Improvements in Cluster Algorithms for Quantum Spin Systems

by

Bernard B. Beard

S.B. Mechanical Engineering, Massachusetts Institute of Technology, June 1979
S.M. Mechanical Engineering, Massachusetts Institute of Technology, September 1982

Submitted to the Department of Physics
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy
in Physics

at the
Massachusetts Institute of Technology
September 1996

Signature of Author _____
Department of Physics
9 August 1996

Certified by _____
Uwe-Jens Wiese
Thesis Supervisor
Assistant Professor, Department of Physics

Accepted by _____
George Koster
Chairman, Departmental Committee on Graduate Students

Improvements in Cluster Algorithms for Quantum Spin Systems

by

Bernard B. Beard

Submitted to the Department of Physics on 9 August 1996
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy
in Physics

Abstract

Loop cluster algorithms provide an efficient implementation of the Monte Carlo technique for evaluating path integrals. The work described here represents improvement in the state of the art in several important areas. The implementation of the loop cluster algorithm in continuous time, as opposed to discretized time, is described in detail for the spin ½ antiferromagnetic Heisenberg model. Implementing the cluster algorithm in continuous time completely eliminates the most severe systematic error in these types of simulations. The loop cluster algorithm in continuous time was used to validate the predictions of chiral perturbation theory in the extreme low-temperature regime ($T = 0.01J$). The numerical results are consistent with the measured properties of the antiferromagnetic insulator precursors of high temperature superconductors.

In addition, generalization of the loop cluster algorithm to higher spin systems is described. The construction of a spin 1 model in discrete time in collaboration with Greven et al. is described, and generalization to the continuous-time case is outlined.

Some discussion of the sign problem in quantum Monte Carlo simulations is also presented.

Thesis Supervisor: Uwe-Jens Wiese
Title: Assistant Professor of Physics

"To realize the unimportance of time is the gate of wisdom."
– Bertrand Russell

Biographical Note

The author, Bernard B. Beard, has previously earned two degrees in Mechanical Engineering from MIT, in 1979 and 1982. While an undergraduate, he was granted membership in the Tau Beta Pi and Pi Tau Sigma Honor Societies. He worked for eleven years in the aerospace industry, designing advanced propulsion systems for military vehicles. In 1991 he was a visiting assistant professor of Mechanical Engineering at Christian Brothers University in Memphis, Tennessee. Since 1992 he has been a graduate student and doctoral candidate at the Center for Theoretical Physics at MIT. He has submitted articles for publication (cond-mat/9202164) and given talks at major conferences on the subject of continuous-time cluster algorithms.

At various times a canoeist, a skydiver, a championship bridge player, a pilot, a scuba diver, a fencer, a musician, a marksman, and a skier, he now makes his home in Memphis, Tennessee – "Home of the Blues."

"In the future, everyone will be famous for fifteen minutes."
– Andy Warhol

Acknowledgments

It has always been my habit to skip the paragraphs of acknowledgments that accompany the texts I read. This character flaw does not lessen in any way the sincerity with which I thank the following individuals.

I thank Dr. John Negele for going out of his way to help me come back to MIT after my years of work in the aerospace industry. He has been a great influence on me and I will always appreciate his attention, honesty, integrity, and insight. Dr. Uwe-Jens Wiese has earned my respect and devotion in countless ways; I sincerely hope to continue collaboration in the future. I also thank Drs. Xiao-Gang Wen, Pawan Kumar, and Ken Johnson for agreeing to sit on my thesis committee and setting aside some of their valuable time to review my work.

I would like to mention two professors whose teaching played a profound role in my development as a physicist. Drs. Nihat Berker and Roman Jackiw uphold the highest standards of teaching and research; my fondest wish is to emulate their example.

I must thank the other members of the BBKLNS study group, with whom and from whom I learned much of the physics that I now know. Andrew Berger, Danielle Kleinberg, Arthur Lue, Steven Nahn, and Neal Spellmeyer have been a great influence on me and I hope we will be lifelong friends. Thanks also to our educational coordinator, Peggy Berkovitz, whose kindness and attention have sustained many a graduate student.

