Prifysgol Abertawe
Swansea University

University of Swansea

Masters Dissertation

BL1: 2D Potts Model with a Twist

Author:
Robert James

Supervisor:
Professor Biagio Lucini

May 19, 2015

Abstract

The goals of this project were to perform a Monte Carlo simulation of the Potts Model, to numerically generate the Density of States of a periodic and twisted lattice and investigate the behaviour of the interfaces that occur when a twist is introduced to the system, for a variety of Q states. These goals have been accomplished using a variety of techniques. Including using both the Metropolis update algorithm and the Wang-Landau restricted sampling technique in a custom lattice solver written in C++.

Contents

Chapter 1

Introduction

This project studies the behaviour of interfaces that occur when a twist is added to a discrete state lattice. The Potts Model is used in this project as it provides a variable number of discrete states, the Hamiltonian can be written in the form

$$H = -J \sum_{(i,j)} \delta\left(s_i, s_j\right) (1.1)$$

In this project we ignore the introduction of an external magnetic field so the Hamiltonian remains in this form.

Chapter 2

Theory

2.1 Thermodynamic Limit

In the thermodynamic limit,

$$N \to \infty, V \to \infty, \frac{N}{V} = const. \tag{2.1}$$

As stated earlier in computer simulations the concept of ∞ is completely redundant. Computers retain a restricted amount of memory available to perform calculations. To this end, simulations in general study the behaviour of smaller systems. Depending on the complexity of the problems invovled and how co-related each site is, you often find a large range in sizes from $8x8$ up to several thousand. Assuming momentarilly you could perform computer simulations for an inifinite lattice, you would find results that after reasonable thermalisation and statistical analysis that are identical to a theoretical prediction. As stated earlier, the number of lattice sites you can reasonablly use for simulation purposes is restricted. This introduces boundary effects and noise that are not present on an inifite grid. To reduce this problem you turn to your lattice construction.

2.2 Lattice

As stated earlier, it is impossible to perform computer simulations for inifite lattice sizes. By shrinking the lattices down to useable sizes you typically introduce boundary effects that can add a layer of noise to the data. To reduce the effects of boundary noise, you can enforce periodic boundary conditions on your lattice such that

$$\begin{aligned} s_{i=L,j=0} &= s_{i=0,j=0} \\ s_{i=0,j=L} &= s_{i=0,j=0} \end{aligned} \tag{2.2}$$

In a physical sense taking a 2 dimensional lattice and enforcing such boundary conditions upon it leads to a torus as shown below.

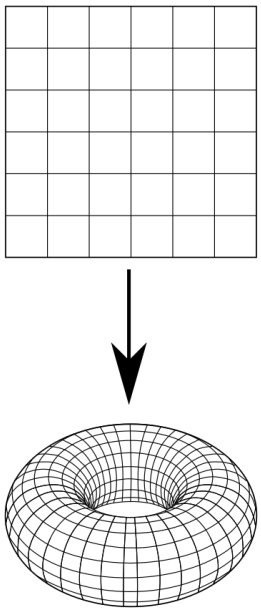


Figure 2.1: 2 dimensional lattice transforming to torus under periodic boundary conditions

2.3 Interfaces

2.4 Observables

2.4.1 Magnetisation

Materials are often classified by their responses to externally applied fields. The types of response to the applied fields are diamagnetic, paramagnetic or ferromagnetic. Diamagnetism is the weakest response and opposes the applied magnetic field. Paramagnetism is stronger than diamagnetism and is proportional to the field strength and aligned to the direction. Ferromagnetism, is the type of magnetism studied in this project, the effects of this type of response can occasionally be orders of magntitude larger than of the other types. In this project, we have no externally applied field. Diamagnetism and Paramagnetism in this case do not

apply because they are in response to an externally applied field.

$$M = \sum_{i}^{V} \sigma_i \tag{2.3}$$

2.4.2 Energy

$$H = E = -\sum_{i,j} \delta(\sigma_i, \sigma_j) \tag{2.4}$$

2.4.3 Specific Heat

$$C = \beta^2[< E^2 > - < E >^2] \tag{2.5}$$

2.4.4 Magnetic Susceptibility

$$\chi = \beta[< M^2 > - < M >^2] \tag{2.6}$$

2.5 Partition Function

2.5.1 Calculating the DoS

2.6 Fixing the Partition Functions

2.7 Free Energy

Chapter 3

Code

3.1 Design Choices

In this project, I started designing the code such that it would compile into a single executable. To ensure that I didn't need to recompile this code every time I needed to change parameters, I opted for a simple configuration file that would determine the functions and the flow of operations. To this end, I ended up using the C++11 standard[1] and a library designed to parse configuration files libconfig [4].

libconfig has two different interfaces, a C API (Application Programming Interface) and a C++ API. I picked the API written in C++ because it matched the rest of the project. An example configuration file is given below.

```
#Program Mode
wanglandau = true;
coldstart = true;
dim_grid = 16;
dim_q = 2;
interface = true;
#Metropolis Algorithm Parameters
beta = 1.00;
randomspin = false;
n_samples = 1000;
#Wang Landau Algorithm Parameters
target_e = -2.00;
target_width = 16.0;
n_entropic_samples = 1000;
n_asamples = 100;
a0 = 2.0;
```

In the above configuration file you can see 5 configuration options that determine the type of simulation that is going to be performed and 3/5 other options that are for these specific operations.

The first parameter, wanglandau, determines the type of simulation that is going to be run. If this value is false, the program runs a typical Metropolis Algorithm and reads the beta, randomspin and n_samples parameters further down the file. If the parameter is true, it runs the Wang Landau algorithm. This algorithm requires a few more parameters, like the target energy (target_e), the target width, the number of a_n samples and a starting value for the iterative calculation of a_n.

The other parameters are required by both of the algorithms, these determine the size of the lattice (dim_grid) and the number of q states (n_q). The parameter, coldstart, determines the initial state of the system be it ordered or disordered.

3.2 Code Operation

In this section, I will describe the operation of the program. As described above, there are two different modes in this project so I will explain each type separately. However in both modes, the first stage is the same. We read the file name of the configuration file from the input arguments and parse it into the variables. Then to calculate the angles for each different q state because they only need to be calculated once at the beginning. Now that all the variables are loaded in from the configuration file, it is closed and the lattice/grid variable is constructed as a 2D array of heap memory. The decision to allocate the memory as a 2D array was to improve the readability of the code. In most applications, it is preferable to construct a 1D array and access elements using Row-Major order to improve the access speed of the elements. In this case, the 2D array was preferable as it significantly improved the overall readability and intuitive understanding of the code. By default the C++ memory allocation operator, new[], doesn't assign any value to the assigned memory which means to initialise the system the program needs to run through every lattice point and assign it a value. In the case of a cold start, the program simply generates a random integer between 1 and q and assigned each lattice point to that value. In the case of a hot start, a new random number is generated for each point. From this point on ward the two operating modes diverge in their assigned functions so I will move on to the Metropolis algorithm

3.2.1 Metropolis

A description of the Metropolis algorithm is now required to ensure that the next stages of the program are understood. A Metropolis-Hastings algorithm is a Markov Chain Monte Carlo method to obtain random samples from a probability distribution. In general this technique is usually used to solve more complex multidimensional problems despite this it is highly adaptable and suitable for the problem at hand. The Metropolis-Hastings approach to solving this problem involves modification of a single lattice point at each step. There are two methods for determining which lattice point to use. The first method is the simplest, using a PRNG (Pseudo Random Number Generator) generate the coordinates of the lattice site which is to be modified. The second method involves progressing through each point in the lattice in order. In this project, I elected to use the second method because at the time it seemed to require less computation and as such be faster. However due to the nature of the 2D array used as the lattice it requires the use of two nested for loops which could slow down the process. Once the lattice site has been chosen from whatever method chosen, the program makes a random change to the state of that point and calls it a *Proposal State*. The energy change between the original state and the proposed state is calculated. Again there are two methods to do this, one a complete sweep across the lattice applying the Hamiltonian and using the sum and the other which looks only at the *links* between lattice points that have changed because everything else will cancel out. Again both methods have a function in the final source code but the second method is used as this reduces the total number of operations required. The energy difference, delta, between the two states is then put through a simple algorithm, shown below, to determine if the change is accepted or not.

