VALENCE BOND QUANTUM MONTE CARLO

By

ANN B KALLIN

TITLE: Valence Bond Quantum Monte Carlo

AUTHOR: Ann Berlinsky Kallin

SUPERVISOR: Dr. E. S. Sørensen

Contents

1 Introduction

1.1 Project

In this project we study the ground state properties of 1-dimensional Heisenberg spin chains using Monte Carlo methods in the valence bond basis. The ground state energy is determined for several system sizes and compared to the Bethe ansatz[3] solution (the exact solution). We also explore the effects of changes in the parameters of the valence bond quantum Monte Carlo scheme.

1.2 Motivation

To learn about quantum Monte Carlo treatments of quantum spin systems using one of the simplest methods – valence bond quantum Monte Carlo – applied to the simplest quantum system – a one-dimensional Heisenberg spin chain.

2 Background

2.1 Classical Monte Carlo

2.1.1 The Ising Model

The Ising model is the simplest theoretical model used to represent a two-state system, such as the spins of electrons in magnetic material. The model consists of a lattice in which each lattice point can take one of two values: +1 or -1, corresponding to spin up and spin down respectively. Figure 1 gives an example of a possible 4x4 lattice.

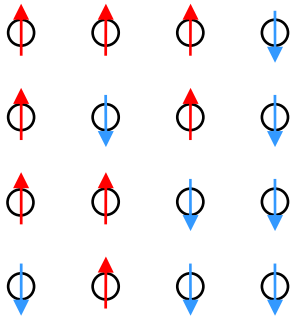


Figure 1 – A possible 4x4 Ising model lattice.

The Ising model can also represent gases, where the two lattice point values represent the presence or absence of a molecule. We can represent binary alloys as well, with the lattice values corresponding to the first or second of the two alloys.

The Ising model is used to study phase transitions in any of these systems. For magnetic materials, the Hamiltonian for spin interactions is[1]

$$H_{Ising} = -J \sum_{\langle i,j \rangle} S_i S_j \tag{1}$$

where J is the strength of the interaction, S_i and S_j are the spin values (±1) at lattice sites i and j respectively, and $\langle i, j \rangle$ denotes a sum over all nearest neighbours i and j. The strength of interaction J is positive for ferromagnetic materials and negative for antiferromagnetic materials.

In lattices of two dimensions and higher, it has been shown that the Ising model has a phase transition at the critical temperature T_c below which the spins will align parallel or antiparallel for ferromagnetic and antiferromagnetic materials respectively. For the two dimensional Ising model the critical temperature is[1]

$$T_c = \frac{2J}{k_B \ln\left(1+\sqrt{2}\right)} \approx \frac{J}{k_B}\left(2.2691852\ldots\right) \tag{2}$$

where k_B is the Boltzmann constant.

The two dimensional Ising model has been solved exactly, without the use of statistical sampling technique such as Monte Carlo, however, there is currently no analytic solution for the three dimensional case.

2.1.2 Monte Carlo and the Ising model

Using statistical mechanics, given a system with discrete states, we can represent the probability of being in state μ by[1]

$$p_\mu = \frac{e^{-E_\mu/k_B T}}{\sum_\mu e^{-E_\mu/k_B T}} = \frac{1}{Z} e^{-E_\mu/k_B T} \tag{3}$$

where E_μ is the energy of the system in state μ, T is the temperature of the system, and Z is called the partition function. In the Ising model, a state μ would be any combination of spins (up and down) over the lattice. We can use the partition function to calculate the expectation values of many observable quantities in a magnetic material. For example, the expectation value of energy can be expressed as[1]

$$\langle E \rangle = \frac{1}{Z} \sum_\mu E_\mu e^{-E_\mu/k_B T} \tag{4}$$

For small systems it is simple to solve this equation analytically. However, as the system sizes get larger, and more dimensions are added, summing over all possible states becomes a more difficult task. When there are simply too many terms to keep track of we

can use a statistical method which samples only the most important terms of the sum in order to get an estimate of the results without having to calculate explicitly. Monte Carlo methods use this technique.

Monte Carlo methods simulate a random walk through the various states of a system where the probability of being in a certain state follows the equation (3), also called the Boltzmann distribution. Quantities such as the energy, magnetization, and specific heat of a system can be measured directly in a Monte Carlo simulation. For example, using the Hamiltonian from (1), the energy of an Ising model system is[1]

$$E = -J \sum_{\langle i,j \rangle} S_i S_j \tag{5}$$

and the mean magnetization is[1]

$$m = \frac{1}{N} \sum_i S_i \tag{6}$$

where N is the total number of lattice sites.

Monte Carlo methods allow for a simpler way to compute the observable values of magnetic systems when exact solutions become too complicated and computationally time consuming.

2.1.3 The Single Spin Flip Monte Carlo Algorithm[1]

1. Generate a lattice of randomly oriented spins (up or down)
2. Pick one of the lattice sites at random
3. Calculate the change in energy ΔE associated with flipping the spin, using equation (5)
4. Flip the spin with probability $\min\left(1, e^{-\Delta E/k_B T}\right)$
5. return to step 2

The spin is always flipped if it lowers the energy, and flipped with probability $e^{-\Delta E/k_B T}$ otherwise. Steps 2-4 are repeated many times. Each time steps 2-4 are completed is referred to as one iteration.

2.2 Valence Bond Quantum Monte Carlo

Valence bond quantum Monte Carlo is a Monte Carlo method – much like the classical Monte Carlo of section 2.1 – on quantum spins, instead of classical spins, using the valence bond basis to represent the state of the spin system. In a spin-½ system, quantum spins are vectors starting at the origin and ending at any point on the unit sphere. While classical spin can only take the values of ±1, quantum spins can point in any direction, but must have unit length. There are many Monte Carlo methods dealing with quantum spin systems (such as the stochastic series expansion method[4] or the Green's function method[5]) but valence bond quantum Monte Carlo is the simplest of these methods.

2.2.1 The Valence Bond Model

Normally we talk about spins measured with respect to the S^Z basis. Spin ½ particles, such as electrons, can have a value of +½ or –½ in the S^Z basis, corresponding to spin up and spin down respectively.

$$S^Z|\uparrow\rangle = +\frac{1}{2}|\uparrow\rangle \qquad S^Z|\downarrow\rangle = -\frac{1}{2}|\downarrow\rangle \tag{7}$$

The state of N particles or electrons can be expressed as a longer vector of N spins $|\uparrow\downarrow\uparrow\uparrow\cdots\downarrow\downarrow\rangle$ where each arrow refers to the spin of a single particle or electron. The Hamiltonian for a system of N spins is a 2^Nx2^N matrix. Diagonalizing the Hamiltonian matrix gives the energy eigenvalues of the system, but the larger the system, the more difficult the problem of diagonalization becomes.

In the valence bond model, lattice sites are paired into a singlet (valence bond) state of the form:

$$|(a,b)\rangle = \frac{1}{\sqrt{2}}\left(|\uparrow_a\downarrow_b\rangle - |\downarrow_a\uparrow_b\rangle\right) \tag{8}$$

The sites a and b are in a superposition of the states $|\uparrow_a\downarrow_b\rangle$ and $|\downarrow_a\uparrow_b\rangle$, with equal probability of being found in either state. A valence bond lattice covering $|S_k\rangle$ can be represented visually as in figure 2, or mathematically as a product of singlet states as in formula (9).

