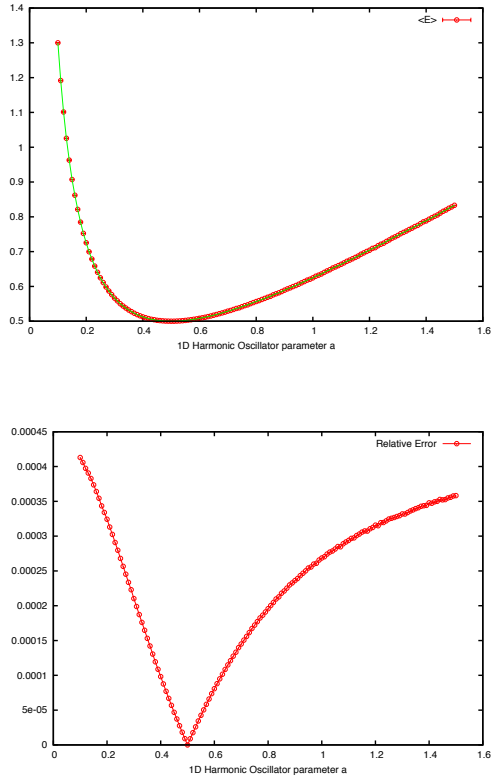



FIG. 1: E and Eerror as a function of $\alpha$ for the 1-D harmonic oscillator

Using atomic units, namely, Bohr radius

$$a_0 = \frac{\hbar^2}{mke^2} = 0.529\mathring{A}$$

Ground Energy of two independent hydrongen atom, which is also known as one *hartree*

$$E = \frac{ke^2}{a_0} = 27.2eV$$

to rewrite the Hamiltonian

$$H = -\frac{1}{2}\nabla^2 - \frac{1}{r} \tag{16}$$

As the hydrogen atom is spherically symmetric, if we write $H$ using spherical coordinates, only the radial part of $H$ will remain:

$$H = -\frac{1}{2}\left(\frac{d^2}{dr^2} + \frac{2}{r}\frac{d}{dr}\right) - \frac{1}{r} \tag{17}$$

Using the trial function is $\Psi(x) = e^{-\alpha r}$ , the local energy is

$$E_L(x) = -\frac{1}{2}\left(\alpha^2 - \frac{2\alpha}{r}\right) - \frac{1}{r} \tag{18}$$

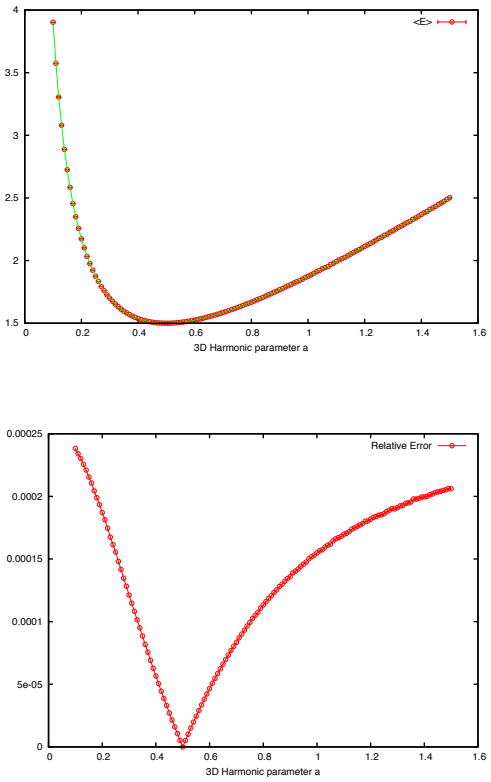



FIG. 2: E and Eerror as a function of $\alpha$ for the 3-D harmonic oscillator

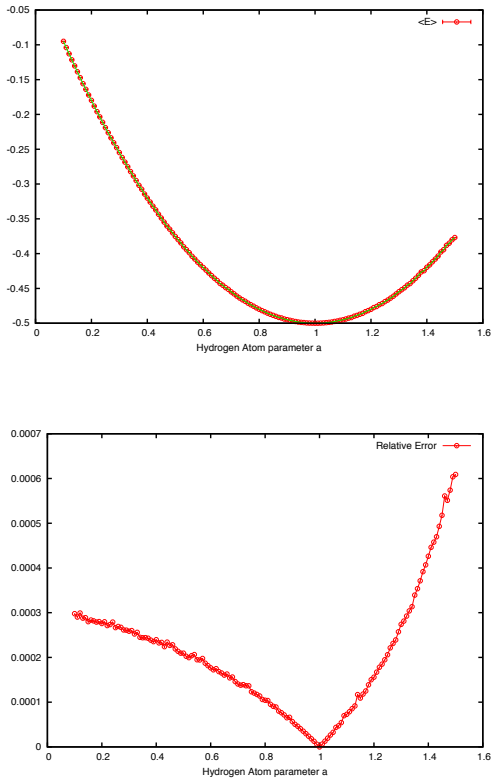



FIG. 3: E and Eerror as a function of $\alpha$ for the hydrogen atom

And the distribution function $\rho(x) \sim e^{-2\alpha r^2}$. It can be calculated that when $\alpha = 1$, the exact ground state energy is reached.

As FIG. 3 shows, after executing the program, we find that when $\alpha = 1$, we arrive at the minimal energy, $-0.5$, with the minimal error, 0. This is the same as the exact result.

### C. VMC for Helium Atom

#### 1. *Hamiltonian and Trial Wave Function*

The Hamiltonian operator for the helium atom is

$$H = -\frac{\hbar^2}{2m}(\bigtriangledown_1^2 + \bigtriangledown_2^2) - \frac{2ke^2}{|\overrightarrow{r_1} - \overrightarrow{R}|} - \frac{2ke^2}{|\overrightarrow{r_2} - \overrightarrow{R}|} + \frac{ke^2}{|\overrightarrow{r_1} - \overrightarrow{r_2}|} \tag{19}$$

Assuming the nucleus is at the origin, and use atomic units to write Hamiltonian

$$H = -\frac{1}{2}(\nabla_1^2 + \nabla_2^2) - \frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}} \tag{20}$$

where $r_1 = |\overrightarrow{r_1}|$, $r_2 = |\overrightarrow{r_2}|$, and $r_{12} = |\overrightarrow{r_1} - \overrightarrow{r_2}|$.