Algorithm 1 Accepting or Rejecting a Proposed Change for the Metropolis Algorithm

if $delta < 0.0$ **then**
  ACCEPT
else
  $0 <= random() < 1$
  if $e^{-\beta \text{ delta}} > rand$ **then**
    ACCEPT
  else
    REJECT
  end if
end if

β is a variable that is provided in the configuration file. Now that the Metropolis-

Hastings method has been explained in the context of this particular problem the remaining behaviour of the Metropolis portion of the code can be explained. Before any measurements can be taken from this system, it needs to be thermalised so that it reaches equilibrium. The average time taken for the system to reach thermal equilibrium is known as the thermalisation time $\tau_e q$ and it varies upon lattice size. After the system thermalises, the program progresses onto the measurement component of the program. In this mode of operation, there are 3 measured quantities, the Acceptance, the Magnetisation and the Energy. The acceptance is the ratio of accepted changes against the total number of proposed states, the Magnetisation and Energy as described earlier in the report are shown below.

$$\begin{aligned} \mathcal{M} &= \sum_{i}^{V} \sigma_i \\ E &= \sum_{\{i,j\}} \delta_{Kr}(\sigma_{i,j}) \end{aligned} \tag{3.1}$$

Now using the n_samples parameter the program starts a FOR loop which performs an update and a measurement algorithm at each loop. By storing the recorded measurement at each step the program stores a 1D array of measurement values as it goes through the loop. When the loop is finished the program has finished the measurement stage and now starts to perform analysis on the measurements. The next stage will be described in a subsequent section.

3.2.2 Wang Landau

This mode of operation is significantly different to the Metropolis algorithm. After all of the parameters have been loaded and the lattice states set either randomly (hotstart) or all aligned (coldstart) the program needs to drive the system state into the desired energy band. The parameters, target_e and target_width which define the lattice are provided in the configuration file. In general, the target width is taken to be equal to the grid size in an attempt to match the conditions in the Guagnelli paper studying this particular problem for Q=10[2] but it can be set to smaller values to try and restrict the sampling region more. I found the simplest way to drive the system towards the target energy band was for each point on the lattice suggest a random state, if the difference in energy between this new state and the target is smaller than that of the original state accept it. The program does this repeatedly until it reaches it's target energy band this is one of the more time consuming parts of the code because it doesn't try to reduce the length of any interfaces, which would allow a significantly higher level of control and it can get stuck, especially in low β conditions. By the time the loop stops, the program will have reached it's target energy band. It is unlikely but it remains possible

that the energy of the system is exactly equal to the configuration file parameter due to the discrete nature of the energies that arises from the discrete number of possible states. The next stage of the program is to take the value a_0 provided by the configuration file and perform an update on the system. This time however the algorithm is slightly modified from the original.

Algorithm 2 Accepting or Rejecting a Proposed Change for the Wang Landau Algorithm

```
if outsideenergyband() then
  REJECT
else
  CONTINUE
  if δ < 0.0 then
    ACCEPT
  else
    0 <= random() <= 1
    if e^{-a_n δ} > random() then
      ACCEPT
    else
      REJECT
    end if
  end if
end if
```

After running this update algorithm across the lattice and taking a measurement of $E^\star$. $E^\star$ is the current energy less the target energy E_0. The program moves onto the iterative equation for a_n which is given below for $Q = 2$.

$$a_{n+1} = a_n + \frac{12}{L^2}\langle E^\star \rangle \tag{3.2}$$

For simulations that involve higher numbers of states taking the form for a_n to be

$$a_{n+1} = a_n + \frac{12}{4L + L^2}\langle E^\star \rangle \tag{3.3}$$

After updating the value for a_n the program returns to the beginning of a FOR loop, that runs a configurable number of times dependent on the value of n_asamples. The aforementioned FOR loop includes updating the lattice a number of times based upon the configuration parameter, n_entropic_samples taking measurements of the lattice and recalculating $E^\star$ to be used in the calculation of a_n using the method described above.

3.2.3 Adding a Twist

Adding a twist to the lattice requires a slight modification to the update algorithm used in the Wang Landau method. In my code, I add a twist to the lattice close to the midpoint in the x direction. It is trivial to calculate what point it occurs using the floor function provided by the C++ math library. The modifications to the code that was used in the Wang Landau method are simple and are switched on and off using the interface boolean parameter in the configuration file. This ensures a twist exists in the lattice and allows us to calculate Z^* that is required to study the interface tension. At the interface point, whose value is determined using $floor(L/2)$, the update algorithm is replaced by setting the value of the lattice on the other side of the interface plus a constant and then put the modulus function to ensure the new state wouldn't be a state that wasn't expected in the program.

3.3 Run

To get meaningful data from the code, it needs to be run multiple times with many different parameters. This means that my code is SPMD, Single Program Multiple Data which is a Subcategory of MIMD, Multiple Instruction Multiple Data. Despite design choices in my code causing it to run slower than expected, it was still sensible to explore a method of pregenerating the configurations and scheduling them to run as efficently as possible. To this end, as part of the process I wrote several shell scripts that were designed to improve this stage of the process. When writing these shell scripts I ran into a very useful utility GNU Parallel[6]. This utility allows a simple single threaded program to run with several different configurations on an arbitary number of cores. An example of one of these shell scripts is below.

```
rm -R -- */
rm output.dat
rm Q*.dat
filename='target.list'
filelines=`cat $filename`
gridsize=18
interface=false
sed -i '/interface\s=\s.*/c\interface = '$interface'' param
    .cfg
sed -i '/dim_grid\s=\s.*/c\dim_grid = '$gridsize'' param.
    cfg
```

```
sed -i '/target_width\s=\s.*/c\target_width = '$gridsize
   '.0' param.cfg

for q in 2 3 4 8 10
do
rm -R -- */
for i in $filelines
do
        mkdir energy$i
    cd energy$i
    cp ../param.cfg .
    sed -i '/target_e\s=\s.*/c\target_e = '$i'' param.cfg
    sed -i '/dim_q\s=\s.*/c\dim_q = '$q'' param.cfg
    cd ..
done

parallel -j16 --eta "cd energy{}; ../potts.app param.cfg;
   cd ../" :::: target.list

filelines=`cat $filename`
for i in $filelines
do
        anaverage=`tail -n 30 energy$i/an.dat | awk '{ sum
          +=$1} END {print sum/30}'`
    energy=$i
    echo "$energy $anaverage" >> Q$q.dat
done
done
```

This script, takes a template configuration file and fills out all of the parameters and creates a directory for the code to run in, runs the code on the number of cores determined by the j parameter of the parallel exeuctable and then collects the results averages the a_n and returns the results for each energy.

3.4 Output

The program was several modes of operation. In the Metropolis mode, the program creates several files that contain the results of the simulation for the measured parameters, Energy Per Unit Volume, Magnetisation Per Unit Volume, Specific Heat and the Magnetic Susceptibility. The output from a single file in this mode contains 3 columns. The value of β, the actual value of the measurement and the

error in the measurement. For the Wang Landau mode, the code returns a file with all of n measurements of a_n. Because the programs themselves are run in parallel with others the single outputs are appended to one another at the end so that, in the Metropolis case you get a list of the results at each β which is in an easily plottable format. In the case of the Wang Landau mode, the nature of the output is such that it isn't simply a case of appending each files contents together. As shown in the example above the final results of the output file are averaged and the averages are appended together to give a single output file.

Chapter 4

Results

From the earlier chapter I described the final form of the data output from the program. Below is a sample from the Metropolis data file.

0	0	0.000277831
0.05	0.00000183835	0.000301732
0.1	0.00000759666	0.000289046
...	...	...
2.5	0.000177274	0.000889706
2.55	0.000121696	0.000737807
2.6	0.000113462	0.000829744

The Wang Landau output file looks like this

-1.972222222	117.227
-1.916666667	114.667
-1.861111111	112.106
...	...
-0.1388888889	-28.7273
-0.08333333333	-26.1667
-0.02777777778	-28.7273

Using this data we can start constructing the physical quantities and plotting them.

4.1 Metropolis

4.1.1 Range of beta

For the Metropolis data, the first step is to plot the data to ensure the program produced expected behaviour. In this case expected behaviour for the Energy as a function of β is that at low β, high T, the energy would be nearing it's maximum and that for high β, low T the energy would be close to its minimum and at there would need to be a transition between these two points.