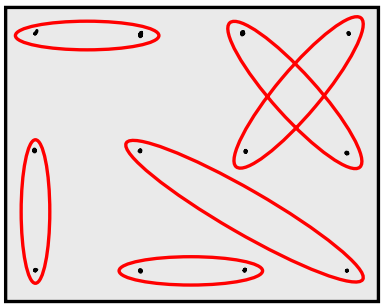


Figure 2 – A sample lattice covering, the red loops represent valence bonds and the black dots represent lattice sites.

$$|S_k\rangle = |(a_1, b_1)(a_2, b_2)\cdots(a_{N/2}, b_{N/2})\rangle \qquad (9)$$

where the bracketed pairs and in form of equation (8).

The valence bond basis can be used to represent any possible state of quantum spins as a weighted sum of valence bond states. The representation is not necessarily unique because the valence bond basis is overcomplete. Representing spin states in terms of the valence bond basis allows for simple calculations of averages, which is useful when using Monte Carlo techniques.

Valence bonds are very stable, with an energy of $-\frac{3}{4}J$ per bond (or per 2 sites). On the other hand the classical antiferromagnetic order (antiparallel spins) has an energy of $-\frac{z}{8}J$ per site, where z is the number of nearest neighbours. It is clear that the valence bond state has a lower energy for a 1-dimensional system (where z=2), and it is known that the ground state of these systems is a valence bond state. However, physicists are still interested in the valence bond model because it is possible that the ground state of 2-dimensional systems (where z=4) may also be in a valence bond state, though it is not yet known.[6]

2.2.2 Ground State Projection

The valence bond basis of lattice coverings is non-orthogonal and overcomplete. This means that some of the basis vectors are redundant, and could be expressed using a linear combination of the other basis vectors. So when we want to express a general state $|\Psi\rangle$ with respect to the valence bond basis, as in (10), the expression is not unique.

$$|\Psi\rangle = \sum_k W_k |S_k\rangle \qquad (10)$$

However, we can always represent a general state $|\Psi\rangle$ as a unique linear combination of its energy eigenvalues and eigenstates.

$$|\Psi\rangle = \sum_n c_n |n\rangle \tag{11}$$

If we then apply the Hamiltonian to this general state we get:

$$H|\Psi\rangle = \sum_n c_n H|n\rangle = \sum_n c_n \lambda_n |n\rangle = c_0\lambda_0|0\rangle + c_1\lambda_1|1\rangle + c_2\lambda_2|2\rangle + \cdots \tag{12}$$

Where λ_n is an energy eigenvalue, $|n\rangle$ is an energy eigenvector, and c_n is the weight of the nth term in the sum. We can then rearrange (12), taking a factor of λ_0 out of each term.

$$H|\Psi\rangle = \lambda_0 \left(c_0|0\rangle + c_1\frac{\lambda_1}{\lambda_0}|1\rangle + c_2\frac{\lambda_2}{\lambda_0}|2\rangle + \cdots \right) \tag{13}$$

Now, when we apply to Hamiltonian to a general state $|\Psi\rangle$ multiple times – say n times – we get the same expression as in (12) and (13), but the eigenvalues are raised to the nth power, giving us:

$$H^n|\Psi\rangle = \lambda_0^n \left(c_0|0\rangle + c_1 \left(\frac{\lambda_1}{\lambda_0}\right)^n |1\rangle + c_2 \left(\frac{\lambda_2}{\lambda_0}\right)^n |2\rangle + \cdots \right) \tag{14}$$

If λ_0 happens to be the eigenvalue with the largest magnitude then the terms $\frac{\lambda_n}{\lambda_0}$ are between -1 and 1, so as they are raised to higher powers, those coefficients will approach zero. Then applying the Hamiltonian to a general state a large number of times will leave us with the eigenstate of the system which corresponds to the largest eigenvalue (in magnitude).

$$H^n|\Psi\rangle \approx c_0\lambda_0^n|0\rangle \tag{15}$$

We can ensure that this state is the ground state by changing the Hamiltonian we use slightly. For example, we could subtract the value of the largest energy from the Hamiltonian so the ground state energy would be the largest in magnitude.

2.2.3 Applying the Heisenberg Hamiltonian to the VB basis

The Hamiltonian acting on a quantum spin system is the Heisenberg Hamiltonian, which takes the form[2]

$$H_{Heis} = -J\sum_{\langle i,j\rangle} \vec{S_i}\cdot\vec{S_j} \tag{16}$$

Where $\vec{S}_i$ and $\vec{S}_j$ are quantum spin operators. Note that for the 1-dimensional case, using the Heisenberg Hamiltonian, the ground state energy has been solved exactly for all system sizes by H. A. Bethe in 1931.[3]

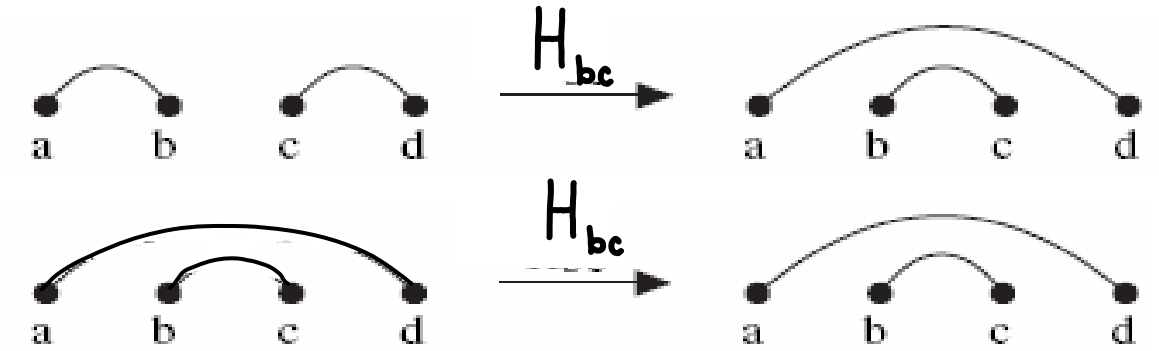
In the case of the valence bond model we modify the Hamiltonian slightly by subtracting ¼. We also set J equal to -1 since we are looking at an antiferromagnetic system. So the Hamiltonian now looks like[2]

$$H = -\sum_{\langle i,j \rangle} \left(\frac{1}{4} - \vec{S}_i \cdot \vec{S}_j \right) = -\sum_{\langle i,j \rangle} H_{ij} \tag{17}$$

We can denote each term more simply, as a bond operator H_{ij} acting on nearest neighbour sites. These bond operators, complicated as they seem, actually have a very simple outcome when applied to a pair of nearest neighbour sites. If the sites are not already joined, the operator joins those sites in a valence bond, joins the two sites that used to be joined to the first two sites, and gives a weight factor of ½ in front of this state, otherwise, if the sites are already joined by a valence bond, the operator does nothing. This can be summarized by (18)[2]

$$\begin{aligned} H_{bc}|\cdots(a,b)\cdots(c,d)\cdots\rangle &= \tfrac{1}{2}|\cdots(a,d)\cdots(c,b)\cdots\rangle \\ H_{bc}|\cdots(a,d)\cdots(c,b)\cdots\rangle &= \ \ |\cdots(a,d)\cdots(c,b)\cdots\rangle \end{aligned} \tag{18}$$

or visually[2]



It is, perhaps, not immediately apparent why we would get these results. However, we can look at the Hamiltonian and lattice coverings mathematically to see what is actually happening.