Dr. Ed Farhi and Dr. Sam Gutmann deserve thanks for their infinite patience in responding to my hopelessly naïve and repetitive questions. Drs. Martin Greven and Albert Ferrando have been outstanding collaborators. Thanks also to the members of the CEFPS collaboration, Drs. Caracciolo, Edwards, Ferreira, Pelissetto, and Sokal, for supplying the finite-size scaling formulae that are an integral part of this work. I should also thank P. Grassberger, whom I have never met, for asking Dr. Wiese the question that led to this research.

I would never have been able to return to graduate school had it not been for the support of Dr. Ray Brown, J. Walter Smith, and Ted Langston. Their confidence in my abilities sustains me.

Speaking of sustenance, I have to thank my close friends Craig and Ellen Leckband, Gary and Maureen Hebert, Stephen and Beverly Estes-Smargiassi, and David and Sharon Rapp for feeding me and/or putting me up at various times. With friends like these no man would ever want for good company.

My oldest friends, Dr. Charles Mobbs, Dr. Henry Dones, and Dr. Burton Shaw, all got their doctorates long before I shall. Spurred by the fabulous lifestyles they lead, I humbly seek to

follow in their footsteps. It is not an exaggeration to say that my intellectual development would have been stunted had I not had the privilege of growing up with them.

Odd as it may sound, I also thank Arnold Schwarzenegger. Many years ago I read a thin volume called "Arnold: The Education of a Body-Builder." I learned one important thing from that book: that it is of utmost importance in life to set high goals for oneself and to set out to achieve those goals with all one's energy.

I should also thank the other members of the Keystone School CyberÀlum mailing list. Larry Ledlow, Jack Kent, Elizabeth Boling, Howard Morrow, and Patricia Fahy have all helped me keep my perspective over the years.

I am especially blessed with a supportive family. My sister Edith and her husband Jay Brady, my late father George E. Beard and my steadfast mother Grace Y. Beard, and my parents-in-law W. Bernie Kerr and JoAnne Kerr have all been exceptionally supportive. Thanks also to my aunt- and uncle-in-law Sue and Sherrill Laney.

My wife, Natalie C. Kerr, is the love of my life and is also my best friend. Without her I am nothing, and nothing I accomplish would mean anything.

Finally, thanks to my son Bradford. He gives me hope for the future.

This work was supported in part by a National Science Foundation Graduate Fellowship and in part by funds provided by the U.S. Department of Energy under cooperative research agreement DE-FC02-94ER40818.

Table of Contents

List of Figures

List of Tables

"Everything is a many-body problem"
– Folk saying

Prologue

From the earliest days of quantum mechanics, both classical and quantum spin systems have provided the raw material for developing the techniques of many-body physics. In this sense they have played a role analogous to the Fermat conjecture in abstract algebra: the field provides easily stated problems which lead to delightful theoretical developments.

In recent years two trends have lent urgency and significance to the solution of quantum spin problems. The first is the development of high-temperature superconductors. The precursors of high-T_c superconductors are now understood to be very well-modeled by the antiferromagnetic Heisenberg model (AFHM). Solving the AFHM for a variety of conditions and configurations is now a minor industry; hopes are that this path will eventually lead to a better understanding of these materials. The second is our growing understanding of the relation of the Heisenberg model and its relative, the non-linear σ model, to field theories of the elementary particles. The 2-D non-linear σ model is closely related to 4-dimensional non-Abelian gauge theories. Verifying the sign of the renormalization group β function in the intermediate coupling regime would help establish whether quantum chromodynamics is asymptotically free.

The work described herein advances the state of the art in many-body physics through explorations and analyses of quantum spin systems, mainly meaning the AFHM. In some cases the applicability of the techniques ranges far afield and I have tried to indicate the significance of such results where appropriate. In any case it is my hope that this work will contribute to our understanding not only of the methodology of many-body physics, but also of the research areas named above.

Bernard Beard
Cambridge, Massachusetts
Summer 1996

"The true method of knowledge is experiment"
– William Blake

Chapter 0. Context: Experiment and Theory

Let us first try to put the current research into context. Since the discovery of high-temperature superconductors in 1986 [Bednorz86], there has been an explosion of activity in both experimental and theoretical disciplines. The confluence of these disciplines has resulted in our current rapidly evolving understanding of the class of superconductors based on lamellar copper oxides.