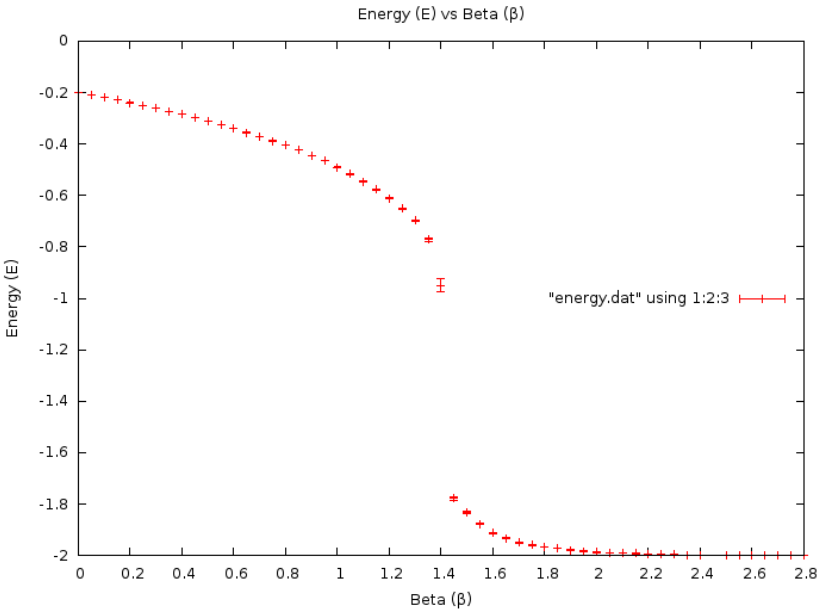


Figure 4.1: Graph showing Energy vs β

In the graph above, the expected behaviours are clearly visible. The data shows that the energy approaches it's maximum at low β and it's minimum at high β. The above graph is set for $Q = 10$ which has a phase transition at $\beta_c = \ln(1 + \sqrt{Q}) \approx 1.426$ in the thermodynamic limit, while the graph clearly shows a transition it isn't at this β_c but this is due to the finite size of the lattice used in this simulation.

For Magnetisation the expected behaviour is that for high β, low T, that magnetisation per unit volume would be equal to 1. At low β, high T, this property disappears and should approach a minimum. Again the expected behaviour is seen in the graph below. The phase transition occurs at around the same position as in the Energy case which is another quick check to make.

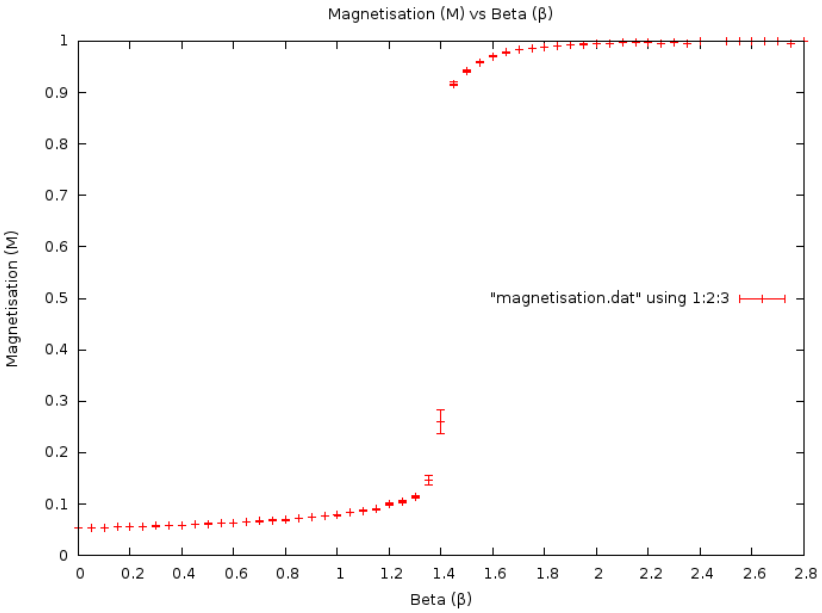
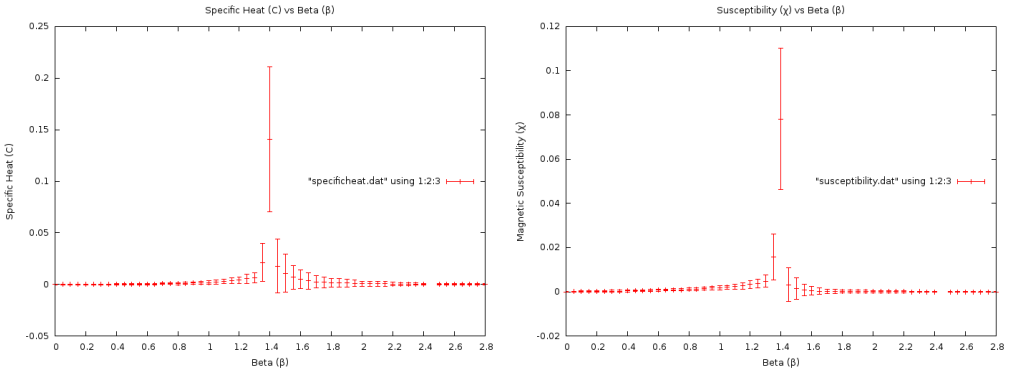


Figure 4.2: Graph showing Magnetisation vs β

For the other data files that are measured, which include the Specific Heat, C and the Magnetic Susceptibility,(χ) the behaviour should be 0 everywhere except at the phase transition as stated above, $\beta_c \approx 1.426$ in the thermodynamic limit.



(a) Q10 Graph showing Specific Heat (C) vs β

(b) Q10 Graph showing Magnetic Susceptibility (χ) vs β

The peaks in the two graphs above are obvious and fit with the expected results for the respective quantities.

While these quick checks are useful while debugging the code to ensure that the simulated behaviours are correct, on their own they don't provide enough

information to verify the code is working correctly this will be dealt with in the next step.

4.2 Wang Landau

4.2.1 Plotting the Output

The first step of dealing with the Wang Landau output is, like the Metropolis, is to plot it. This is to visually verify that the behaviour is as expected, in this case the iterative process for calculating a_n leads to data that starts at an initial value a_0 and that will converge towards and oscillate around in a band of acceptable values.

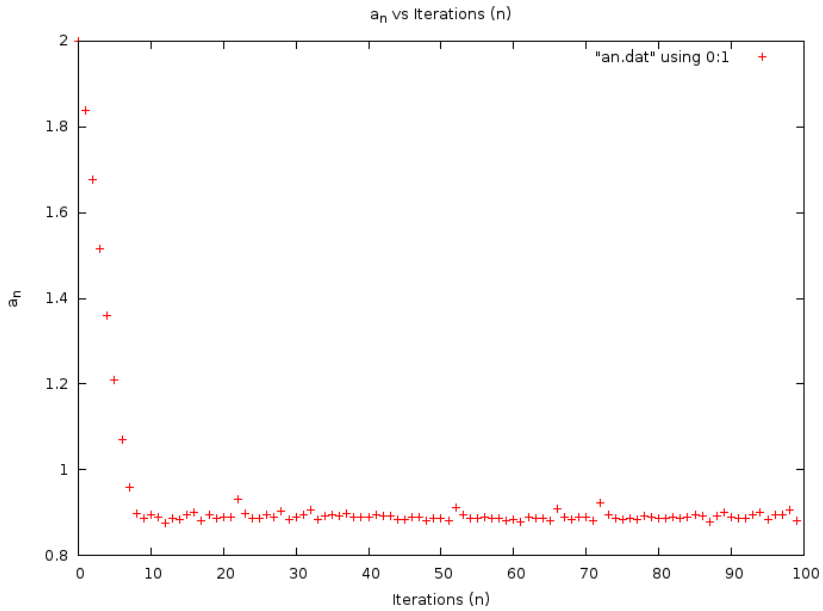


Figure 4.3: Graph showing the a_n convergence vs n

It is clear that in the above graph the behaviour is as expected. The value of a_n converges extremely quickly towards it's target value and oscillates slightly around its value.

4.2.2 Are the results compatible with the Metropolis?

In an earlier section, I stated that the information provided by the metropolis wasn't enough information to verify the code was working correctly. The Metropolis part of the program was a stepping stone to the implementation of the Wang

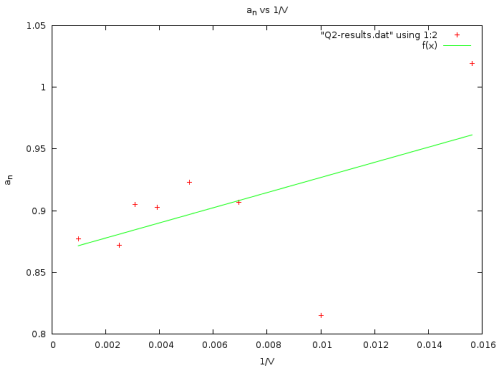
Landau algorithm that was the primary portion of this project. That being said, you can verify the results returned by the Wang Landau mode of this project by using the energy values returned by the Metropolis code at the β_c at the thermodynamic limit. a_n should be roughly equivalent to the β required to keep the system at the target energy using this information and the value of β_c I calculated the value of a_n at the critical point using the energy value returned from the Metropolis data. This leads to a data file looking similar to the one below.