First let us examine the how $\vec{S}_i \cdot \vec{S}_j$ will act on spins i and j. We are looking at a spin-½ antiferromagnet and we can express $\vec{S}_i \cdot \vec{S}_j$ in another way as:

$$\vec{S}_i \cdot \vec{S}_j = \frac{1}{2}\left[(S_i + S_j)^2 - S_i^2 - S_j^2 \right] \tag{19}$$

Now, any spin operator squared always gives a result of ¾ for a spin-½ system because

$$S^2|\psi\rangle = s(s+1)|\psi\rangle \tag{20}$$

where s, the spin, is equal to ½ in a spin-½ system. So equation (19) becomes:

$$\vec{S}_i \cdot \vec{S}_j = \frac{1}{2}\left(S_i + S_j\right)^2 - \frac{3}{4} \tag{21}$$

But the term $\left(S_i + S_j\right)^2$ acts like a squared spin operator as well, except in this case it acts on both site i and site j. If sites i and j have total spin 1 and the operator gives s(s+1) = 2. Otherwise, they have total spin 0 (as in the valence bond state) and the operator gives s(s+1) = 0. So when we apply $\vec{S}_i \cdot \vec{S}_j$ to a pair of spins we get:

$$\vec{S}_i \cdot \vec{S}_j \ |\psi\rangle = \begin{cases} \left[\frac{1}{2}(2) - \frac{3}{4}\right]|\psi\rangle = \ \ \frac{1}{4}|\psi\rangle & \text{total spin 1} \\ \left[\frac{1}{2}(0) - \frac{3}{4}\right]|\psi\rangle = -\frac{3}{4}|\psi\rangle & \text{total spin 0} \end{cases} \tag{22}$$

This means that when we apply the full bond operator from (17) we get a coefficient of 0 out front if $|\psi\rangle$ is in a triplet state (total spin 1) and a coefficient of 1 if $|\psi\rangle$ is in a singlet state (total spin 0).

$$H_{ij} \ |\psi\rangle = \begin{cases} 0|\psi\rangle & \text{if } |\psi\rangle \text{ has total spin 1} \\ 1|\psi\rangle & \text{if } |\psi\rangle \text{ has total spin 0} \end{cases} \tag{23}$$

Now let's look at what happens when we apply these bond operators to a state composed of valence bonds. First we look at the case where there is already a bond between the two sites in question.

$$H_{bc}|(c,b)\rangle = H_{bc}\frac{1}{\sqrt{2}}\left(|\uparrow_c\downarrow_b\rangle - (|\downarrow_c\uparrow_b\rangle)\right) \tag{24}$$

We start off with a covering including only the lattice sites in which we are interested. The bond operator H_{bc} only affects sites b and c so we need not include other sites. Equation (24) follows directly from (8).

Now from (23) we get

$$H_{bc}|(c,b)\rangle = \frac{1}{\sqrt{2}}\left(|\uparrow_c\downarrow_b\rangle - |\downarrow_c\uparrow_b\rangle\right) = |(c,b)\rangle \tag{25}$$

Since sites c and b are in a singlet state with total spin 0 the state of the system is unchanged, as stated in (18). Now let's look at a state in which the sites in question are not already in a singlet state.

$$
\begin{aligned}
H_{bc}|(a,b)(c,d)\rangle &= H_{bc}\tfrac{1}{\sqrt{2}}\left(|\uparrow_a\downarrow_b\rangle - |\downarrow_a\uparrow_b\rangle\right)\tfrac{1}{\sqrt{2}}\left(|\uparrow_c\downarrow_d\rangle - |\downarrow_c\uparrow_d\rangle\right) \\
&= \tfrac{1}{2}H_{bc}\left(|\uparrow_a\downarrow_d\rangle|\downarrow_b\uparrow_c\rangle - |\uparrow_a\uparrow_d\rangle|\downarrow_b\downarrow_c\rangle - |\downarrow_a\downarrow_d\rangle|\uparrow_b\uparrow_c\rangle + |\downarrow_a\uparrow_d\rangle|\uparrow_b\downarrow_c\rangle\right) \\
&= \tfrac{1}{2}H_{bc}(|\uparrow_a\downarrow_d\rangle\tfrac{1}{\sqrt{2}}[-\tfrac{1}{\sqrt{2}}\left(|\uparrow_b\downarrow_c\rangle - |\downarrow_b\uparrow_c\rangle\right) + \tfrac{1}{\sqrt{2}}\left(|\uparrow_b\downarrow_c\rangle + |\downarrow_b\uparrow_c\rangle\right)] \\
&\qquad -|\uparrow_a\uparrow_d\rangle|\downarrow_b\downarrow_c\rangle - |\downarrow_a\downarrow_d\rangle|\uparrow_b\uparrow_c\rangle \\
&\qquad\qquad +|\downarrow_a\uparrow_d\rangle\tfrac{1}{\sqrt{2}}[\tfrac{1}{\sqrt{2}}\left(|\uparrow_b\downarrow_c\rangle - |\downarrow_b\uparrow_c\rangle\right) + \tfrac{1}{\sqrt{2}}\left(|\uparrow_b\downarrow_c\rangle + |\downarrow_b\uparrow_c\rangle\right)]) \\
&= -\tfrac{1}{4}|\uparrow_a\downarrow_d\rangle\left(|\uparrow_b\downarrow_c\rangle - |\downarrow_b\uparrow_c\rangle\right) + |\downarrow_a\uparrow_d\rangle\left(|\uparrow_b\downarrow_c\rangle - |\downarrow_b\uparrow_c\rangle\right) \\
&= \tfrac{1}{2}\tfrac{1}{\sqrt{2}}(|\uparrow_a\downarrow_d\rangle - |\downarrow_a\uparrow_d\rangle)\tfrac{1}{\sqrt{2}}\left(|\uparrow_c\downarrow_b\rangle - |\downarrow_c\uparrow_b\rangle\right) \\
H_{bc}|(a,b)(c,d)\rangle &= \tfrac{1}{2}|(a,d)(c,b)\rangle
\end{aligned}
\tag{26}
$$

The terms with total spin 1 are annihilated by the bond operator, as in (23), and we are left with a factor of ½ out front of a state with the bonds switched, just as was stated in (18). Note that $|(i,j)\rangle = -|(j,i)\rangle$, which is used between lines 4 and 5 of the above equations. So now that we know the result of acting any bond operator H_{ij} on any state expressed in the valence bond basis, we can use this to project any initial state onto the ground state of the system.

3 Method

3.1 Valence Bond Quantum Monte Carlo

Valence bond quantum Monte Carlo is a method of importance sampled ground state projection (section 2.2.1). Instead of applying the full Hamiltonian operator to an initial state n times, we apply n bond operators (equation (17)) to the initial state. This gives us one term in the sum which would result from applying the Hamiltonian operator n times. We go through combinations of bond operators and find those which result in the largest coefficients, and are therefore, give the most important terms in the resulting sum. Recall, from 2.2.1, that if n is a large number, the resulting sum is just the ground state energy eigenvector and eigenvalue. Using valence bond quantum Monte Carlo, we get this ground state energy eigenvector and eigenvalue expressed in the valence bond basis.