To construct the trial wave function, we use the Pade-Jastrow wave function

$$\Psi(\overrightarrow{r_1}, \overrightarrow{r_2}) = \phi(r_1)\phi(r_2)\psi(\overrightarrow{r_1}, \overrightarrow{r_2}) \tag{21}$$

where $\phi(r_1)$ and $\phi(r_2)$ are the wavefunctions of the electron in a hydrogen-like atom,

$$\phi(r_1) \sim e^{-2r_1}, \ \phi(r_2) \sim e^{-2r_2}$$

And

$$\psi(\overrightarrow{r_1}, \overrightarrow{r_2}) = \exp\left[\frac{r_{12}}{2(1 + \alpha r_{12})}\right] \tag{22}$$

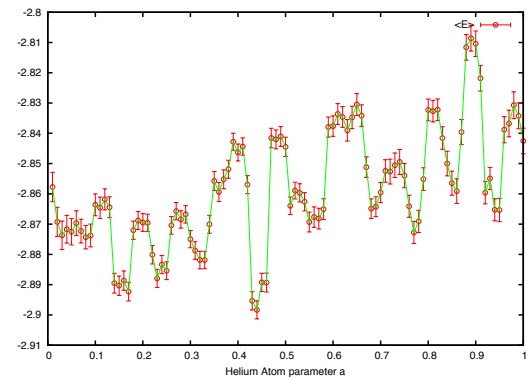
incorporates the interacion of two electrons. Here, $\alpha$ is the variational parameter.

Local energy is calculated as

$$E_L(\overrightarrow{r_1},\overrightarrow{r_2}) = -4+\frac{\alpha}{1+\alpha r_{12}}+\frac{\alpha}{(1+\alpha r_{12})^2}+\frac{\alpha}{(1+\alpha r_{12})^3} -\frac{1}{4(1+\alpha r_{12})^4}+\frac{\hat{r_{12}}\cdot(\hat{r_1}-\hat{r_2})}{(1+\alpha r_{12})^2} \quad (23)$$

### 2. *Result Analysis*

The experimental value is $-2.903$[1]. In our computation, when $\alpha = 0.44$, the energy achieves its minimum, -2.898, with Eerror 0.003, as shown in FIG. 4.



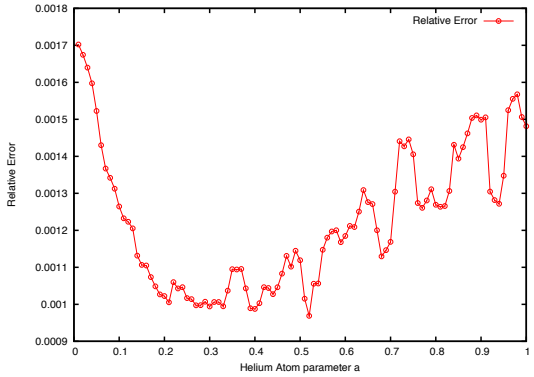
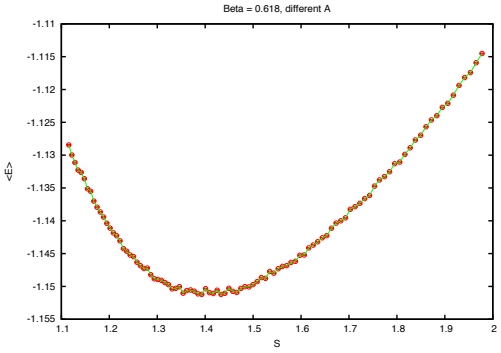




FIG. 4: E and Eerror as a function of $\alpha$ for the helium atom

Our experimental value is close to the theoretical one, with deviation

$$\frac{|(-2.898)-(-2.903)|}{|-2.903|} = 0.2\%$$



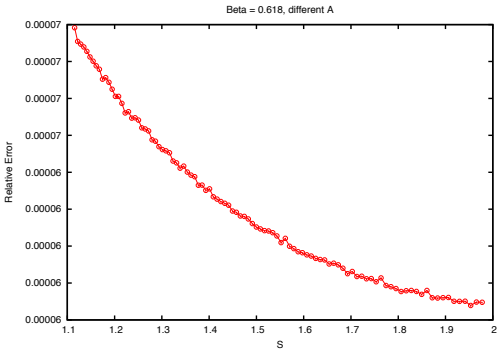




FIG. 5: Set $\beta = 0.618$, $E$ and $\sigma(E)$ with difference $s$

## D. VMC for Hydrogen Molecule

### 1. *Hamiltonian and Trial Wave Function*

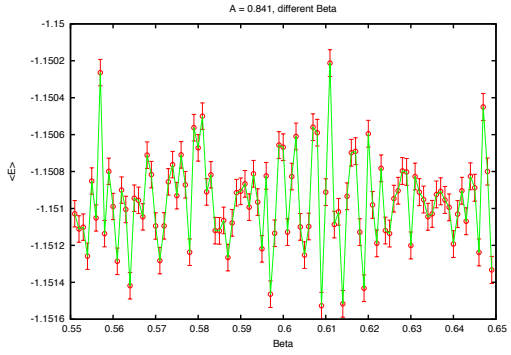
Born-Openhiemer approximation is taken to assume that the nuclear motion is negligible, and the two nuclei are symmetrically located on the X axis at $(\pm s/2, 0, 0)$. The Hydrogen molecule consisits of two protons and two electrons, thus there are three parts that contribute to the Hamiltonian.

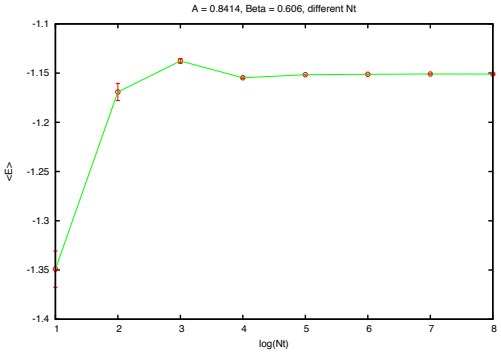
Two kinetic energy items for elctron motion;

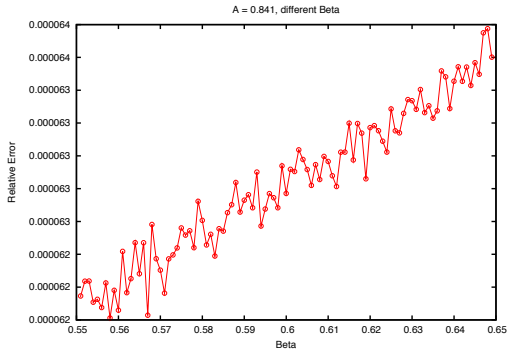
Four electron-proton items of attrative electronsatic potential;

Two electron-electron, proton-proton items of attrative electronsatic potential.

$$H = -\frac{\hbar^2}{2m}(\bigtriangledown_1^2+\bigtriangledown_2^2)-[\frac{ke^2}{|\overrightarrow{r_1}-\overrightarrow{R_1}|}+\frac{ke^2}{|\overrightarrow{r_1}-\overrightarrow{R_2}|} +\frac{ke^2}{|\overrightarrow{r_2}-\overrightarrow{R_1}|}+\frac{ke^2}{|\overrightarrow{r_2}-\overrightarrow{R_2}|}] +\frac{ke^2}{|\overrightarrow{r_1}-\overrightarrow{r_2}|}+\frac{ke^2}{|\overrightarrow{R_1}-\overrightarrow{R_2}|} \quad (24)$$