8	-1.74296	0.00591688
10	-1.72614	0.00648999
12	-1.71512	0.0058484
14	-1.69895	0.0067316
...	...	...
28	-1.7049	0.00379986
30	-1.7069	0.00395975
32	-1.72059	0.00357536

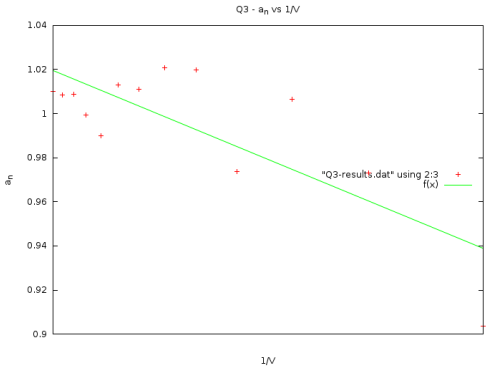
The first column is the lattice size, the 2^{nd} column is the Energy at β_c and the third column is the error in the 2^{nd} column. Using this data I wrote several shell scripts that are designed to run the program in the Wang Landau mode using the target energy provided in the 2^{nd} column. Taking the results from the Wang Landau mode of the program using the shell scripts described above to collect the values of a_n for the various grid sizes you can plot them against the inverse volume. At the thermodynamic limit, the inverse volume should be 0. If you fit the results with a linear function $y = ax + b$, it's intercept represents the value at the thermodynamic limit. The intercepts for this particular test are shown in the table below.

Q	Theoretical	Result	Error
2	0.881373587	0.865616	0.03173
3	1.005052539	0.996294	0.01123
4	1.098612289	1.11538	0.002026
8	1.342454046	1.34759	0.008933
10	1.426062439	1.43679	0.0152

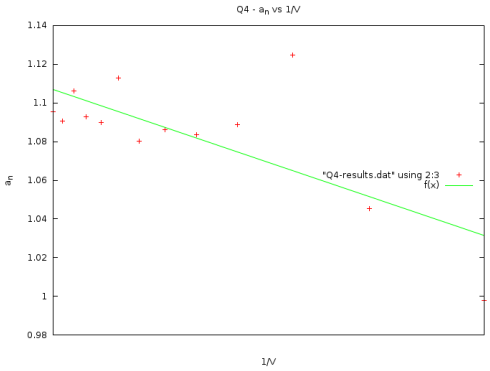
A plot of the data as it appeared is now provided below.



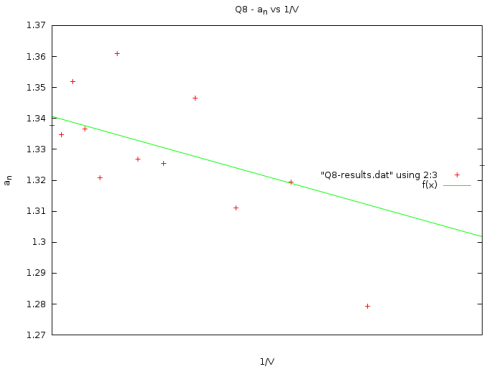
(a) Q2 $a_n = 0.881373 \pm 0.03173$



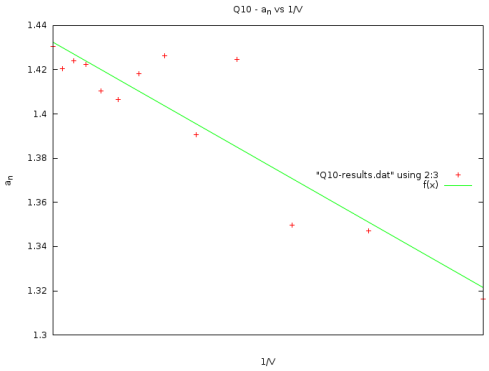
(b) Q3 $a_n = 0.996294 \pm 0.01123$



(c) Q4 $a_n = 1.11538 \pm 0.02026$



(d) Q8 $a_n = 1.34759 \pm 0.008933$



(e) Q10 $a_n = 1.43679 \pm 0.0152$

The above graphs show that in the thermodynamic limit, the calculated values of a_n agree with the β_c which is a form of verification to the results from the Wang Landau.

4.2.3 Calculating the Partition Function

The next stage of the project, which is required to calculate the Partition Function, Z, is to generate values of a_n at regular intervals between the minimum and maximum energies. The regular intervals are determined by the energy band width. That means that the midpoints of these intervals for a small grid size for $Q = 2$ would be as shown below.

Minimum	Maximum	Midpoint
-2	-1.9375	-1.96875
-1.9375	-1.875	-1.90625
-1.875	-1.8125	-1.84375
...	...	...
-0.1875	-0.125	-0.15625
-0.125	-0.0625	-0.09375
-0.0625	0.000	-0.03125

Running the Wang Landau mode of the program with the midpoint as the target energy and the target width as the grid size.

The output from the program for each midpoint when plotted against the energy leads to the plot below.

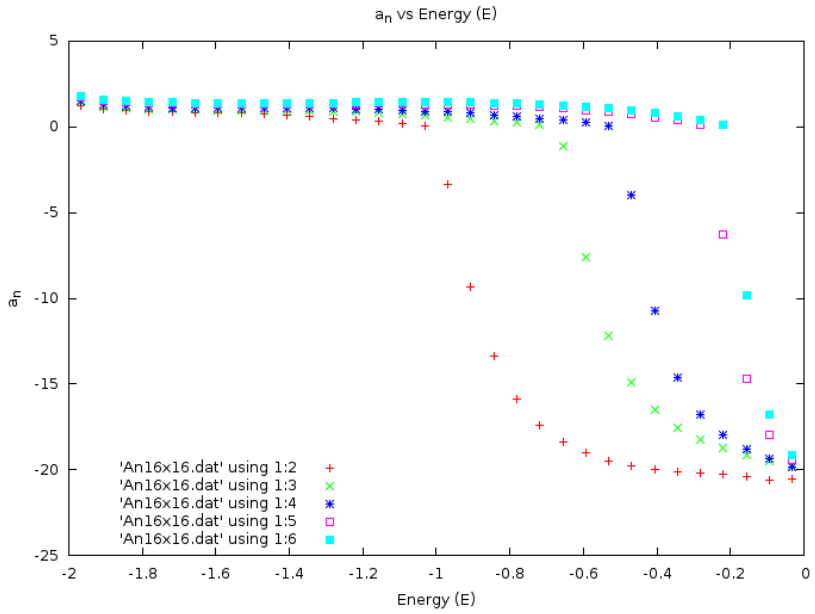


Figure 4.4: Graph showing the plot of a_n across the energy bands for various Q

This graph has issues with scaling due to the fact that the maximum energy is dependant on Q. The maximum energy is dependant on Q because the number of configurations that can represent each energy increases with Q. A rescaled graph shows a clearer picture for the different Q.

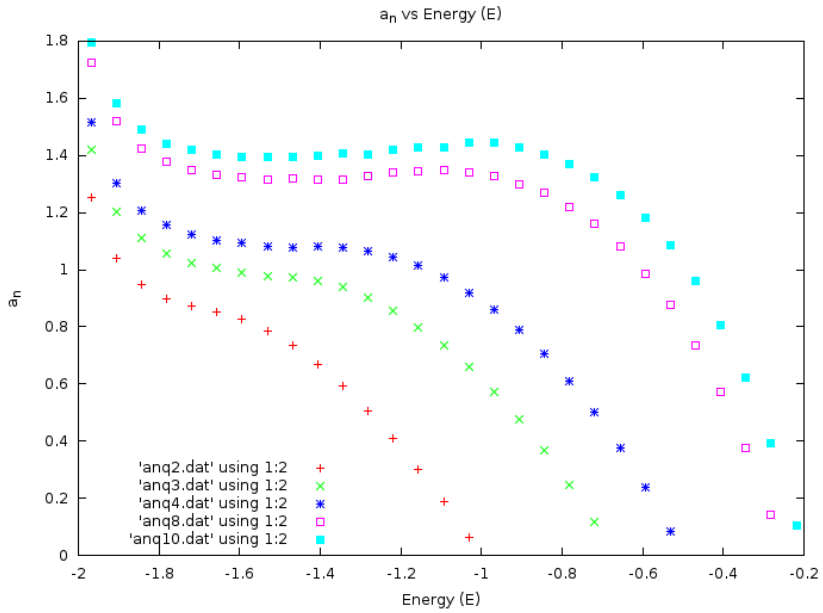
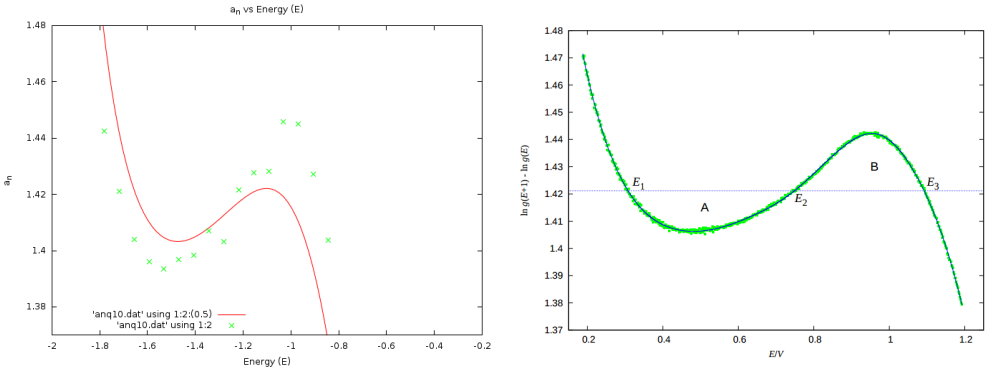


Figure 4.5: Graph showing the rescaled plot of a_n across the energy bands for various Q

A comparing my results with a similar result from the paper by Guagnelli for Q=10 is shown below.