3.1.1 Algorithm

To apply the Hamiltonian operator to an initial state n times:

1. Generate a random initial lattice covering $|S_0\rangle$
2. Generate n random bond operators

3. Apply the n bond operators to $|S_0\rangle$ to get $|S_n\rangle$ using (18). Save the coefficient from the application of the ith bond operator as w_i, the ith weight.
4. Multiply the weights together to get the total weight W, which will be equal to 2 to some negative power.

$$W_{old} = \prod_{i=1}^{n} w_i = \left(\frac{1}{2}\right)^m = 2^{-m}$$

5. Randomly change a few of the n bond operators and repeat steps 3 and 4 to generate a new final covering $|S'_n\rangle$ and a new total weight W_{new}.
6. If $W_{new} \geq W_{old}$ keep changes (set $W_{old} = W_{new}$), otherwise keep the changes with probability W_{old}/W_{new}.
7. Go to step 5

Steps 5 and 6 are repeated until the desired number of iterations is reached. The completion of steps 5 and 6 is referred to as one iteration.

3.1.2 Energy

The valence bond quantum Monte Carlo algorithm is an importance sampling technique that simulates a random walk through the terms in the expansion of the ground state of the system in terms of the valence bond basis. We can calculate the energy from the results of the algorithm.

The expression used to evaluate the ground state energy per site is[2]

$$\frac{E_0}{N} = J \sum_{\langle i,j \rangle} \left(-\frac{1}{2}\langle n_{ij} + 1 \rangle + \frac{1}{4} \right) \tag{27}$$

where N is the total number of lattice sites and n_{ij} is 1 if there is a bond between sites i and j, and 0 otherwise. In other words, we have the sum over nearest neighbours of the expectation value of the number of bonds between sites i and j plus one, along with some constants and factors. To get this expectation value we need to know the number of nearest neighbour bonds in the final coverings $|S_n\rangle$. Since all sites are equivalent in my system – the probability of a nearest neighbour bond is the same for every pair of sites – we can neglect the sum and just multiply the expectation value by the number of sites.

(Actually half the number of sites times the number of nearest neighbours, but it's the same thing for a 1-dimensional chain where there are two nearest neighbours per site.)

Now, if you remember, we subtracted ¼ from the regular Heisenberg Hamiltonian (1) to get our Hamiltonian (17) with the convenient bond operators. This explains the addition of ¼ per nearest neighbour pair in formula (27). We can then use an equivalent expression to find the total energy.

$$E_0 = -J\left(\frac{1}{2}\frac{(\text{total \# NN bonds})}{(\text{\# final lattice coverings})} + \frac{N}{4}\right) \tag{28}$$

where N is the number of lattice sites. The "+ 1" from the expectation value comes out and is just a fact of -½, which then combines with the ¼ to give -¼ in (28). During the VB QMC we sum the total number of nearest neighbour bonds. Then we can divide that total by the total number of final lattice coverings looked at to get the average number of nearest neighbour bonds per covering.

3.1.3 Warm-up Time

In valence bond quantum Monte Carlo, the initial state and initial bond operators generated will not, in general, give a good approximation of the ground state. It takes many Monte Carlo steps before the system reaches a stable equilibrium state where, on average, the energy and other measurable quantities stay the same. The period it takes to get to that stable state is called the warm-up time, and no measurements are taken during that time. After the warm-up time the system is assumed to be in the ground state.

The initial state can be random or not. In our case, if we choose to use a random initial state it will almost always have an energy much higher than the ground state energy, because there will be hardly any nearest neighbour bonds. This leads to a significantly longer warm-up time than if we were to intentionally choose a low energy initial state.

3.2 The Jack-knife Method

Normally to determine the uncertainty in a measurement, such as energy in our case, we can take the standard deviation of all the measurements. However in this case, as in all Monte Carlo simulations, our measurements from one step to another are not statistically independent, so the standard deviation would be an underestimate of the true error.

The Jack-knife method was used to determine the true error in the energy measurements.

The Jack-knife method:

1. Divide the measurements $\{q_1, q_2, q_3, \ldots, q_m\}$ into bins containing k measurements each
2. Find the mean value $\overline{q}_i$ of each bin
3. Calculate the standard deviation of the mean (SDOM) using the mean values from each bin

$$SDOM = \sqrt{\frac{\left(\sum_i \overline{q}_i\right)^2 - \sum_i \left(\overline{q}_i^2\right)}{\#\text{ of bins}}}$$

4. Plot SDOM as a function of bin size k.

The plot should reach a plateau at some larger k value. The value of the SDOM at this point is the true error, and the value of k at this point is the distance between statistically independent measurements.

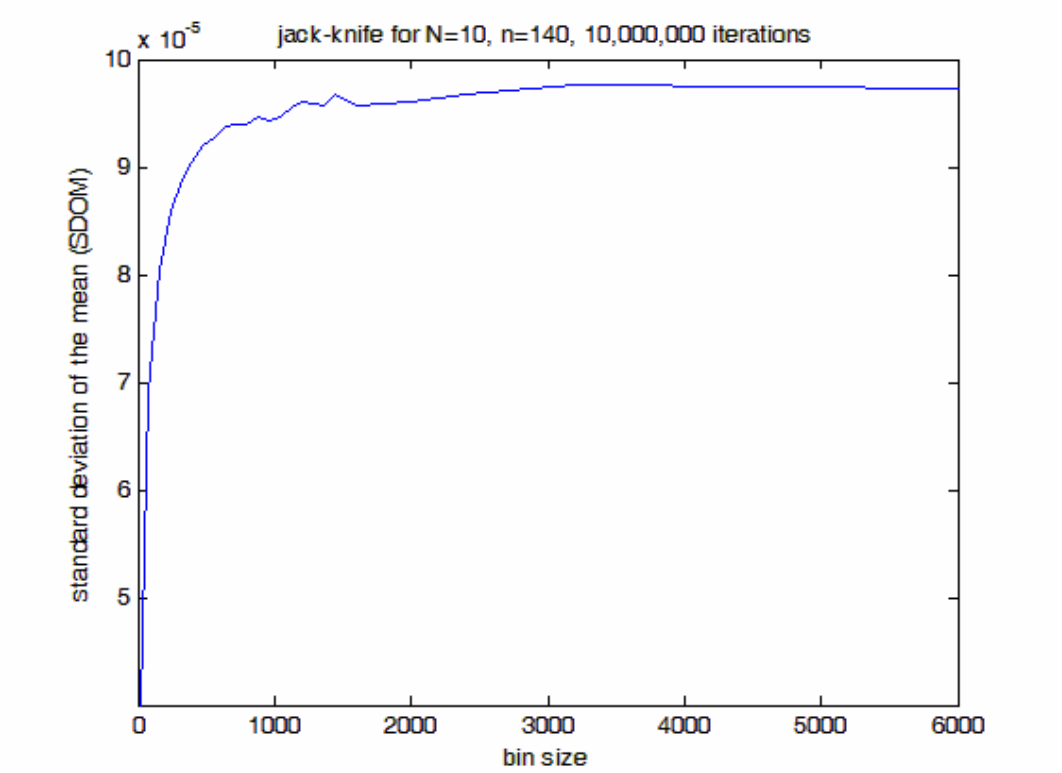


Figure 3 – Plot made using the Jack-knife method for 10 lattice sites, n=140 where n is the number of times the Hamiltonian is applied, and 11,000,000 iterations with a warm-up time of 1,000,000 iterations. The error bar from this figure would be around 0.00095 and you can see that the time between independent measurements is about 1000 iterations.