Experimental Situation

Since the discovery of the phenomenon of superconductivity in 1911, physicists have been striving to understand the superconductive phase and to increase the temperature at which superconductivity is encountered. A spectacular breakthrough came with the 1986 discovery by Bednorz and Müller of superconductivity in La-Ba-Cu-O at a temperature near 30K. This discovery was quickly confirmed and followed with subsequent observations of superconductivity in related families of compounds like $La_{2-x}M_xCuO_4$, with M being any of Ba, Sr, or Ca. Wu and collaborators substituted yttrium, a IIIB element like lanthanum, and found a class of compounds related to YBaCuO [Wu87]. The yttrium atom can be replaced by other rare earths such as europium or gadolinium. Thallium- and mercury-based compounds have been formed with critical temperatures T_c as high as 164K.

All of these materials have a common feature: they are built around 2-D layers of CuO_2. The copper oxide layers are interspersed with layers of other atoms. Figure 0-1 shows the arrangement of atoms in a unit cell of the precursor material La_2CuO_4 (which has the same structure as $Sr_2CuO_2Cl_2$). Copper and oxygen atoms form layers that are interspersed with LaO (or SrCl) atomic layers. It is believed by many researchers that the phenomena responsible for superconductivity in the cuprates lie mainly in the CuO_2 layers, and that the rare-earth layers serve merely to stabilize the 2-D structure and to provide charge carriers. Substitution of small amounts of non-magnetic Zn^{2+} atoms for the Cu^{2+} atoms has been shown to destroy the superconductivity. This behavior is inverse to conventional superconductors, where introduction of magnetic elements is detrimental.

In fact, experiments with these materials indicate that they form a new class of superconductors, for which the electron-electron interaction is apparently not mediated by phonons as in conventional superconductors. Measurements of resistivity near the optimal doping level are inconsistent with the phonon-mediated Bardeen-Cooper-Schrieffer (BCS) model [Bardeen57]. Other evidence – for example the temperature dependence of the Hall coefficient – also suggests that the electron-electron interaction in the cuprates is novel.