(a) Q10 Graph showing the rescaled plot of a_n across the energy bands from the program

(b) Q10 Graph showing the rescaled plot of a_n across the energy bands from the Guagnelli paper[5]

The primary discrepancies between the two results is likely to occur due to the difference in lattice sizes used to generate the results and some arbitrary scaling factors used in the simulation. To calculate the partition function, Z, from this data I turned to the symbolic computational mathematics tool kit Mathematica to calculate the constants that keep the Piecewise function of $\log \rho(E)$ continuous. The table 6.2.2 contains these constants for each of the 5 Q tested, the Null values occur due to the aforementioned scaling issues.

By plotting a Piecewise function in side Mathematica where each component is of the form.

$$f(E) = C_i + a_i(E - E_i) \tag{4.1}$$

E_i is the Midpoint of the interval, C_i is the constant value for that interval to ensure continuity and a_i is the measured value of a_n at the midpoint. A sample of the piecewise function that Mathematica generates for Q2 is provided in the table at 6.2.3. Plotting the these Piecewise functions is a good way to ascertain the continuity of the newly generated functions. However due to the way Mathematica renders these functions there appears to be gaps in the rendered output. Included below is the plot for the Q2 piecewise function of $\log g\,(E)$.

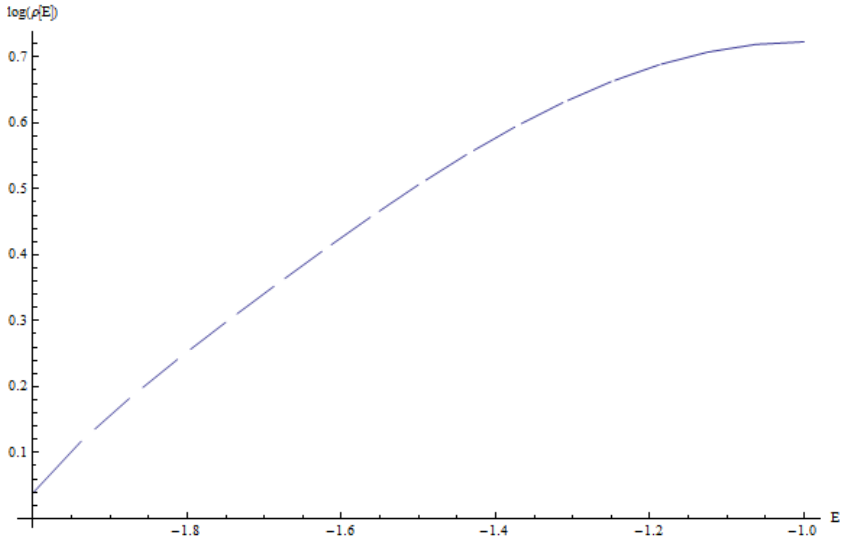


Figure 4.6: Graph showing the $\log g\,(E)$ vs E

The plots for Q3, Q4, Q8, Q10 can be found in the appendix at 6.2.4. All the plots show similar behaviour for all the various values of Q. Now the only piece of information that remains to provide the complete Density of States is the constant, c_0 that is added to all of the piecewise functions. As explained earlier this value

is calculated using the following equation

$$C_0 = \frac{2L^2}{\int g(E)} \tag{4.2}$$

In this case, $g(E)$ is the incomplete Density of States. The integral can be computed easily using the tools provided by Mathematica however it could also be computed using a simple Simpson's rule integrator due to the linear nature of the approximations used to calculate the Density of States. After adding this constant to the incomplete Density of States, you get the complete Density of States for that simulation. The next stage of the processing for the Mathematica program is to calculate the Partition Function. The partition function can be written in the form.

$$\mathcal{Z} = \int g(E) e^{-\beta E} \tag{4.3}$$

Using Mathematica to perform the integration numerically across the sampled energy range you can extract the Partition Function as a function of β. With this done the analysis of this part of the project is complete.

4.3 Wang Landau with a Twist

After modifying the configuration file to add the twist into the simulation and running the simulation across various lattice sizes. After extracting the data from the programs output files and importing it into Mathematica the code is nearly indistinguishable from that described earlier for the Wang Landau data so to ensure that I am not repeating any material I will describe the next stages of analysis here. The plots of the incomplete Density of States are available in the appendix at 6.2.5.

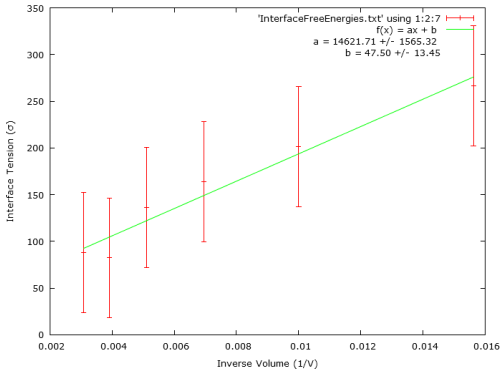
Having now calculated the Partition Functions for the untwisted Z and twisted Z^* lattice, the next stage is to calculate the interface free energy. The Interface Free Energy as described earlier can be written as

$$F_I(L) = -\log \frac{Z^*}{Z} - \log L \tag{4.4}$$

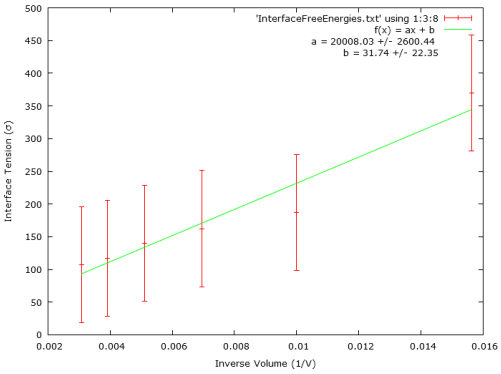
The Mathematica notebook provided in the appendix 6.2.6 is a copy of the working code used to perform all of the analysis for the Wang Landau model. Taking the results from the Mathematica notebook for the Interface Free Energy at β_c and using the form for the interface tension [3].

$$\sigma = \lim_{L\to\infty} \frac{F_I}{V_{D-1}} \tag{4.5}$$

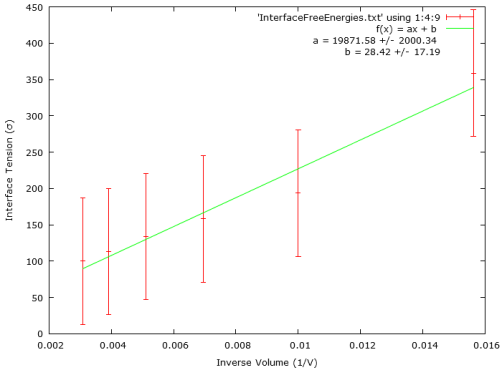
Taking the results from the Mathematica notebook for the various grid sizes and taking a fit of the plot to the thermodynamic limit leads to the results below.



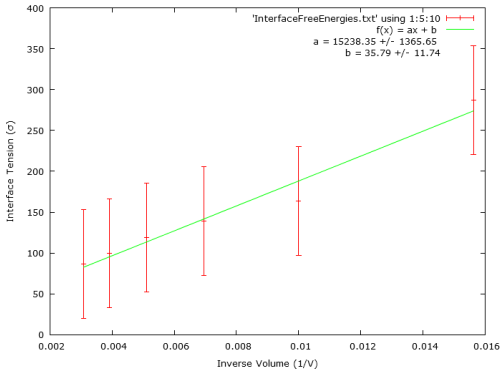
(a) Q2 $\sigma = 47.50 \pm 13.45$



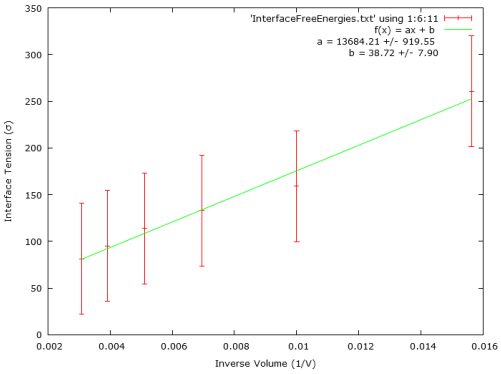
(b) Q3 $\sigma = 31.74 \pm 22.35$



(c) Q4 $\sigma = 28.42 \pm 17.19$



(d) Q8 $\sigma = 35.79 \pm 11.74$



(e) Q10 $\sigma = 38.72 \pm 7.90$

Chapter 5

Conclusion and Reflection

Wibble Wobble

Bibliography

[1] Working draft, standard for programming language c++, 2011.