4 Results

The valence bond quantum Monte Carlo program was used to calculate the ground state energy per site for different numbers of sites. I started with a 10-spin system and calculated the ground state energy using different numbers of iterations to see if the energy converged to the correct value. Then, I varied n (the number of times the Hamiltonian is applied to the system) in order to check the effect on the convergence of the energy. Next, I looked at larger system size, up to a 100-spin system, to see if the algorithm got the correct energy values for other system sizes. These results were compared directly to the Bethe ansatz[3] solution. Last, I studied the effect of r (the number of bond operators changed in a single step) on the ratio of changes accepted and on the error in the calculated energy.

After some problems with long warm-up times, I decided to use a very low energy dimerized state (in which all the bonds are nearest neighbour bonds) instead of a random initial state. The dimerized state has a much larger overlap with the ground state, so this minimizes warm-up time as well as the size of n (Hamiltonian exponent) needed to obtain good results.

Unless otherwise stated, all data is generated using $n = 6N$ and $r = 3$, where n is the number of bond operator, N is the total number of sites, and r is the number of bond operators which are changed at each step.

4.1 Starting Small: A 10-spin system

The 10-spin system is ideal for testing the program; it's small enough that the program will give results quickly and large enough to be interesting. To test the statistics of the program a plot was make of energy per site versus the number of iterations. If the results behave normally the error bars should decrease by ½ when there are 4 times as many iterations. Also, as the number of iterations is increased, the data should get closer to the exact value. The data behaves as expected, which can be seen in figure 4 below.

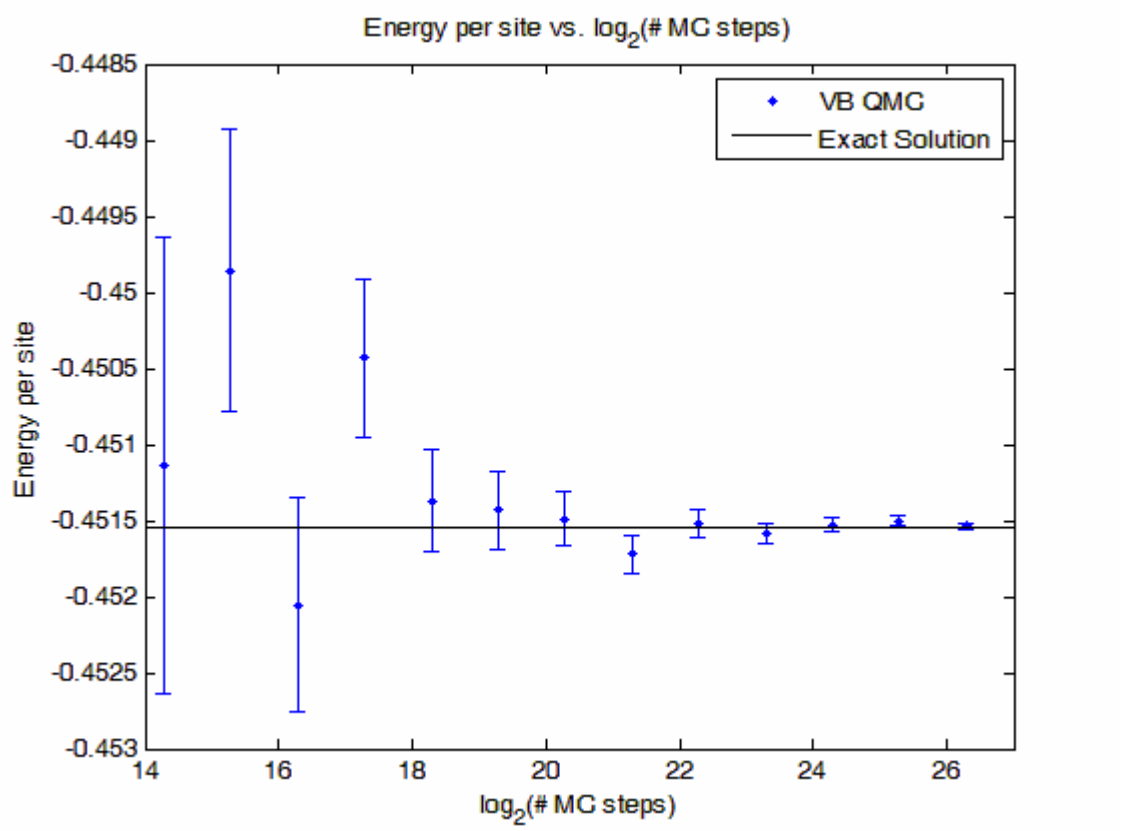


Figure 4 – The energy per site versus the logarithm of the number of iterations. The number of iterations started at 20,000 and was increased by a factor of 2 each time until 81,920,000. There was a warm-up time of 1,000,000 iterations, not included in the total number of iterations.

It may seem strange that 4 of the results do not cross the exact solution, but statistically, only 68% of the points should cross, since the error bar is the standard deviation which only includes 68% of the data. This plot shows that increasing the

number of iterations decreases the error in the data generated as we expect. It also shows that the data converges to the Bethe ansatz[3] solution for a 10-spin system.

Next, we look what happens when n (the power of the Hamiltonian operator) is changed. The simulation ran with different n values ranging from n=N to n=15N for an N=10 spin system. The same number of iterations was used for each value of n.

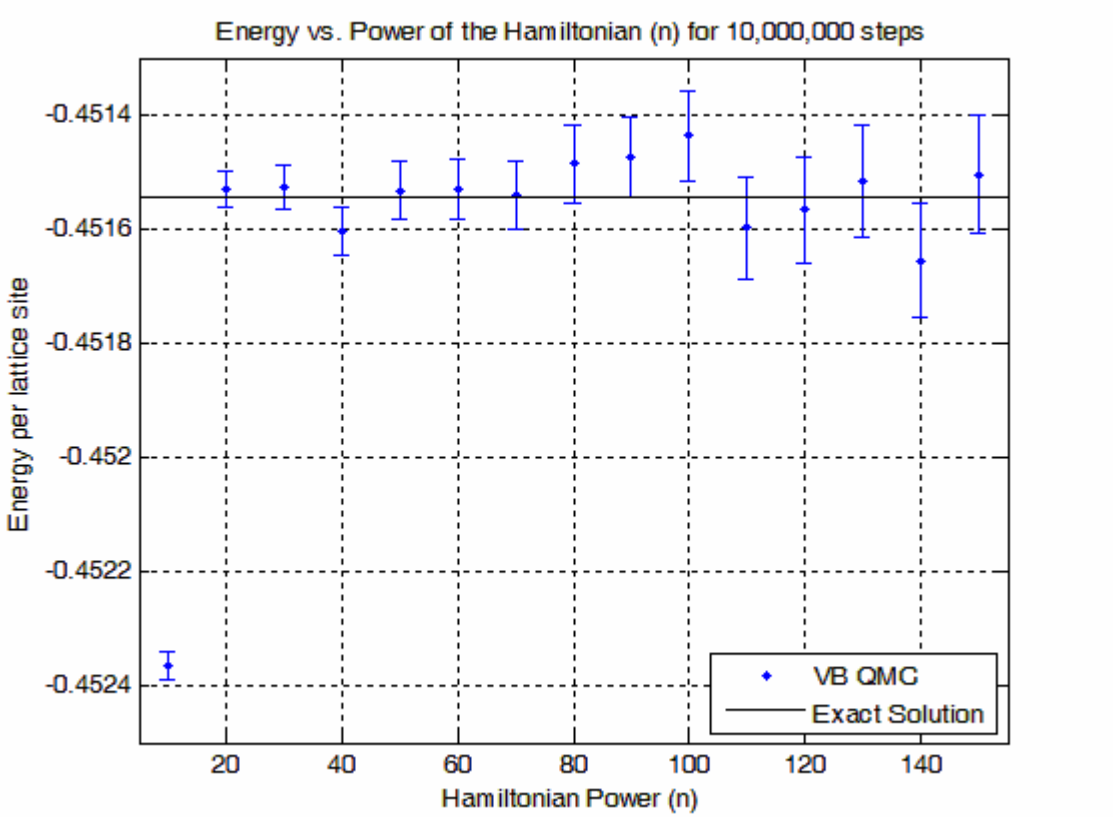


Figure 5 – A plot of the energy per site vs the number of times the Hamiltonian is applied to the initial state using a system with 10-spins. The solid black line is the exact solution (Bethe ansatz) for N=10 sites.