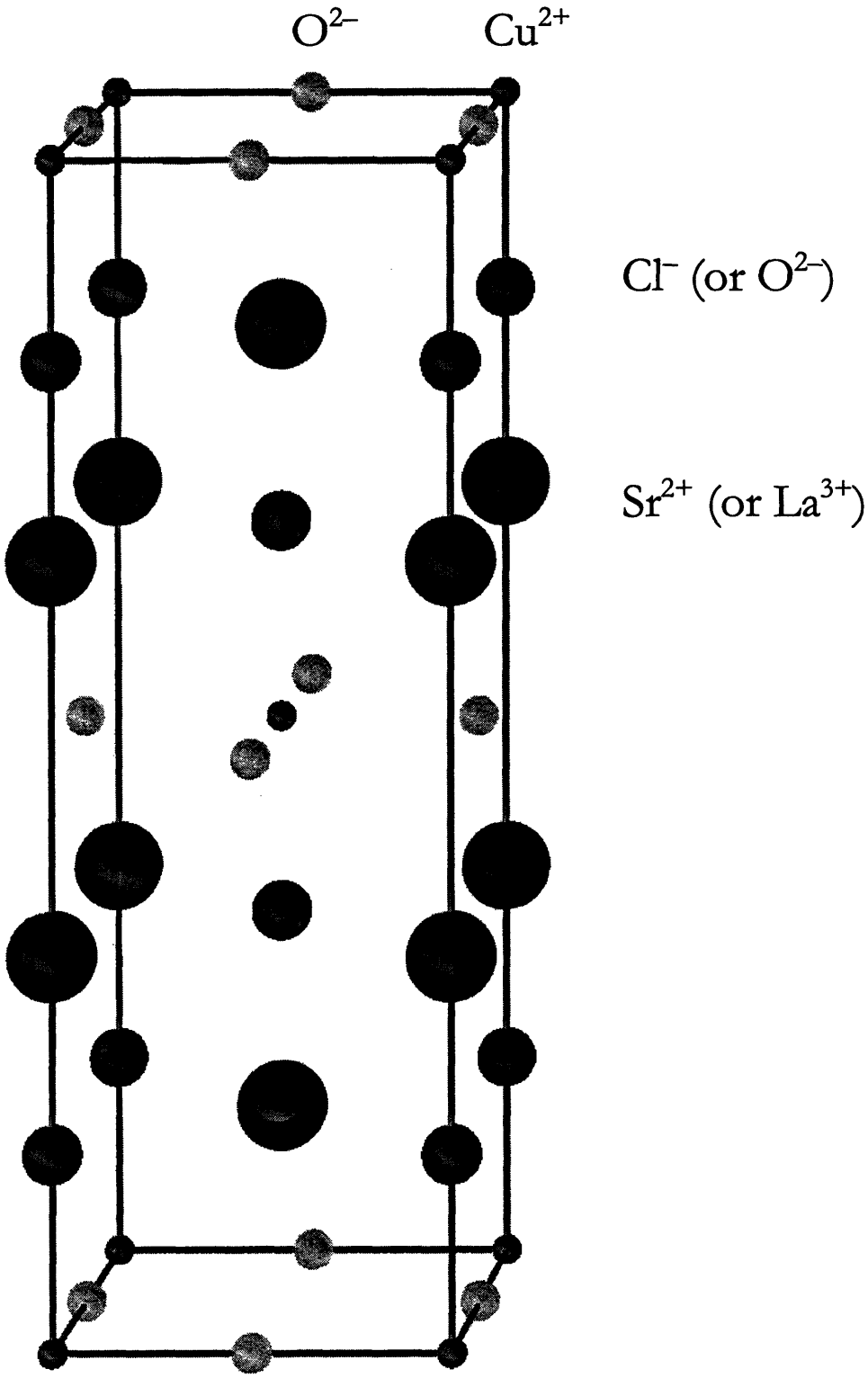


Figure 0-1. The atomic arrangement of the unit cell of $Sr_2CuO_2Cl_2$. The arrangement of atoms in La_2CuO_4 is identical, modulo the substitution of La for Sr and of additional oxygen atoms for the chlorine atoms. Copper and oxygen form 2-D sheets interleaved with layers of SrCl. These high T_c superconductor precursors are the ultimate target of the current investigation into the antiferromagnetic Heisenberg model.

It became apparent soon after the discovery of the cuprate superconductors that their undoped precursors are well-described by the antiferromagnetic Heisenberg model (AFHM) [Shirane87]. Since the cuprates are the first known empirical realization of the 2-D square lattice spin ½ AFHM, there has been a resurgence of interest in this model.

An array of experimental techniques are used to explore the properties of the cuprate superconductors. One of the most versatile techniques is neutron scattering. The de Broglie wavelength of thermal neutrons is comparable to the interatomic distances of condensed matter, on the order of an angstrom. As a consequence, interference effects deliver substantial information about the magnetic and nuclear structure of condensed matter. In addition, the energy of thermal neutrons (300 K or 1/40 eV) is on the order of many of the excitations of interest to the condensed matter physicist.

Within the context of various theoretical models, numerous measurements can be made of the parameters that govern the behavior of the cuprates. Analysis of Raman spectroscopy data gives a value for the exchange constant J that appears in the Heisenberg Hamiltonian. Correlation lengths and structure factors can be measured more or less directly with neutron scattering. Spin-wave velocity is taken from the magnon dispersion relation, also measured with neutron scattering. In particular M. Greven has shown good agreement among experimental data, the predictions of chiral perturbation theory, and numerical simulations for the correlation length in $Sr_2CuO_2Cl_2$ [Greven95b]. Greven shows that the relation between the spin stiffness and the coupling constant J, derived from the confluence of chiral perturbation theory and numerical simulations, is also consistent with experimental data from neutron scattering.

Theoretical Models

A number of different theoretical avenues for modeling the electronic degrees of freedom in the cuprate superconductors have been explored. The Heisenberg model, already referred to above, will be discussed in detail in the next chapter. The AFHM is itself a special case of the Hubbard model at half-filling and in the strong coupling limit. The Hubbard Hamiltonian is

$$H = \sum_{\langle ij \rangle} \sum_{\sigma} T_{ij} \left(c_{i\sigma}^{\dagger} c_{j\sigma} + h.c. \right) + \tfrac{1}{2} I \sum_{i,\sigma} n_{i\sigma} n_{i,-\sigma}$$