[2] M. Guagnelli. Sampling the density of states, 2012.

[3] A. Hietanen and B. Lucini. Interface tension of the 3d 4-state potts model using the wang-landau algorithm, 2011.

[4] Mark A. Lindner. libconfig - a library for processin structured configuration files, 2012.

[5] Elliott W. Montroll, Renfrey B. Potts, and John C. Ward. Correlations and spontaneous magnetization of the two dimensional ising model. *Journal of Mathematical Physics*, 4(2):308–322, 1963.

[6] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

Chapter 6

Appendices

6.1 Metropolis

6.1.1 Q10EnergyBeta16x16RangeofBeta.dat

```
0 −0.200109 0.00025786
0.05 −0.209624 0.00027772
0.1 −0.21899 0.000264747
0.15 −0.22827 0.000328895
0.2 −0.239697 0.00034315
0.25 −0.249341 0.000322799
0.3 −0.260898 0.000375688
0.35 −0.272723 0.000393963
0.4 −0.284388 0.000390524
0.45 −0.297082 0.000406967
0.5 −0.311139 0.000454669
0.55 −0.324349 0.000481373
0.6 −0.339617 0.000556149
0.65 −0.355262 0.000571083
0.7 −0.370582 0.000588718
0.75 −0.388206 0.000693746
0.8 −0.404432 0.000805053
0.85 −0.423668 0.000777047
0.9 −0.44452 0.000851305
0.95 −0.465189 0.000954719
1 −0.49025 0.00117225
1.05 −0.516876 0.00126606
1.1 −0.546703 0.00145358
```

```
1.15 -0.57747 0.0017864
1.2 -0.611437 0.00197478
1.25 -0.650466 0.00275758
1.3 -0.696883 0.00279037
1.35 -0.772292 0.00896837
1.4 -0.948917 0.0261211
1.45 -1.77709 0.00725456
1.5 -1.83122 0.00498143
1.55 -1.87696 0.00297309
1.6 -1.91116 0.00235711
1.65 -1.93321 0.00199471
1.7 -1.94931 0.00146225
1.75 -1.95923 0.00132297
1.8 -1.96639 0.00101562
1.85 -1.97237 0.001026
1.9 -1.97907 0.000982634
1.95 -1.98302 0.000793976
2 -1.98726 0.000609572
2.05 -1.98861 0.000599667
2.1 -1.99062 0.000576544
2.15 -1.99192 0.000539044
2.2 -1.99415 0.000428889
2.25 -1.9955 0.00038934
2.3 -1.99605 0.000289569
2.35 -1.99703 0.000253472
2.4 -1.99816 0.000224678
2.5 -1.99819 0.000216858
2.55 -1.99873 0.000179629
2.6 -1.99891 0.000202063
2.65 -1.99915 0.000155375
2.7 -1.99914 0.000163213
2.75 -1.99921 0.000150579
2.8 -1.99939 0.000147808
```

6.1.2 Q10MagnetisationBeta16x16RangeofBeta.dat

```
0 0.0536258 0.000282599
0.05 0.0540781 0.000279901
0.1 0.0542501 0.000287509
0.15 0.0554078 0.000295225
0.2 0.0564436 0.000321571
```

0.25 0.0566199 0.000357871
0.3 0.0576936 0.000371024
0.35 0.0579753 0.000341963
0.4 0.0585935 0.000385968
0.45 0.0603006 0.000411977
0.5 0.0615635 0.000414219
0.55 0.0630296 0.000409879
0.6 0.0633744 0.000417007
0.65 0.0654273 0.000408695
0.7 0.0667299 0.000475517
0.75 0.06832 0.000532082
0.8 0.0693393 0.000636549
0.85 0.0719688 0.00063416
0.9 0.0744893 0.000661863
0.95 0.0769229 0.000845882
1 0.0786101 0.000861817
1.05 0.083604 0.000845359
1.1 0.0866739 0.00114479
1.15 0.0903927 0.00144828
1.2 0.0996223 0.00169752
1.25 0.105431 0.00180421
1.3 0.114108 0.00265604
1.35 0.14669 0.00880269
1.4 0.260539 0.0227491
1.45 0.916998 0.00384802
1.5 0.940809 0.00234376
1.55 0.958311 0.00120807
1.6 0.970224 0.000934043
1.65 0.977959 0.000659634
1.7 0.983036 0.000501082
1.75 0.986376 0.000447583
1.8 0.988708 0.000321665
1.85 0.990527 0.000337317
1.9 0.992693 0.00032042
1.95 0.993967 0.000271542
2 0.995432 0.000211862
2.05 0.99583 0.000202744
2.1 0.996618 0.000202872
2.15 0.997036 0.000191388
2.2 0.997793 0.000166809

```
2.25  0.995348  7.58233e-05
2.3  0.998512  0.000107969
2.35  0.995623  4.63501e-05pn
2.4  0.9993  8.78052e-05
2.5  0.999318  7.76653e-05
2.55  0.999527  6.85556e-05
2.6  0.999585  7.80653e-05
2.65  0.999671  6.15873e-05
2.7  0.999652  6.92973e-05
2.75  0.995975  2.34157e-05
2.8  0.999773  5.52442e-05
```

6.1.3 Q10SpecificHeatBeta16x16RangeofBeta.dat

```
0  0  0.000277831
0.05  1.83835e-06  0.000301732
0.1  7.59666e-06  0.000289046
0.15  1.8079e-05  0.000362044
0.2  3.32178e-05  0.000381555
0.25  5.3092e-05  0.000361097
0.3  8.05329e-05  0.000424603
0.35  0.000116444  0.00045009
0.4  0.000153867  0.000448959
0.45  0.000200627  0.000474747
0.5  0.000266681  0.000538302
0.55  0.00033652  0.000575057
0.6  0.000424915  0.000675175
0.65  0.000509799  0.000702659
0.7  0.000625063  0.000735869
0.75  0.000778356  0.000885296
0.8  0.000890101  0.00104076
0.85  0.00114436  0.00102571
0.9  0.00135769  0.00114159
0.95  0.00157159  0.00131239
1  0.00188835  0.00166014
1.05  0.0023751  0.00183388
1.1  0.00278538  0.00217225
1.15  0.00348612  0.00274892
1.2  0.00416383  0.00314387
1.25  0.00556986  0.0046199
1.3  0.00643498  0.00486409
```

```
1.35  0.0214081  0.0186271
1.4  0.140861  0.0700809
1.45  0.0178388  0.0262297
1.5  0.010962  0.0184243
1.55  0.00702258  0.0114387
1.6  0.00515308  0.00921808
1.65  0.00358491  0.00791604
1.7  0.00256486  0.00585533
1.75  0.00220429  0.00532126
1.8  0.00170033  0.00410681
1.85  0.00164258  0.00415803
1.9  0.0013511  0.00399262
1.95  0.00107063  0.00323329
2  0.000802398  0.00249073
2.05  0.000762133  0.00244968
2.1  0.000673591  0.00235798
2.15  0.000638553  0.00220467
2.2  0.000472682  0.00175664
2.25  0.000351523  0.00159687
2.3  0.000319642  0.00118773
2.35  0.00025569  0.00103987
2.4  0.000164989  0.000922352
2.5  0.000177274  0.000889706
2.55  0.000121696  0.000737807
2.6  0.000113462  0.000829744
2.65  0.000100027  0.000637751
2.7  9.9788e-05  0.000670199
2.75  9.32963e-05  0.000618449
2.8  7.96827e-05  0.000606782
```