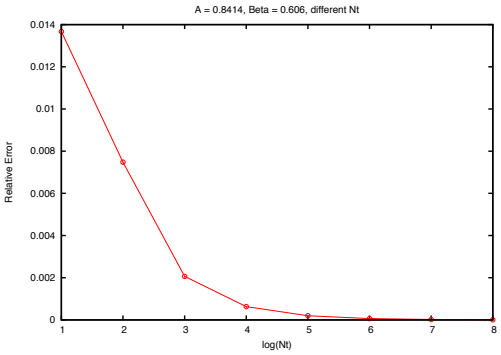




FIG. 6: Set $a = 0.841$, $E$ and $\sigma(E)$ with difference $\beta$

FIG. 7: $E$ and $\sigma(E)$ with optimized $a = 0.8414$, $\beta = 0.606$

Use atomic units to write Hamiltonian

$$H = -\frac{1}{2}(\nabla_1^2 + \nabla_2^2) - [\frac{1}{r_{1L}} + \frac{1}{r_{1R}} + \frac{1}{r_{2L}} + \frac{1}{r_{2R}}] + \frac{1}{r_{12}} + \frac{1}{R_{12}} \quad (25)$$

where

$$r_{1L} = |\overrightarrow{r_1} - \overrightarrow{R_1}| \quad r_{1R} = |\overrightarrow{r_1} - \overrightarrow{R_2}|$$
$$r_{2L} = |\overrightarrow{r_2} - \overrightarrow{R_1}| \quad r_{2R} = |\overrightarrow{r_2} - \overrightarrow{R_2}|$$
$$r_{12} = |\overrightarrow{r_1} - \overrightarrow{r_2}| \quad R_{12} = |\overrightarrow{R_1} - \overrightarrow{R_2}|$$

We use a general form of multiparticles wavefunction

$$\Psi(\overrightarrow{r_1}, \overrightarrow{r_2}) = \phi(r_1)\phi(r_2)\psi(\overrightarrow{r_1}, \overrightarrow{r_2}) \quad (26)$$

It is constructed by combining wave function of Hydrogen Atom

$$\phi(r_1) = \phi_{1L} + \phi_{1R} = e^{-r_{1L}/a} + e^{-r_{1R}/a}$$
$$\phi(r_2) = \phi_{2L} + \phi_{2R} = e^{-r_{2L}/a} + e^{-r_{2R}/a}$$

and the Jastrow function

$$\psi(\overrightarrow{r_1}, \overrightarrow{r_2}) = \exp\left[\frac{r_{12}}{\alpha(1+\beta r_{12})}\right]$$

There are four parameters in the Hamiltonian and wave function, $a$, $\alpha$, $\beta$, and the distance of two protons $s$. $a$ describes the electron-proton item as single hydrogen atom, $\alpha$ and $\beta$ stand for the electron-electron interaction, and $s$ is the distance of two protons. Two of them can be expressed by others after applying the Coulomb cusp conditions

$$a(1 + e^{-s/a}) = 1, \alpha = 2 \quad (27)$$

The conditions above are model-specific, and can remove singularities in local energy. After applying the conditions, the local energy is now

$$\begin{aligned}\epsilon = & \frac{1}{s} - \frac{1}{a^2} + \frac{1}{a\phi_1}\left(\frac{\phi_{1L}}{r_{1L}} + \frac{\phi_{1R}}{r_{1R}}\right) + \frac{1}{a\phi_2}\left(\frac{\phi_{2L}}{r_{2L}} + \frac{\phi_{2R}}{r_{2R}}\right) \\ & - \left[\frac{1}{r_{1L}} + \frac{1}{r_{1R}} + \frac{1}{r_{2L}} + \frac{1}{r_{2R}}\right] + \frac{1}{r_{12}} \\ & - \frac{4(1+\beta r_{12}) + r_{12}}{4(1+\beta r_{12})^4 r_{12}} + (\frac{\phi_{1L}\hat{r}_{1L} + \phi_{1R}\hat{r}_{1R}}{\phi_1} \\ & - \frac{\phi_{2L}\hat{r}_{2L} + \phi_{2R}\hat{r}_{2R}}{\phi_2}) \cdot \frac{\hat{r}_{12}}{2a(1+\beta r_{12})^2}\end{aligned} \quad (28)$$

Now we can use VMC method to optimize the parameters and wave function.

### 2. Result Analysis

A good result can be found in the reference[2]. The energy of Hydrogen molecules is $-1.1640239 \pm 9 \times 10^{-7}$ *hartrees*, the distance of two nuclei is about 1.40 $a_0$.

The result of our code is close to this region, with $E_0 \sim -1.151017 \pm 0.000007$, $s \sim 1.404028$, for total $10^8$ Monte Carlo sweeps. The deviation of ground state energy is

$$\frac{|(-1.151017) - (1.1640239)|}{|1.1640239|} = 1\%$$

and the deviation of the bonding distance is

$$\frac{|1.404 - 1.4011|}{|1.40|} = 0.8\%$$

Our calculation give a very good approximation of the theoretical values.

The following figures show the energy and standard deviation of the molecule while searching one parameter with the other one fixed.

As FIG. 5 and FIG. 6 show, it's clear that the $E$ and $\sigma(E)$ are sensitive to proton distance $s$, which can be explained by the large ratio of proton-proton energy.

However, the parameter $\beta$ in Jastrow exponential factor has an implicit effect on the results, suggesting that the behavior of electrons should be carefully tuned to approach more precise results.

Better results can be reached by increasing Monte Carlo iteration times and using smaller parameter steps, which is shown in FIG. 7. The error can be reduced to $10^{-4}$ by performing $10^6$ iterations.