where $c_{i\sigma}^{\dagger}$ and $c_{i\sigma}$ are creation and annihilation operators and $n_{i\sigma} = c_{i\sigma}^{\dagger} c_{i\sigma}$ is the number operator for electrons at site i in spin state σ, and $\langle ij \rangle$ indicates that sites i and j are nearest neighbors [Hubbard63a]. This model in principle should be extended to multiple bands to address the specifics of electronic structure of the cuprate superconductors (the Cu $3d^9$, O $2p^6$, and Cu $3d^{10}$ bands are involved). Since this model is very difficult to work with, theorists often turn to simplifications, such as the t-J model and the one-band Hubbard model above. The t-J Hamiltonian is

$$H = -t \sum_{\langle ij \rangle} \sum_{\sigma} \left(c_{i\sigma}^{\dagger} \left(1 - n_{i,-\sigma}\right)\left(1 - n_{j,-\sigma}\right) c_{j\sigma} + h.c. \right) + J \sum_{\langle ij \rangle} \left(\vec{S}_i \cdot \vec{S}_j - \tfrac{1}{4} n_i n_j \right)$$

In this model the oxygen sites have been eliminated and double occupancy is forbidden.

The difficulty in dealing with the high T_c superconductors is specifically that the electron wave functions are neither strictly localized (as in a strong-coupling limit) nor strictly itinerant (as in a weak-coupling limit). The difficulty of modeling this intermediate regime makes it imperative that we exploit the sophisticated techniques that have been recently developed for dealing with these quantum systems.

Chiral Perturbation Theory

Let us return to the context of the Heisenberg model, which we use to understand the antiferromagnetic precursors of the high T_c superconductors.

A particularly powerful tool for analyzing quantum systems is chiral perturbation theory (CPT). Originally developed and applied to pion dynamics in the context of spontaneously broken symmetries [Weinberg66, Weinberg68], the tools of CPT have been found to be quite general, as is often the case with quantum field theories.

CPT starts with the most general effective local action which respects all the symmetries of a system. The interaction of massless excitations that arise from spontaneous symmetry breaking is strongly constrained by symmetry principles. The method of effective Lagrangians gives a systematic way to calculate higher-order corrections. One derives then a theory with some number of undetermined parameters but with otherwise strong predictions for dynamics.

In the ground state of the 2-D AFHM the staggered magnetization develops an expectation value, and hence the $O(3)$ spin rotational symmetry gets spontaneously broken down to $O(2)$. (This result has been shown rigorously for spin $S \geq 1$ and is inferred to be the case for $S = 1/2$). The low energy excitations of the systems are spin-waves (called magnons) which are the Goldstone bosons of the spontaneously broken $O(3)$ symmetry. In accordance with Goldstone's theorem, we get two magnons since the dimension of the quotient group $O(3)/O(2) = S_2$ is 2. The resulting field theory is 2+1 dimensional, with fields Ω living in the 2-sphere and hence obeying the constraint $\Omega^2 = 1$.

In the case of the antiferromagnetic Heisenberg model, CPT predicts magnon dynamics at low temperatures with a handful of parameters as unknown constants. The most important of these parameters for our purposes are the staggered magnetization M_s, the spin-wave velocity $\hbar c$, and the spin stiffness ρ_s. The staggered magnetization is analogous to $\langle \bar{\psi}\psi \rangle$ in more familiar settings and the spin stiffness plays the same role as the pion decay constant F_π^2. The spin-wave velocity does not play precisely the role of the speed of light, since there is no analog of Lorentz invariance, but it can still be used to relate time (that is, inverse temperature) to spatial distance.

To be explicit, the action is written in terms of a staggered symmetry-breaking field H and coupling constants F and Σ

$$S_{eff} = \int_0^{L_\tau} d\tau \int d^2x \left[\tfrac{1}{2} F^2 \partial_\mu \Omega \partial_\mu \Omega - \Sigma (H\Omega) + \ldots \right]$$

$$\left(\Omega^2 = 1\right)$$

We suppress terms with more derivatives or higher powers of H. The effective action has infinitely many coupling constants concealed in those higher terms. However, it can be shown that for small H the coupling constants in the action above suffice to determine the leading and next-to-leading order low-energy properties of the system. Note that F and Σ are the Goldstone boson decay constant and magnetization for infinite volume, zero temperature, and zero staggered field, and that they are subsequently identified with $\rho_s = F^2$ and $M_s = \Sigma$ [Hasenfratz93].