6.1.4 Q10SusceptibilityBeta16x16RangeofBeta.dat

```
0  0  0.00028509
0.05  4.29466e-05  0.000282763
0.1  8.58881e-05  0.000290136
0.15  0.000135661  0.000298115
0.2  0.000184004  0.000325097
0.25  0.000230868  0.00036167
0.3  0.000297625  0.00037515
0.35  0.000341974  0.000345617
0.4  0.000402675  0.000390005
```

```
0.45 0.000479985 0.000416988
0.5 0.000555462 0.000419142
0.55 0.000639779 0.000415187
0.6 0.000702171 0.000422092
0.65 0.000797168 0.000414669
0.7 0.000880077 0.000481797
0.75 0.000993099 0.000539691
0.8 0.00109877 0.000646136
0.85 0.00125121 0.000643803
0.9 0.00138957 0.000673981
0.95 0.00166783 0.000861908
1 0.00178388 0.000879005
1.05 0.00208251 0.000864356
1.1 0.0023205 0.00116932
1.15 0.00272497 0.00148544
1.2 0.00330586 0.00174601
1.25 0.00370356 0.00185931
1.3 0.00485822 0.0027615
1.35 0.0158726 0.010339
1.4 0.0781662 0.0319328
1.45 0.00324901 0.00774562
1.5 0.00153613 0.00483858
1.55 0.000770371 0.00257786
1.6 0.000535052 0.00201244
1.65 0.000292019 0.00144091
1.7 0.000207642 0.00110016
1.75 0.000169414 0.000984836
1.8 0.00012585 0.000710757
1.85 0.000119812 0.000746123
1.9 9.35882e-05 0.000710222
1.95 7.74758e-05 0.000602487
2 5.65397e-05 0.000471058
2.05 5.37267e-05 0.000450788
2.1 4.51293e-05 0.000451287
2.15 4.36806e-05 0.00042575
2.2 3.4398e-05 0.000371114
2.25 9.52731e-06 0.000168444
2.3 2.20644e-05 0.000240564
2.35 5.39795e-06 0.000103016
2.4 1.10412e-05 0.000195722
```

```
2.5  1.11168e-05  0.00017311
2.55  7.38777e-06  0.000152878
2.6  7.03322e-06  0.000174046
2.65  6.69305e-06  0.000137209
2.7  6.75523e-06  0.000154427
2.75  1.38085e-06  5.20999e-05
2.8  4.30191e-06  0.000123173
```

6.2 Wang Landau

6.2.1 An16x16Convergence.dat

```
2
1.83742
1.67596
1.51659
1.36017
1.21047
1.07115
0.958293
0.898123
0.886636
0.894645
0.890632
0.876113
0.886485
0.882993
0.893742
0.900179
0.882147
0.896148
0.885572
0.890101
0.889612
0.930086
0.897667
0.886961
0.885495
0.895436
0.889006
0.904736
```

0.884794
0.88835
0.894414
0.906143
0.885411
0.893166
0.895981
0.892837
0.896723
0.888887
0.888455
0.889703
0.896108
0.891426
0.891259
0.884917
0.882937
0.888925
0.889294
0.879971
0.887493
0.885633
0.881775
0.912052
0.894377
0.885789
0.886023
0.890293
0.887923
0.886067
0.881623
0.883337
0.879752
0.888423
0.885911
0.8877
0.881466
0.908074
0.889962
0.88437

0.890146
0.890344
0.881311
0.923801
0.895426
0.887581
0.883116
0.887668
0.883806
0.893626
0.890971
0.886484
0.885689
0.888526
0.885702
0.888504
0.895164
0.892044
0.879771
0.892959
0.900478
0.888948
0.887261
0.888088
0.894206
0.901281
0.885127
0.896582
0.896501
0.90774
0.880374

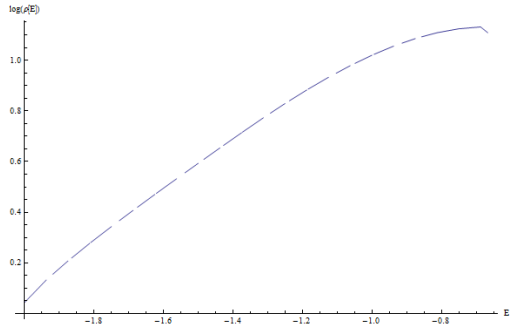
6.2.2 Continuity Constants

$Q2$	$Q3$	$Q4$	$Q8$	$Q10$
0.0783575	0.0887444	0.0948331	0.107823	0.112169
0.149986	0.170668	0.182925	0.209201	0.217729
0.212082	0.242924	0.261318	0.301153	0.313728
0.269831	0.310671	0.335173	0.38864	0.405326
0.325173	0.375721	0.40646	0.473809	0.494806
0.378996	0.439152	0.47602	0.557657	0.583087
0.431435	0.501496	0.544567	0.640646	0.670586
0.481899	0.562968	0.61251	0.723112	0.75776
0.529428	0.623959	0.679999	0.805459	0.844962
0.573241	0.684446	0.74747	0.887795	0.932316
0.612643	0.743848	0.814892	0.970099	1.01998
0.647018	0.801437	0.881851	1.05277	1.10781
0.675715	0.856414	0.947792	1.1361	1.19608
0.69803	0.908095	1.0121	1.21999	1.28512
0.713377	0.955984	1.0742	1.30418	1.37436
0.721189	0.999568	1.13333	1.3882	1.46418
Null	1.03814	1.189	1.4716	1.55451
Null	1.07098	1.24065	1.55375	1.64428
Null	1.09738	1.28747	1.63399	1.73275
Null	1.11655	1.32871	1.71175	1.81937
Null	1.12791	1.36349	1.78614	1.9035
Null	Null	1.39094	1.85622	1.98427
Null	Null	1.41012	1.92089	2.06066
Null	Null	1.42016	1.97909	2.13154
Null	Null	Null	2.02948	2.19551
Null	Null	Null	2.0704	2.25085
Null	Null	Null	2.10006	2.29553
Null	Null	Null	2.11625	2.32719
Null	Null	Null	Null	2.34277
Null	Null	Null	Null	Null
Null	Null	Null	Null	Null
Null	Null	Null	Null	Null

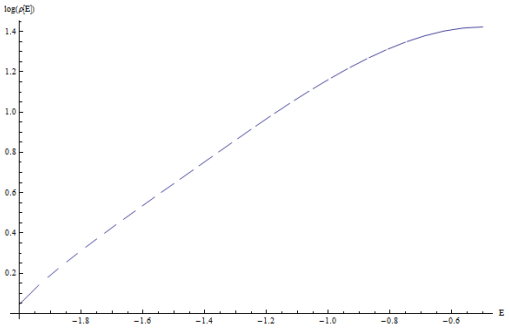
6.2.3 Q2 Piecewise Function

$$
\begin{array}{cc}
1.25372(x+1.96875)+0.0783575 & -2 \leq x \leq -1.9375 \\
1.03838(x+1.90625)+0.149986 & -1.9375 \leq x \leq -1.875 \\
0.948715(x+1.84375)+0.212082 & -1.875 \leq x \leq -1.8125 \\
0.899257(x+1.78125)+0.269831 & -1.8125 \leq x \leq -1.75 \\
0.87168(x+1.71875)+0.325173 & -1.75 \leq x \leq -1.6875 \\
0.85064(x+1.65625)+0.378996 & -1.6875 \leq x \leq -1.625 \\
0.827427(x+1.59375)+0.431435 & -1.625 \leq x \leq -1.5625 \\
0.787394(x+1.53125)+0.481899 & -1.5625 \leq x \leq -1.5 \\
0.733548(x+1.46875)+0.529428 & -1.5 \leq x \leq -1.4375 \\
0.668473(x+1.40625)+0.573241 & -1.4375 \leq x \leq -1.375 \\
0.592389(x+1.34375)+0.612643 & -1.375 \leq x \leq -1.3125 \\
0.507607(x+1.28125)+0.647018 & -1.3125 \leq x \leq -1.25 \\
0.410693(x+1.21875)+0.675715 & -1.25 \leq x \leq -1.1875 \\
0.303397(x+1.15625)+0.69803 & -1.1875 \leq x \leq -1.125 \\
0.187714(x+1.09375)+0.713377 & -1.125 \leq x \leq -1.0625 \\
0.0622752(x+1.03125)+0.721189 & -1.0625 \leq x \leq -1
\end{array}
$$

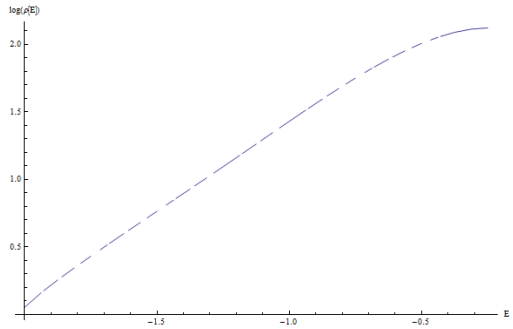
6.2.4 Q3 Q4 Q8 Q10 $Log(g(E))$



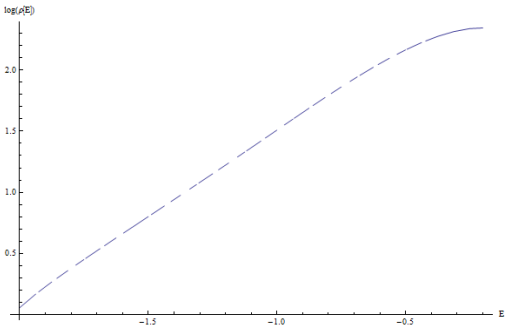
(a) Graph showing the $\log g\,(E)$ vs E for Q3