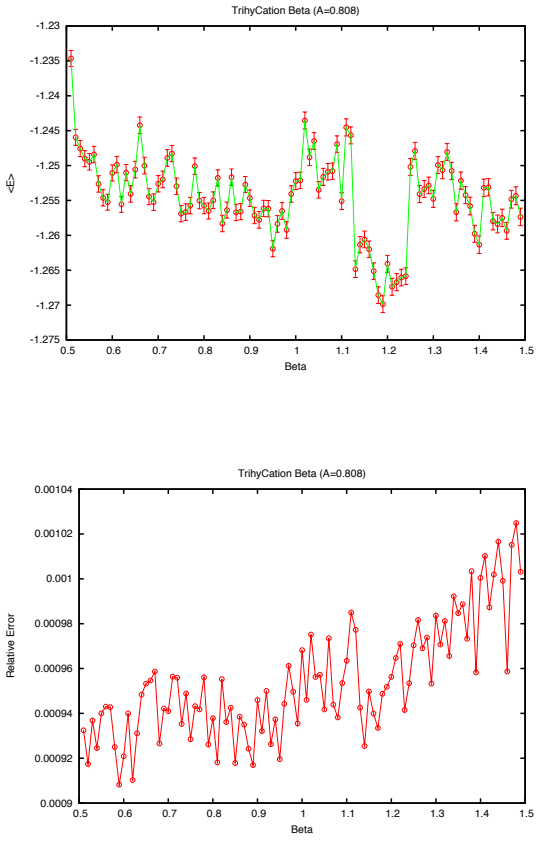



FIG. 8: Set $a = 0.808$, $E$ and $\sigma(E)$ with difference $\beta$

## E. VMC for Trihydrogen Cation ($H_3^+$)

### 1. Hamiltonian and Trial Wave Function

Again, we assume that the nuclear motion is negligible. According to the literature, the arrangement of the hydrogen atoms in the molecule is an equilateral triangle[3]. Thus, we assume that the three nuclei are located at $(-\frac{s}{2}, 0, 0)$, $(\frac{s}{2}, 0, 0)$, and $(0, \frac{\sqrt{3}}{2}s, 0)$.

The Hamiltonian for the trihydrogen cation in atomic units is

$$\begin{aligned} H &= -\frac{1}{2}(\bigtriangledown_1^2 + \bigtriangledown_2^2) - [\frac{1}{r_{1L}} + \frac{1}{r_{1R}} + \frac{1}{r_{1U}} \\ &\quad + \frac{1}{r_{2L}} + \frac{1}{r_{2R}} + \frac{1}{r_{2U}}] + \frac{1}{r_{12}} + \frac{3}{R_{12}} \end{aligned} \tag{29}$$

We use the trial function

$$\Psi(\overrightarrow{r_1}, \overrightarrow{r_2}) = \phi(r_1)\phi(r_2)\psi(\overrightarrow{r_1}, \overrightarrow{r_2}) \tag{30}$$

where

$$\begin{aligned} \phi(r_1) &= \phi_{1L} + \phi_{1R} + \phi_{1U} \\ &= e^{-r_{1L}/a} + e^{-r_{1R}/a} + e^{-r_{1U}/a} \end{aligned}$$

$$\begin{aligned} \phi(r_2) &= \phi_{2L} + \phi_{2R} + \phi_{2U} \\ &= e^{-r_{2L}/a} + e^{-r_{2R}/a} + e^{-r_{2U}/a} \end{aligned}$$

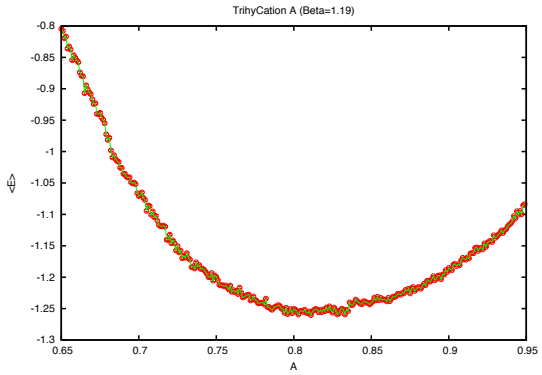
and the Jastrow function

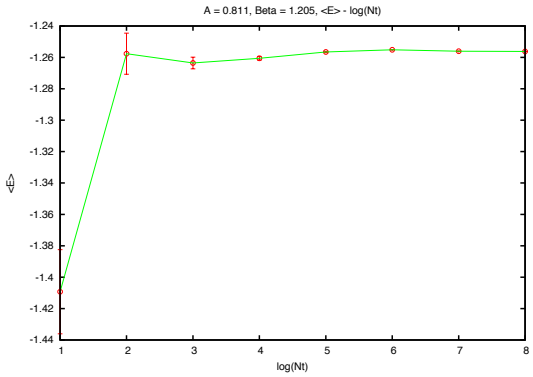
$$\psi(\overrightarrow{r_1}, \overrightarrow{r_2}) = \exp\left[\frac{r_{12}}{\alpha(1 + \beta r_{12})}\right]$$

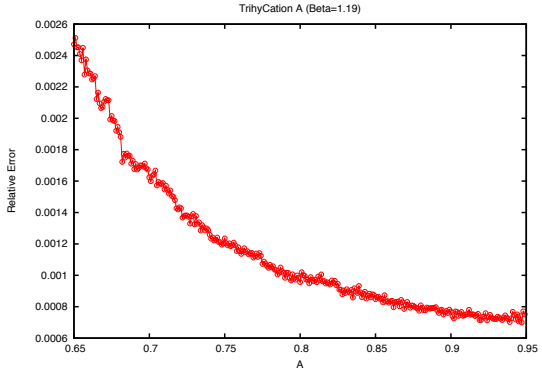
The Coulomb cusp conditions for this model are derived as

$$a(1 + 2e^{-s/a}) = 1, \ \alpha = 2 \tag{31}$$

Note that there is a "2" factor here in the first condition compared to that of the hydrogen molecule. Local energy



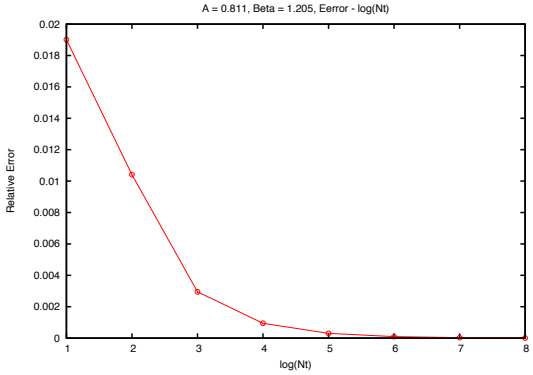




FIG. 9: Set $\beta = 1.19$, $E$ and $\sigma(E)$ with different $a$

FIG. 10: $E$ and $\sigma(E)$ with optimized $a = 0.811$, $\beta = 1.19$

becomes

$$\begin{aligned}\epsilon \;=\; & \frac{3}{s} - \frac{1}{a^2} + \frac{1}{a\phi_1}\left(\frac{\phi_{1L}}{r_{1L}} + \frac{\phi_{1R}}{r_{1R}} + \frac{\phi_{1U}}{r_{1U}}\right) + \frac{1}{a\phi_2}\Big(\frac{\phi_{2L}}{r_{2L}} \\ & + \frac{\phi_{2R}}{r_{2R}} + \frac{\phi_{2U}}{r_{2U}}\Big) - \Big[\frac{1}{r_{1L}} + \frac{1}{r_{1R}} + \frac{1}{r_{1U}} + \frac{1}{r_{2L}} + \frac{1}{r_{2R}} \\ & + \frac{1}{r_{2U}}\Big] + \frac{1}{r_{12}} - \frac{4(1+\beta r_{12}) + r_{12}}{4(1+\beta r_{12})^4 r_{12}} \\ & + \Big(\frac{\phi_{1L}\hat{r}_{1L} + \phi_{1R}\hat{r}_{1R} + \phi_{1U}\hat{r}_{1U}}{\phi_1} \\ & - \frac{\phi_{2L}\hat{r}_{2L} + \phi_{2R}\hat{r}_{2R} + \phi_{2U}\hat{r}_{2U}}{\phi_2}\Big) \\ & \cdot \frac{\hat{r}_{12}}{2a(1+\beta r_{12})^2} \end{aligned} \tag{32}$$

### 2. *Result Analysis*

The theoretical value of the ground state energy is -33.42 eV (-1.228676 *hartrees*), with the nucleus-nucleus distance 0.9 $\mathring{A}$ (1.701323 $a_0$)[3].