It is well-known that these parameters govern the leading order of the calculation of observables; Hasenfratz and Leutwyler showed that no new parameters enter into the next-leading order correction as well [Hasenfratz90a]. This fact facilitates the high-precision numerical studies that are discussed and extended in this dissertation.

A Tale of Two Regimes

Using these tools, Hasenfratz and Niedermayer analyzed the behavior of susceptibilities and internal energy in the 2-D AFHM [Hasenfratz91,Hasenfratz93]. Most helpfully, they made predictions for finite-volume effects on the observables. It turns out that the behavior of these observables is readily analyzed with CPT for certain volume-temperature geometries. We quote results here for the so-called "cubical" regime, for which $\beta \equiv L/\hbar c$, and for the "cylindrical" regime, where $\beta >> L/\hbar c$. Hasenfratz and Niedermayer use the parameter $\ell^3 = \beta \hbar c / L$ to discuss the various regimes.

The **cubical** regime results are expressed in terms of shape factors that are functions of ℓ for $\ell \approx 1$. We find for the uniform and staggered susceptibilities

$$\chi_u = \frac{2}{3} \frac{\rho_s}{(\hbar c)^2} \left\{ 1 + \frac{1}{3} \frac{\hbar c}{\rho_s L \ell} \tilde{\beta}_1(\ell) + \frac{1}{3} \left(\frac{\hbar c}{\rho_s L \ell} \right)^2 \left(\tilde{\beta}_2(\ell) - \frac{1}{3} \tilde{\beta}_1(\ell)^2 - 6\psi(\ell) \right) + \ldots \right\}$$

and

$$\chi_s = \frac{1}{3} M_s^2 \frac{L^2}{T} \left\{ 1 + 2 \frac{\hbar c}{\rho_s L \ell} \beta_1(\ell) + \left(\frac{\hbar c}{\rho_s L \ell} \right)^2 \left(\beta_1(\ell)^2 + 3\beta_2(\ell) \right) + \ldots \right\}$$

where the shape factors are defined by the set of relations

$$\beta_n(\ell) = \left(-\frac{1}{4\pi} \right)^n \left[\alpha_n(\ell) + \frac{3}{n(2n-3)} \right],$$

$$\tilde{\beta}_1(\ell) = \frac{1}{\ell} \frac{d}{d\ell} \left(\ell^2 \beta_1(\ell) \right),$$

$$\tilde{\beta}_2(\ell) = \frac{1}{\ell^3}\frac{d}{d\ell}\left(\ell^4 \beta_2(\ell)\right),$$
$$\alpha_p(\ell) = \hat{\alpha}_{p-3/2}(\ell) + \hat{\alpha}_{-p}\left(\ell^{-1}\right),$$
$$\hat{\alpha}_r(\ell) = \int_0^1 dt\, t^{r-1}\left[S\left(\frac{1}{\ell^2 t}\right)^2 S\left(\frac{\ell^4}{t}\right) - 1\right],$$
$$S(x) = \sum_{n=-\infty}^{\infty} \exp\left(-\pi n^2 x\right).$$

Also, $\psi(\ell)$ is another shape factor; for $\ell = 1$ we have

$$\psi(1) = -\frac{1}{3}\beta_1(1)^2 - \frac{1}{3}\beta_2(1) = -0.020529$$

In the **cylindrical** regime the spectrum is that of an $O(3)$ rotator,

$$E_j = j(j+1)\frac{(\hbar c)^2}{2\rho_s L^2}\left[1 - \frac{\hbar c}{\rho_s L}\frac{3.900265}{4\pi} + O\left(\frac{1}{L^2}\right)\right].$$

The leading term in the uniform susceptibility is

$$\chi_u \xrightarrow[T\to 0]{} \frac{6}{L^2 T}\exp\left(-\frac{(\hbar c)^2}{\rho_s L^2 T}\right).$$

The staggered susceptibility is predicted to approach the temperature-independent form

$$\chi_s \xrightarrow[T\to 0]{} \frac{2M_s^2 \rho_s}{(\hbar c)^2} L^4\left[1 + 3\frac{\hbar c}{\rho_s L}\frac{3.900265}{4\pi} + O\left(\frac{1}{L^2}\right)\right].$$

These forms are verified in the numerical simulations discussed in Chapter 2.