The first point is not even close to the exact solution because n is too small. With only 10 bond operators acting on 10 sites the system is not able to stray as far from the initial state as it needs to get the correct ground state energy. The size of the error bars increases as n increases because changing one of the bond operators that is applied first or nearly first can cause a chain reaction affecting many of the bonds of the system, so from one step to another the energy changes more for a large n than it would for a smaller n. This gives us a larger standard deviation and therefore a larger error bar. The drawback of a large n is that it takes much longer to get the same accuracy you would get with a smaller n. However, with a smaller n it is not certain that the system will have to freedom it needs

to reach the ground state. I worked with n=6N which seemed to run quickly enough without me having to worry about whether the system was actually reaching the ground state.

4.2 Larger System Sizes

Next we look at systems with more than 10-spins, to see if the ground state energy always converges to the Bethe ansatz[3] values. In the following figure the energy per site is plotted against the exact solution for system sizes from N=10 to N=100 spins.

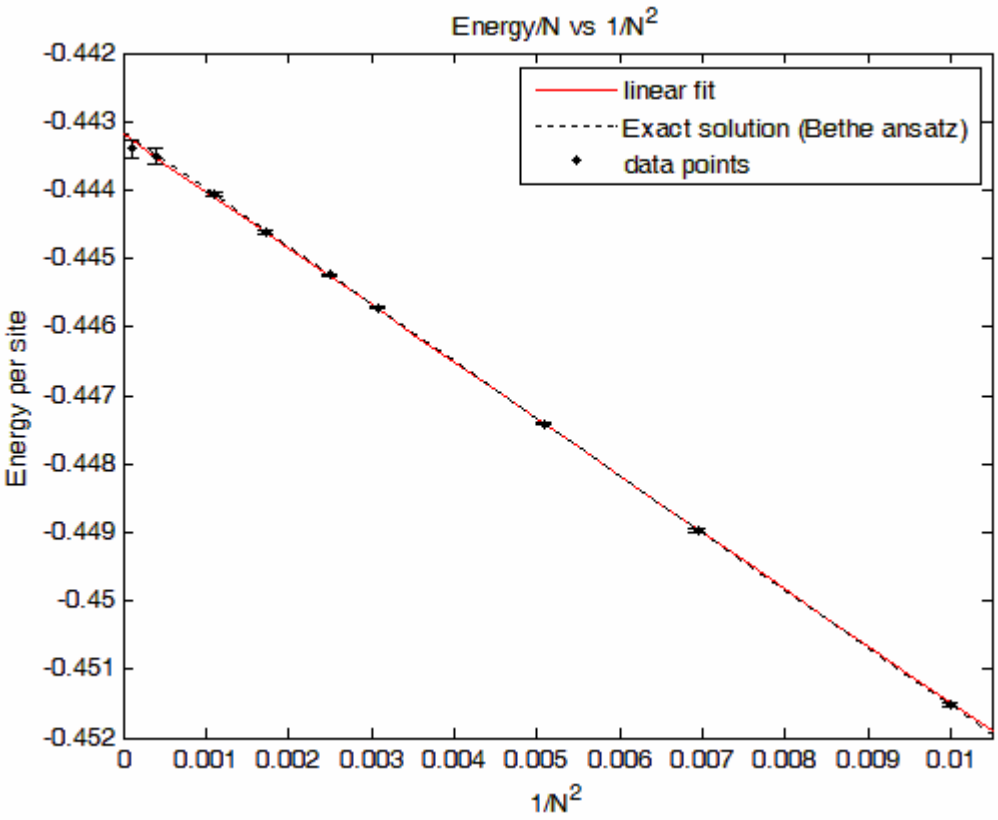


Figure 6 – A plot of energy per site vs $1/N^2$ where N is the number of sites. The dotted black line is the exact solution for all values of N. These data were generated using r=3, n=6N, and various numbers of iterations to obtain the desired accuracy for each point.

The data generated almost exactly overlaps with the Bethe ansatz[3] values. It became difficult to get small error bars for the larger system sizes due to the time it took to produce each point. I could not get a more accurate value for the N=100 system (the first point) even after running a simulation for over 12 hours. My results were consistently lower than the true value when my initial state was the low energy dimerized state, and

consistently above the true value when my initial state was the random high energy state. This suggests to me that the system was not allowed enough warm-up time before the measurements were started.

4.3 Changing r

r is the number of bond operators changed at each step. I have been using r=3 up to this point, but now we will look at what happens when r is changed. First, I took four different values of r and found the acceptance ratios using these values along with different system sizes.

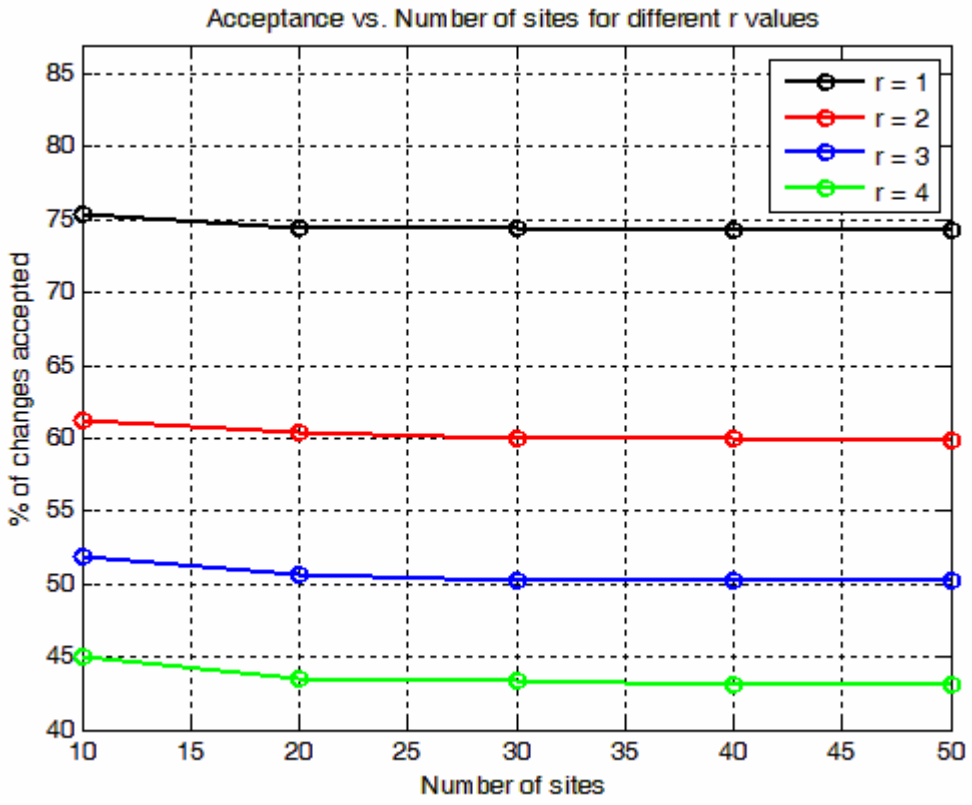


Figure 7 – A plot of % acceptance vs the number of sites for different values of r.