The result of our code is close to this region, with $E_0 \sim -1.25618 \pm 0.00001$, $s \sim 1.743$, for total $10^8$ Monte Carlo sweeps. The deviation of ground state energy is

$$\frac{|(-1.25618) - (-1.228676)|}{|-1.228676|} = 2\%$$

and the deviation of the bonding distance is

$$\frac{|1.743 - 1.701323|}{|1.701323|} = 2\%$$

Our calculation give a very good approximation of the theoretical values.

The following figures show the energy and standard deviation of the molecule while searching one parameter with the other one fixed.

As FIG. 8 and FIG. 9 show, we get the similar results compared to those of hydrogen molecule. $E$ and $\sigma(E)$ are sensitive to $a$, which corresponds to the nuclear-nuclear distance $s$. The parameter $\beta$ has an implicit effect on the

results, suggesting that the behavior of electrons should be carefully tuned to approach more precise results.

Better results can be reached by increasing Monte Carlo iteration times and using smaller parameter steps, which is shown in FIG. 10. The error can be reduced to $10^{-4}$ by performing $10^6$ iterations.

## III. SUMMARY AND CONCLUSIONS

The values of the minimal energy of several systems such as the 1D harmonic oscillator, 3D harmonic oscillator, hydrogen atom, helium atom, hydrogen molecule, and trihydrogen cation, agree with the theoretical ones very well.

However, error exists for our model, as seen in every system above. In order to reduce the error and improve the precision of the results, considering the possible source of the error, we propose three ways:

1. Use better trial wavefunction. In our model, for a multiparticle system, we use a Jastrow trial wavefunction. There may be some other wavefunction which agrees with the real one better.

2. Search parameters more precisely. In our model, the precision of the parameters is $10^{-3}$, which may be not small enough.

3. Average over more Monte Carlo sweeps. The error is proportional to $1/\sqrt{N}$, where $N$ is the number of Monte Carlo sweeps. More sweeps, smaller error. $N = 10^6$ in our case, which may be not big enough.

Despite of the error, our results are close to the theoretical ones remarkably.

## IV. REFERENCES

[1] http://en.wikipedia.org/wiki/Helium_atom

[2] L. Wolniewicz. Nonadiabatic energies of the ground state of the hydrogen molecule. J. Chem. Phys. 103, 1792 (1995).

[3] http://en.wikipedia.org/wiki/Trihydrogen_cation

### Acknowledgments

We thank the NSF (Grant No. DMR-1151387) for financial support and Texas A&M University for access to their Eos cluster.

# APPENDIX

## A. VMC Flowchart

The implementation of VMC alogrithm can be instructed by the following flowchart.