(b) Graph showing the $\log g\,(E)$ vs E for Q4

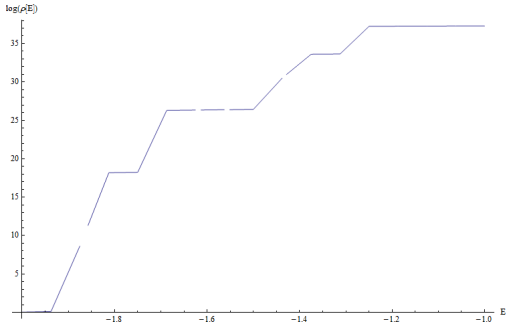


(c) Graph showing the $\log g\,(E)$ vs E for Q8

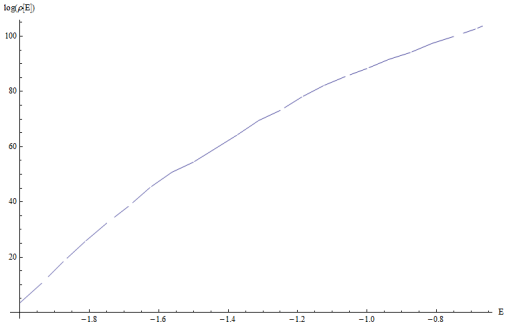


(d) Graph showing the $\log g\,(E)$ vs E for Q10

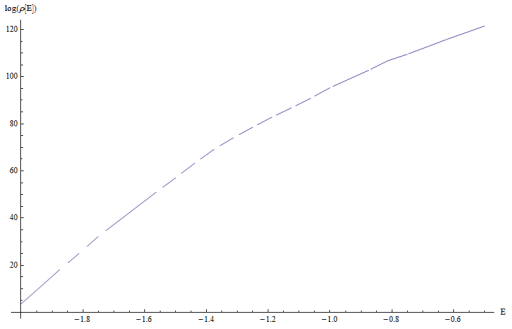
6.2.5 Q3 Q4 Q8 Q10 $Log(g(E))$ Twisted



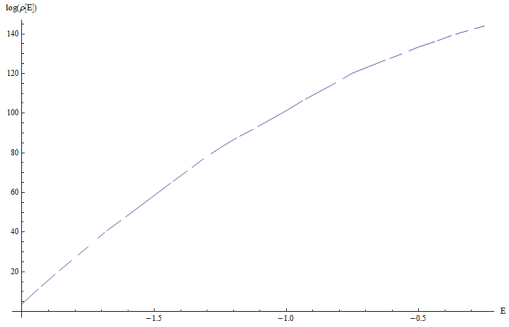
(e) Graph showing the $\log g\,(E)$ vs E for Q2



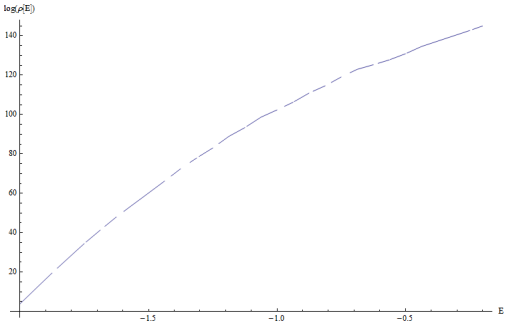
(f) Graph showing the $\log g\,(E)$ vs E for Q3



(g) Graph showing the $\log g\,(E)$ vs E for Q4



(h) Graph showing the $\log g\,(E)$ vs E for Q8



(i) Graph showing the $\log g\,(E)$ vs E for Q10

6.2.6 Mathematica Data Processing Notebook

Process simulated data for all Q values simulataneously

Clear all previously calculated variable values

```
ClearAll["Global*"]
```

Set the grid size for the input data and calculate data

```
GridSize = L;
```

```
delta = (GridSize)/(GridSize^2);
```

Now Import the Lattice Data from the spreadsheet

```
DataGrid = ImportString[“”, “TSV”];
```

Process the Perioidic Lattice data

Define a function that converts a list {1,2,3,4,5} to access the correct element in the imported data

```
CorrectRegColumn[q_]:=If[q==2, 4, If[q==3, 5, If[q==4, 6, If[q==8, 7, If[q==10, 8, 4]]]]]
```

Define the function that calculates the continuity constants as a function of midpoint number and q.

```
ContinuityConstantsReg[k_, q_]:=(DataGrid[[1]][[CorrectRegColumn[q]]]/2 + Sum[DataGrid[[i]][[CorrectRegColumn[q]]],
{i, 1, k − 1}] + DataGrid[[k]][[CorrectRegColumn[q]]]/2) ∗ delta
```

Define a function of E done as x for simplicity and q using the Piecewise function generator natively embedded in Mathematica

```
S0[x_, q_]:=Piecewise[Table[{ContinuityConstantsReg[i, q] + DataGrid[[i]][[CorrectRegColumn[q]]](x − DataGrid[[i]][[3]]),
DataGrid[[i]][[1]]<=x<=DataGrid[[i]][[2]]}, {i, 1, Length[DataGrid]}]]
```

Plot a graph across the Energy Range {-2,-2/q} for all of the Q values being studied to ensure that continuity occurs as expected.

```
Table[Plot[S0[x, q], {x, −2, −2/q}, ImageSize->Large, AxesLabel->{E, Log[ρ[E]]}], {q, {2, 3, 4, 8, 10}}]
```

Calculate C0 for the Periodic Lattice

```
C0regular[q_]:=(2 ∗ (GridSize^2))/(NIntegrate[S0[x, q], {x, −2, −2/q}])
```

Define the DoS for the Periodic Lattice

```
DoSregular[x_, q_]:=Exp[C0regular[q] + S0[x, q]]
```

Define the Partition Function for the Periodic Lattice

```
Zreg[beta_, q_]:=NIntegrate[Log[DoSregular[x, q]] + beta ∗ x, {x, −2, −2/q}]
```

Now to process the twisted Data

Define a function that converts a list {1, 2, 3, 4, 5} to access the correct element in the imported data

1

CorrectIntColumn[q_]:=If[q==2, 9, If[q==3, 10, If[q==4, 11, If[q==8, 12, If[q==10, 13, 9]]]]]

Define the function that calculates the continuity constants as a function of midpoint number and q.

ContinuityConstantsInt[k_, q_]:=(DataGrid[[1]][[CorrectIntColumn[q]]]/2 + Sum[DataGrid[[i]][[CorrectIntColumn[q]]],

{i, 1, k − 1}] + DataGrid[[k]][[CorrectIntColumn[q]]]/2) ∗ delta

Define a function of E done as x for simplicity and q using the Piecewise function generator natively embedded in Mathematica

S0int[x_, q_]:=Piecewise[Table[{ContinuityConstantsInt[i, q] + DataGrid[[i]][[CorrectIntColumn[q]]](x − DataGrid[[i]][[3]]),

DataGrid[[i]][[1]]<=x<=DataGrid[[i]][[2]]}, {i, 1, Length[DataGrid]}]]

Plot a graph across the Energy Range {-2,-2/q} for all of the Q values being studied to ensure that continuity occurs as expected.

Table[Plot[S0int[x, q], {x, −2, −2/q}, ImageSize->Large, AxesLabel->{E, Log[ρ[E]]}], {q, {2, 3, 4, 8, 10}}]

Calculate C0 for the Twisted Lattice

C0interface[q_]:=(2 ∗ (GridSize^2))/(NIntegrate[S0int[x, q], {x, −2, −2/q}])

Define the DoS for the Twisted Lattice

DoSinterface[x_, q_]:=C0interface[q] + Exp[S0[x, q]]

Define the Partition Function for the Periodic Lattice

Zint[beta_, q_]:=NIntegrate[Log[DoSinterface[x, q]] + beta ∗ x, {x, −2, −2/q}]

NowtocalculatetheFreeEnergyoftheInterfaceattheCriticalPoint

atcalculateCriticalEnergyFreeInterfaceNowofPointthe[3]to

Calculate the Interface Free Energy from the Twisted and Periodic Partiton Functions

IntFreeEnergy[beta_, q_]:= − Log[Zint[beta, q]/Zreg[beta, q]] − Log[GridSize]

Table[IntFreeEnergy[Log[1 + Sqrt[q]], q], {q, {2, 3, 4, 8, 10}}]

2