Figure 7 shows that, while the acceptance ratio varies with r, the acceptance ratio stays approximately the same for all system sizes. The smaller systems show the largest change in acceptance ratios for different r. We expect the smaller system sizes to be affected the most, because these correspond to the systems with the smallest number of bond operators. The change in acceptance ratio really depends of the fraction of bond operators

that are being changed, and a 10-spin system has the highest fraction of bond operators changed.

Next, I decided to focus on one system size, using even more values for r. I chose N=30 as the system size, because of the results in figure 7. The acceptance ratio hardly changes with system size after you get past N=20. So the results for N=30 can probably be applied to larger system sizes as well.

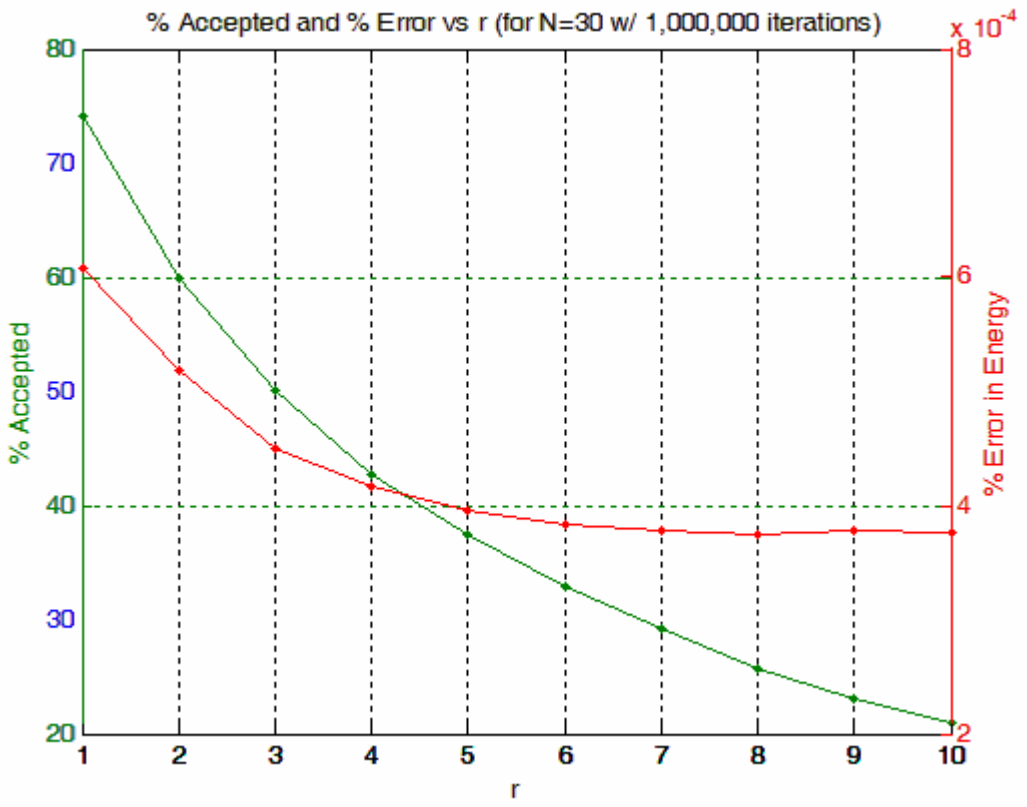


Figure 8 – A plot of the percent of changes accepted vs r and the percent error in the energy vs r. The data were taken for a 30 spin system with 1,000,000 iterations after a warm-up of 100,000 iterations.

This figure shows the percent of changes accepted for different r, where r is the number of bond operators changed per step. It might seem more efficient to use r=1 since it has an acceptance rate of ~75%. However, using a low r gives a larger error because results from one step to another are more statistically dependent when less bond operators are changed. So, for the same total number of iterations you get less information when using a smaller r. Using a larger r also slows down the program. I used r=3, which has an acceptance ratio around 50% with a relatively low % error, for most of my simulations.

5 Conclusions

The ground state energy results from my valence bond quantum Monte Carlo program using various system sizes agreed with the Bethe ansatz[3], that is, the exact solution for a 1-dimensional Heisenberg spin chain. I found that is it much more difficult to get accurate results for large system sizes (>50 spins) because the time associated with running such a simulation is much larger for these large system. It is possible, however, that my program could be made more efficient, or that it could be run on a more powerful computer (I am working on physserv), to get large system results.

My results become more accurate as more iterations are used, exactly as one would expect with Gaussian statistics. I found it is important to use a large enough Hamiltonian power, n, to get accurate results, but using an n that is too large takes much longer to get results. Also, the error for a certain number of iterations increased when a larger n is used.

The acceptance ratio for a certain r (number of bond operators changed per step) is almost independent of system size. If we vary r for a certain system size, the acceptance ratio decreases as r increases. However, the error in the solution also decreases as r increases.

I think I have gained a good general knowledge of Monte Carlo techniques. I started off learning about classical Monte Carlo on the Ising model, including the statistical analysis which can also be applied to quantum Monte Carlo techniques. I then learned about the analogous quantum spin problem. And finally, I learned how Monte Carlo techniques can be applied to the quantum problem.

6 Acknowledgements

I'd like the thank Dr. Sørensen and Dr. Berlinsky for all of their help with my project.

7 References

[1] M. E. J. Newman & G. T. Barkema, *Monte Carlo Methods in Statistical Physics*, Oxford University Press (1999)

[2] A. W. Sandvik, *Ground State Projection of Quantum Spin Systems in the Valence-Bond Basis*, PRL **95**, 207203 (2005)

[3] H. A. Bethe, *Zur Theorie der Metalle, I. Eigenwerte und Eigenfunktionen der linearen Atomkette*,, Z. Phys. 71, 205 (1931)

[4] A. W. Sandvik, *Stochastic series expansion method with operator-loop update*, Phys. Rev. B 59, R14157 (1999)

[5] Y. Song, H. Q. Lin, and A. W. Sandvik, *Green's function theory of the two-dimensional antiferromagnetic Heisenberg model with local magnetic impurities*, J. Phys. Cond. Mat. 12, 5275 (2000)

[6] P. W. Anderson, Science **235**, 1196 (1987)