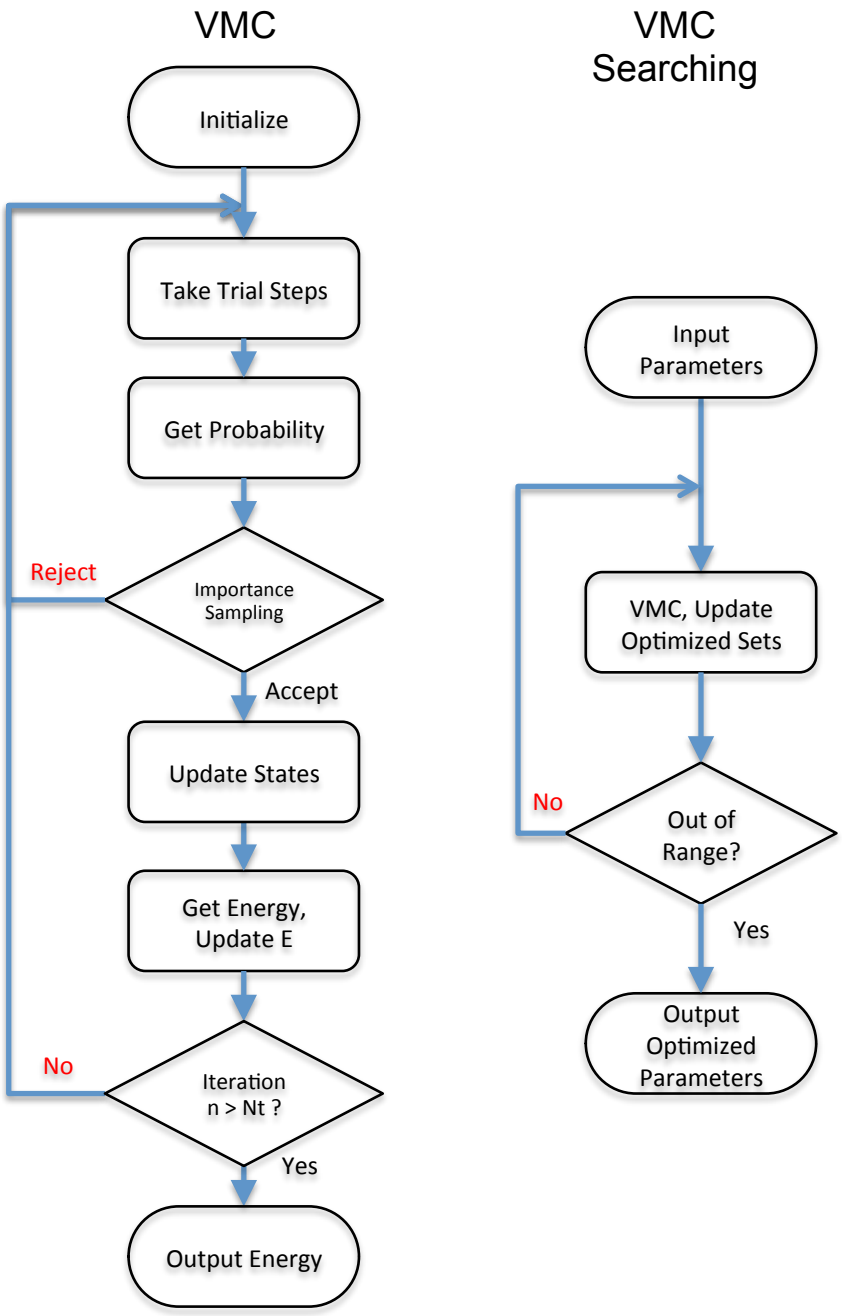



FIG. 1: Flowchart of the MC main program

### B. VMC Main Codes

The main code of VMC in C programming language.

```
/* Important Global Virables */
int M = 1;     // M Makov chains
double T = 1.0;    // Teperature
int Nt = 1000;    // Nt Monte Carlo sweeps
int Nskip = 1000;   // Sweep Nskip steps to get thermal equibilium

#define Nw 10    // Nw independent random walker
#define N 2     // N particles in a single system
#define RDIM 3     // Dimension of the position in a single system
#define Np 3     // Numbers of parameters

double Parameters[Np];  // Np parameters for trial wavefunction
double R[Nw][N][RDIM];  // Store the configuration of Nw independent walkers
double Eavg = 0.0;   // Average energy <E> of a single system
double E2avg = 0.0;   // Average energy squared <E^2> of a single system
double Eerror = 0.0;  // Energy error Delta(E) of a single system
double Step = 1.0;   // Step of position in random walk that affects accept ratio

/* VMC Main Function */
void VMC(FILE *File)
{
 int m, t;
 double sum, sumE, sumE2;     // Sum of E, E^2 over each Markov chain
 int Nadjust = ceil(0.1 * Nskip);   // Interval to adjust the step
 double AR;         // Accept ratio of each Sweep
 int Naccept = 0;       // Counting Accept Ratio
 Eavg = E2avg = 0.0;       // Initialize external viables

// Sweep for M Markov chains
 for(m = 0; m < M; m++) {
  // Initialize r1279() random number generator
  long seed = seedgen();
  setr1279(seed);
  // Initialize R to equal probability random states (cooling)
  InitState();

  // Sweep Nskip steps to get thermal equibilium
  Naccept = 0;
  for(t = 1; t <= Nskip; t++) {
   // MCSweep & get accept ratio
   Naccept += MCSweep();
   // Adjust the step (every Nadjust steps) to get accept ratio ~ 0.5
   if(t % Nadjust == 0) {
    AR = 1.0 * Naccept/ (Nw * Nadjust);
    Step *= AR/ 0.5;
    // reset counter
    Naccept = 0;
   }
  }

  Naccept = 0;
  // Initialize sumE, sumE2 to zero
  sum = 0.0;
  sumE = 0.0;
```

```
  sumE2 = 0.0;
  // Sweep & measure for Nt steps
  for(t = 1; t <= Nt; t++)
  {
   Naccept += MCSweep();
   // Average over Nw systems
   sum = GetEnergy()/ Nw;
   // Sum up over Nt steps
   sumE += sum;
   sumE2 += sum * sum;
  }

  // Sum up over each Chain
  Eavg += sumE/ Nt;
  E2avg += sumE2/ Nt;
  // Monitor accept ratio
  AR = 1.0 * Naccept/ (Nw * Nt);
 }

// Get <E>, <E^2>, Error of a single system over M Markov chains
 Eavg /= M;
 E2avg /= M;
 Eerror = sqrt(E2avg - Eavg * Eavg)/ sqrt(Nt - 1.0);

// Output parameters & results to a file
 for(t = 0; t < Np; t++) {
  fprintf(File, "%.10lf ", Parameters[t]);
 }
 fprintf(File, "%.10lf %.10lf %.10lf %lf\n", Eavg, E2avg, Eerror, AR);

return;
}

/* Perform one MC Sweep, return number of accepts */
int MCSweep()
{
 int i;
 int pos;    // Index [0, Nw-1] of current single system R[pos][N][RDIM]
 State Trial;   // Create Trial[N][RDIM] to store the state of trial single system
 int Naccept = 0;  // Count number of accepts
 double prob;   // Probability of transition

// Update the chain for Nw walkers
 for(i = 0; i < Nw; i++) {
  // Pick a single system randomly, and assign pointer Pos to it
  pos = ir1279() % Nw;
  // Get trial state from current state and certain distribution
  GetTrialPos(R[pos], Trial);
  // Get probability from Pos to Trial
  prob = GetProbability(R[pos], Trial);
  // Perform importance sampling
  if(prob > r1279()) {
   // Update state Pos to Trial
   UpdateState(R[pos], Trial);
   Naccept++;
  }
 }
```

```
// Return Naccept to count accept ratio
 return Naccept;
}
```

# 2D Ising Model with Simple MC and Cluster Algorithm

JiaNing Zhang
*Department of Atmospheric Sciences, Texas A&M University, College Station, Texas 77843-4242, USA*
(Dated: April 26, 2014)

Simulations of the two-dimensional Ising model were performed on a square lattice using Metropolis and Wolff cluster algorithms. Near critical point, Metropolis algorithm showed strong "Critical Slowing Down" phenomena while cluster algorithm concurred this defect quite well. Autocorrelation functions and Finite Size Scaling analysis were conducted in various sizes of systems in detail.

## I. INTRODUCTION

The Ising model, named after the physicist Ernst Ising, is a mathematical model of magnetism in statistical physics. In Ising model, spins are represented by discrete variables (pointing up: +1 or pointing down: -1) . The interactions between spins, arranged in a lattice, is limited to nearest neighbors. And the two-dimensional square-lattice Ising model is one of the simplest statistical models to show a phase transition.

On the other hand, Enrico Fermi, John Pasta, and Stanslaw Ulam created the first "computer experiment" to study a vibrating atomic lattice using Monte Carlo methods. The Monte Carlo methods, named after the casino in Monaco, use randomly generated numbers to solve computationally intensive problems when other techniques unfortunately fail. However, the widespread usage of Monte Carlo methods did not begin until Metropolis algorithm was invented by Metropolis, Rosenbluth and Teller. The Metropolis algorithm works quite well in simulating two-dimensional Ising model except the cases close to Curie temperature. In these cases, relaxation times diverge when approach the Curie temperature (critical slowing down). Hence, more efficient algorithms like cluster algorithm are needed to cure this problem.

In this report, we implement simulations of the two-dimensional Ising model on a square lattice using Metropolis and Wolff cluster algorithms. Critical slowing down phenomena, autocorrelation times as well as the finite size scaling were analyzed in details.

## II. MODEL AND ALGORITHM

### A. Ising Model

The Hamiltonian of the Ising model on a scale-free graph in an external magnetic field is given by

$$\mathcal{H} = -J \sum_{<i,j>} \sigma_i \sigma_j - h \sum_i \sigma_i, \qquad (1)$$

where are spins on a square lattice and sum is over the four nearest neighbor bonds. It is conventional to set the coupling strength $J = 1$ and Boltzmann's constant $k = 1$, which amounts to measuring energies and temperatures in units of $J$. The constant h is called the external field, and $\mathcal{M} = \sum_i \sigma_i$ is called the magnetization.

### B. Metropolis Algorithm

The Metropolis algorithm was named after Nicholas Metropolis, who along with Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller, first proposed it for calculating the states from the canonical ensemble. It can be shown that this popular method satisfies the detailed balance condition. Metropolis Algorithm:

(1) Pick a spin randomly;

(2) Compute the energy difference $\Delta$E for flipping it;

(3) If $\Delta E < 0$ flip it; if $\Delta E > 0$ flip it with the probability $e^{-\beta \Delta E}$

### C. Cluster Algorithm

Near the Cure temperature $T_c$, the single-spin-filp dynamics becomes quite slow known as the correlation time diverges. Wolff improved on the idea of Swendsen and Wang, camping up with a brilliant algorithm to flip the spin cluster each time. Wolff Cluster Algorithm:

(1) Pick a spin randomly, record its direction, then flip it.

(2) For each of the four neighboring spins, if its direction is the same with the host, flip it with a probability p.

(3) For each new flipped spins, repeat the procedure (2).

Due to the finite probability for spin flipping, the Wolff algorithm is ergodic and Markovian. It also satisfy the detailed balance.

## III. SIMULATIONS AND RESULTS

### A. Thermalization

When we perform simulations of Ising model, the initial configuration is quite crucial. Since it usually determine how long we need to attain the equilibrium states. When starting at low temperature, it makes sense to begin with an ordered configuration in which all spins have

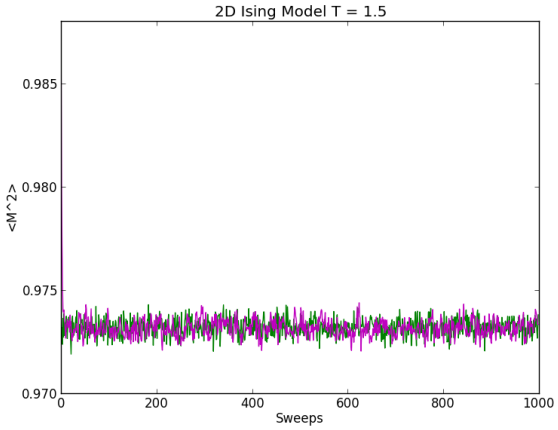



FIG. 1: $< \mathcal{M}^2 >$ calculated using Wolff and Metropolis algorithms are shown as functions of time. Time is represented by the number of spin sweeps. The calculations have been performed with a $100 \times 100$ lattice at temperature $T = 1.5$. Both stat with the ordered configuration.

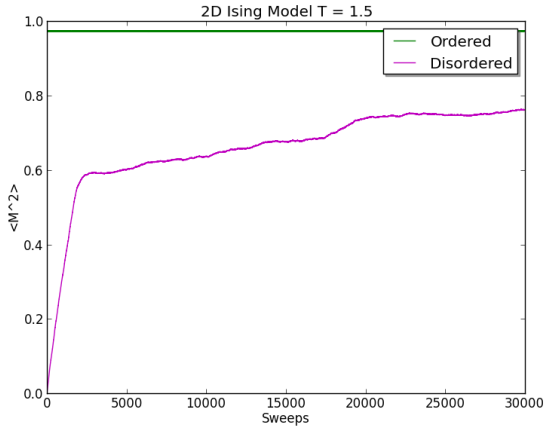



FIG. 2: $< \mathcal{M}^2 >$ is shown as functions of time. Time is represented by the number of spin sweeps. The calculations have been performed with a $100 \times 100$ lattice at temperature $T = 1.5$. Two initial configurations were used, on with a random orientation of spins and the other with an ordered configuration.

the same direction, whereas for a high temperature, it would be better to start at a configuration with randomly assigned spins.(Fig. 1) It would be difficult to guess a suitable initial configuration at a general given $T$. Sometimes, we may need much more Monte Carlo sweeps before the system reaches the most probable states if we unfortunately made a poor estimate. (Fig. 2) The number of sweeps to reach a equilibrium configuration is known as the thermalization. (Fig. 3)

### B. Critical Slowing Down

At the critical temperature $T_c$, some observables become divergent in the thermodynamic limit. It is believed that long range correlations between spins result in this critical divergence. When the system is approaching $T_c$, the spins are constantly changing dependently. Large clusters of the same spin direction persist, so spins far apart from each other are strongly correlated. On the other hand, dynamical observables are functions of time. The "measurements" at each time step construct a time series. Near $T_c$, the relaxation time of these time series become very large and also prove to be shown diverge for infinite system. Such phenomena is known as "Critical Slowing Down". (Fig. 4)

### C. Correlated Measurements and Autocorrelation Times

The variance of the data in equilibrium can be obtained

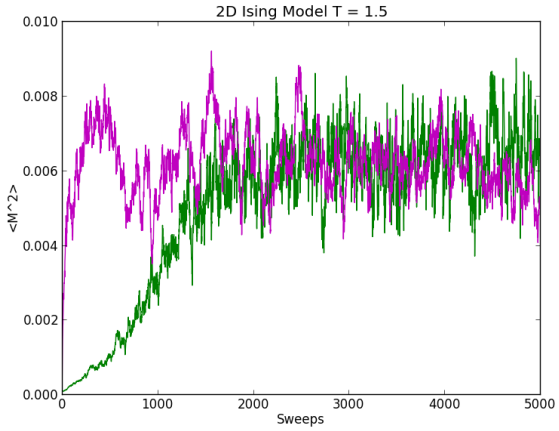



FIG. 3: $< \mathcal{M}^2 >$ calculated using Wolff and Metropolis algorithms are shown as functions of time. Time is represented by the number of spin sweeps. The calculations have been performed with a $100 \times 100$ lattice at temperature $T = 2.5$. Both stat with the random distributed configuration.

$$\sigma^2_{\mathcal{O}^-} = \frac{1}{N}[\sigma^2_{\mathcal{O}_i} + 2\sum_{k=1}^{N}(< \mathcal{O}_1\mathcal{O}_{1+k} > - < \mathcal{O}_1 >< \mathcal{O}_{1+k} >)(1 - \frac{k}{N})], \tag{2}$$

where, due to the last factor $(1 - k/N)$, the $k = N$ term may be trivially kept in the summation. And we introduce the autocorrelation time

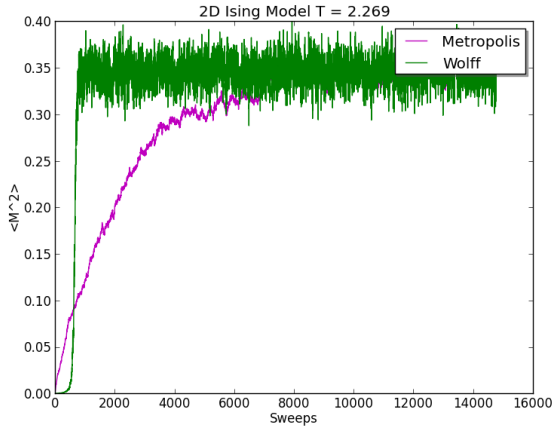



FIG. 4: $< \mathcal{M}^2 >$ calculated using Wolff and Metropolis algorithms are shown as functions of time. Time is represented by the number of spin sweeps. The calculations have been performed with a $100 \times 100$ lattice at temperature $T_c = 2.269$. Both stat with the random distributed configuration. It illustrates critical slowing down at the beginning sweeps.