Appendix A – Main Program

Coded in C++

```
// vb.cpp

#include<iostream>
#include<math.h>
#include"mtrand.h"
using namespace std;

void shuffle(int [][2], int a);
void print_chain(int chain [][2], int a);
void generate_operator(int operater[2]);
int apply_operator(int op0, int op1, int chain[][2], int n);
void change_operators(int operaters[][2], int a);

MTRand drand;
MTRand_int32 irand;

const int L = 100;
const int half_L = L/2;
const int n = L*6;
const int start = 10000000;
const int iterations = start*10;
int chain [half_L][2] = {0};
int operaters[n][2], new_operaters[n][2] = {0};
int operater[2] = {0};
int initial_state[half_L][2] ={0};
int acc = 0, rej =0;
long int w_old=0, w_new=0;
double energy = {0};
int bonds[iterations] = {0};

main()
{
  cout.precision(10);
  cout << endl;

  shuffle(initial_state, half_L);

  //  print_chain(initial_state, half_L);

  for(int i0=0; i0<n; i0++) // generate operators
    {
      generate_operator(operater);
      operaters[i0][0] = operater[0];
      operaters[i0][1] = operater[1];
    }

  for(int j=0; j<half_L; j++)
    {
      chain[j][0] = initial_state[j][0];
      chain[j][1] = initial_state[j][1];
    }
```

```
for(int k=0; k<n; k++)
  {
    w_old  += apply_operator(operaters[k][0],operaters[k][1],chain,
                             half_L);
  }

int q = 0;

FILE * bondss;
bondss = fopen("bonds.txt", "w");

for (int i=0; i<iterations; i++)
  {
    if(i%100000==0){cout<<i<<endl;}

    for(int i7=0; i7<n; i7++)
    {
      new_operaters[i7][0] = operaters[i7][0];
      new_operaters[i7][1] = operaters[i7][1];
    }

    change_operators(new_operaters, n);

    for(int j=0; j<half_L; j++)
    {
      chain[j][0] = initial_state[j][0];
      chain[j][1] = initial_state[j][1];
    }

    for(int k=0; k<n; k++)
    {
      w_new += apply_operator(new_operaters[k][0],
                new_operaters[k][1],chain, half_L);
    }

    if(drand() < pow(2.0,(w_old-w_new)))
    {
      if(i >= start)
        {
            for(int id=0; id<half_L; id++)
              {
                if((chain[id][0] == (chain[id][1]+L+1)%L) |
                      (chain[id][0] == (chain[id][1]+L-1)%L))
                      {bonds[q]+=1;}
              }
          q+=1;
          acc+=1;
        }

      for(int i8=0; i8<n; i8++)
        {
          operaters[i8][0] = new_operaters[i8][0];
          operaters[i8][1] = new_operaters[i8][1];
        }
      w_old = w_new;
    }
    else{if(i>=start)
```

```
            {if(q==0){} else{bonds[q]=bonds[q-1]; q+=1; rej+=1;}}}

      w_new = 0;
    }

  for(int i9=0; i9<q-1; i9++){energy += bonds[i9];}

  energy /= -2*(q-1);
  energy -= L*(.25);
  energy /= L;

  cout << "energy = " << energy << endl;

  cout << q << " energies" << endl;

  cout << 100*double(acc)/q << "% accepted"<< endl;

  for(int j=0; j<half_L; j++)
    {
      chain[j][0] = initial_state[j][0];
      chain[j][1] = initial_state[j][1];
    }

  for(int k=0; k<n; k++)
    {
      w_new += apply_operator(operaters[k][0],operaters[k][1],chain,
                              half_L);
    }

  //  print_chain(chain, half_L);

  int z = 0;
  for(int i=0; i<q; i+=16)
    {
      bonds[z] = bonds[i]+bonds[i+1]+bonds[i+2]+bonds[i+3]+bonds[i+4]
            +bonds[i+5]+bonds[i+6]+bonds[i+7]+bonds[i+8]+bonds[i+9]
            +bonds[i+10]+bonds[i+11]+bonds[i+12]+bonds[i+13]
            +bonds[i+14]+bonds[i+15];
      z +=1;
    }

  for(int i=0; i<z; i++)
    {
      fprintf(bondss, "%i", bonds[i]);
      fprintf(bondss, ",\n ");
    }

  fclose(bondss);

  cout << endl;

  return 0;
}

void shuffle(int chain[][2], int a)
{
  for(int i=0; i<a; i++){chain[i][0] = -1;}
```

```
  int row, column;

  for (int site = 0; site < L; site+=2)
    {
      row = irand() %a;

      while(chain[row][0] != -1)
      {
        row = irand() %a;
      }

      chain[row][0] = site;
      chain[row][1] = site+1;
    }
}

void print_chain(int chain[][2], int a)
{
  for (int i2 = 0; i2 < a; i2++)
    {
      cout << chain[i2][0] << ',' << chain[i2][1] << endl;
    }

  cout << endl;
}

void generate_operator(int operater[2])
{
  operater[0] = irand() %L;
  operater[1] = (operater[0] + ((irand() %2)*2 - 1) + L)%L;

  // cout << '(' << operater[0] << ',' << operater[1] << ") ";
}

int apply_operator(int op0, int op1, int chain[][2], int a)
{
  int index1[2] = {0};
  int index2[2] = {0};

  //find indices of the sites in question
  for (int i4 = 0; i4 < a; i4++)
    {
      for (int j2 = 0; j2 < 2; j2++)
      {
        if(op0 == chain[i4][j2])
          {
            index2[0] = i4;
            index2[1] = j2;
          }

        if(op1 == chain[i4][j2])
          {
            index1[0] = i4;
            index1[1] = j2;
          }
       }
```

```
    }

  if(index1[0] == index2[0]) {// print_chain(chain, half_L);
  return 0;}

  //applying operator and changing chain
  chain[index1[0]][index1[1]] = chain[index2[0]][(index2[1]+1)%2];
  chain[index2[0]][(index2[1]+1)%2] = op1;

  //  print_chain(chain, half_L);

  return 1;
}

void change_operators(int operaters[][2], int a)
{
  int first = irand() %a;
  int second = irand() %a;
  int third = irand() %a;
  //  int fourth = irand() %a;

  while(first == second){second = irand() %a;}
  while(first == third) {third = irand() %a;}
  while(third == second){third = irand() %a;}
  //  while(fourth == first){fourth = irand() %a;}
  //  while(fourth == second){fourth = irand() %a;}
  //  while(fourth == third){fourth = irand() %a;}

  int changings[3] = {first,second,third};//,fourth};

  for(int m=0; m<3; m++)
    {
      int news[2] = {0};

      generate_operator(news);

      while((operaters[changings[m]][0] == news[0]) &&
            (operaters[changings[m]][1] == news[1]))
      {
        generate_operator(news);
      }
      operaters[changings[m]][0] = news[0];
      operaters[changings[m]][1] = news[1];
    }
}
```

Appendix B – Jack-knife Program

Coded in MATLAB

```
%vbjack.m
L=10;
a=16;
m = 0;
for k=[1 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 100 125
      150 200 400 500 600 700 800 900 1000 1125 1250 1500 1750 2000
      2500];
    m = m + 1;
    hep = int32(k);
    bound = idivide(length(bonds),hep,'floor');
    bound = double(bound);
    clear energy
    energy(1:bound) = 0;
    hep = k;
    for j=1:(bound)
        i=(j-1)*hep +1;
        energy(j) = sum(bonds(i:i+hep-1,:)) + hep*a*L;
        energy(j) = energy(j)/(-2*a*hep) + 0.25*L;
        energy(j) = energy(j)/L;
    end

    sdom(m) = sqrt(abs((mean(energy.^2) - mean(energy)^2))/bound);

    clear i
    clear j
end

figure(1)
v=a*[1 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 100 125 150
      200 400 500 600 700 800 900 1000 1125 1250 1500 1750 2000 2500];

plot(v,sdom(1:length(v)))
title('jack-knife for N=10, n=140, 10,000,000 iterations')
xlabel('bin size')
ylabel('standard deviation of the mean (SDOM)')
%axis([0 6000 0.00004 0.0001])

energye = sum(bonds(1:(end-1))) + L*a*(length(bonds)-1);
energye = (energye/(-2*a*(length(bonds)-1)) + 0.25*L)/L
```