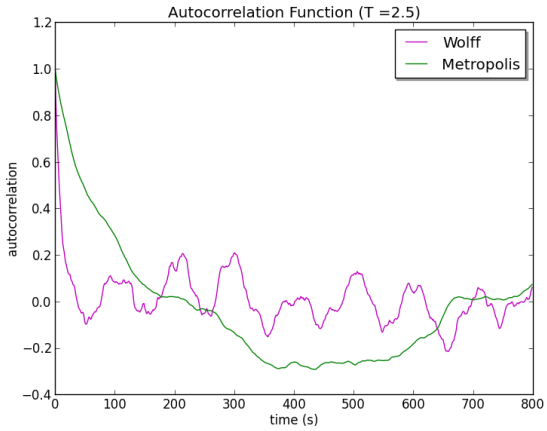



FIG. 5: Autocorrelation function calculated using Metropolis and Wolff algorithms. The time data of magnetization were collected from $t = 20000$ to $t = 21600$.

$$\tau'_{\mathcal{O},\mathrm{int}} = \frac{1}{2} + \sum_{k=1}^{N} A(k)(1 - \frac{k}{N}), \tag{3}$$

The normalized autocorrelation function $\phi(t)$(Fig. 5) for overvables is defined as

$$A(k) = \frac{< \mathcal{O}_1 \mathcal{O}_{1+k} > - < \mathcal{O}_1 >< \mathcal{O}_{1+k} >}{\sigma^2_{\mathcal{O}_i}} \tag{4}$$

### D. Finite Size Scaling

To analyze the systems, we make measurement and collect data. Standard quantities are the energy and magnetization, but depending on the model at hand it may be useful to record also other observables. In this way the full dynamical information can be extracted still after the actual simulation runs and error estimation can be easily performed. For example it is no problem to experiment with the size and number of Jackknife bins. Since a reasonable choice depends on the a priori unknown autocorrelation time, it is quite cumbersome to do a reliable error analysis on the flight during the simulation. Furthermore, basing data reweighting on time-series data is more efficient since histograms, if needed or more convenient, can still be produced from this data but working in the reverse direction is obviously impossible.

For our model, it is sufficient to perform a single long run at some coupling T close to the critical point $T_c$. Hence, we use Binder parameters to withdraw the information we need, since in our model

The Binder parameters

$$g(\mathrm{T}) = 1 - \frac{< \mathrm{m}^4 >}{3 < \mathrm{m}^2 >^2}, \tag{5}$$

In the infinite limit, most of these quantities exhibit singularities at transition point. If we use a scaling variable, these singularities are smeared out

$$x = (\mathrm{T} - \mathrm{T_c})\mathrm{L}^{1/\nu}, \tag{6}$$

Finite Size Scaling Procedure:

(1) Estimate the critical exponent $\nu$. In our model , we let $\nu = 1$.

(2) Extract the $T_c$ from the crossings from the Binder parameter.

(3) Re-analyze the Finite Size Scaling behavior of observables as a function of the scaling variable x.

## IV. SUMMARY AND CONCLUSIONS

We have performed a Monte Carlo simulations for two-dimensional Ising model with Metropolis algorithm and Wolff cluster algorithm. Critical slowing down are shown. The autocorrelation function are computed for both algorithms. And we also conduct the Finite Size Scaling analysis.

### Acknowledgments

We thank Dr. Katzgraber for his wonderful lectures on computation physics and patience in the lab. We also thank Mr. Mcdonald for the guidance in the lab and Texas A&M University for access to their Eos cluster.

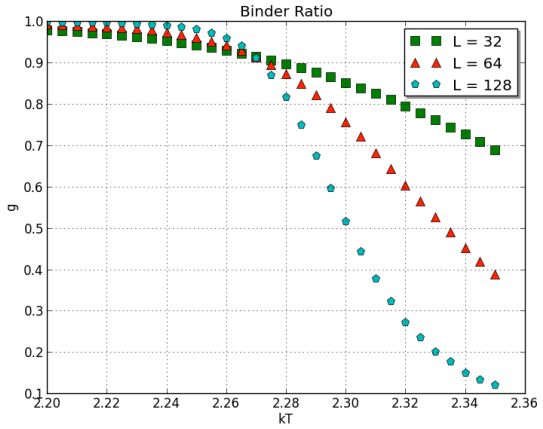



FIG. 6: Binder Ratio calculated using Wolff cluster algorithm. We use 3 different sizes L = 32, L = 64, L = 128. For each size, we measured every $\Delta$T = 0.005 from T = 2.20 to T = 2.35. And for each particular temperature, we collected 50 samples.

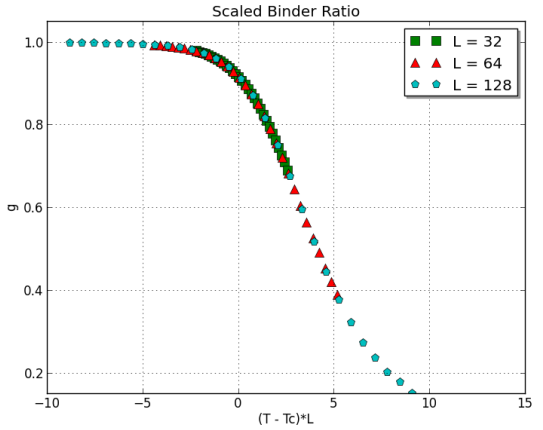



FIG. 7: Scaled Binder Ratio calculated using Wolff cluster algorithm. We use 3 different sizes L = 32, L = 64, L = 128. For each size, we measured every $\Delta$T = 0.005 from T = 2.20 to T = 2.35. And for each particular temperature, we collected 50 samples.

[1] Kerson Huang, 1987, *Statistical Mechanics(2nd edition)* (Wiley).
[2] Kurt Binder, and Dieter W. Heermann, 2010, *Monte Carlo Simulations in Statistical Physics(Fifth Edition)* (Springer).
[3] Holger Fehske, Ralf Schneider,and Alexander weibe, 2008, *Computational Many-Particle Physics* (Springer).