The CH_2N_2 Formula

In an earlier paper, Hasenfratz and Niedermayer extended the work of Chakravarty, Halperin, and Nelson (CHN) in the analysis of the correlation length in the 2-D AFHM [Chakravarty88, Hasenfratz91]. The CHN analysis used perturbative renormalization group arguments to conclude

$$\xi \propto \frac{\hbar c}{2\pi\rho_s}\exp(2\pi\rho_s\beta)\left[1 + O(1/2\pi\rho_s\beta)\right]$$

where $\beta \equiv 1/T$ is the inverse temperature. Hasenfratz, Maggiore, and Niedermayer derived an exact expression for the correlation length and obtained

$$\xi = \frac{e}{8\Lambda_{\overline{MS}}}$$

where $\Lambda_{\overline{MS}}$ is the renormalization group invariant scale defined in the $\overline{MS}$ renormalization scheme [Hasenfratz90b]. They found

$$\Lambda_{\overline{MS}} = \mu \frac{2\pi}{g(\mu)} \exp(-2\pi/g(\mu)) \left[1 - \frac{g(\mu)}{8\pi} + O\left(g(\mu)^2\right) \right]$$

The term $g(\mu)/8\pi$ is the result of a three-loop calculation. The above expression is valid in the asymptotic-scaling region in which the coupling $g(\mu)$ runs with the scale μ. Using chiral perturbation theory together with renormalization group methods, Hasenfratz and Niedermayer expressed the running coupling as

$$\frac{1}{g(T)} = \frac{\rho_s}{T} - \frac{3T}{16\pi^2 \rho_s} + O\left(\frac{T^2}{\rho_s^2}\right)$$

where the scale μ was chosen to be the temperature T. Combining these results gives

$$\xi(\beta) = \frac{e}{8} \frac{\hbar c}{2\pi\rho_s} \exp(2\pi\rho_s\beta) \left[1 - \tfrac{1}{2}(1/2\pi\rho_s\beta) + O\left((1/2\pi\rho_s\beta)^2\right) \right]$$

which we refer to as the CH_2N_2 formula. This relation should hold at low temperatures. When the correlation length is correctly described by the CH_2N_2 formula, we say that it scales asymptotically. It is known that asymptotic scaling sets in only at very large correlation lengths for the 2-D $O(3)$ model ($\xi \approx 10^5 a$ for $\approx 5\%$ agreement). One of the challenges that are addressed in the present work is to quantify the degree to which the CH_2N_2 formula accurately describes the correlation length for the 2-D spin ½ AFHM at low temperatures. This issue is presented in Chapter 5.

Numerical Simulation as a third discipline

Armed with this experimental and theoretical background, we are now ready to begin exploration of numerical approaches to dealing with the Heisenberg model and its cousins.

"The past is the present, isn't it? It's the future, too."
– Eugene O'Neill

Chapter 1. The Loop Cluster Algorithm

This chapter is intended as an introduction to quantum spin systems and the conventional techniques for analyzing them. The knowledgeable reader may find little new material here but an acquaintance with the terminology and concepts summarized below is vital to understanding the rest of this work.

Quantum physics on a lattice

It is best to begin immediately with the Heisenberg model. Werner Heisenberg introduced this model into the literature in 1928 [Heisenberg28]. The intention is to capture some of the important aspects of the quantum mechanical many-body problem in condensed matter, specifically on a spatial lattice. Heisenberg proposed the following Hamiltonian:

$$H = J\sum_{\langle xy\rangle} \vec{S}_x \cdot \vec{S}_y$$

where the x and y are sites in a lattice, $\vec{S}_x$ is a spin operator for site x, and the notation $\langle xy\rangle$ indicates that the sum is to be taken over nearest neighbors only. It is important to understand that this Hamiltonian is an operator in the Hilbert space of lattice states and not just a classical functional in the spirit of, say, the Ising model. The motivation for this model is that the wave functions for the valence electrons are localized on lattice sites and have significant overlap only with their neighbors. [Feynman72] has a cogent discussion of the interpretation of the coupling constant J as an exchange integral.

The Heisenberg model in this form can be applied to a variety of physical situations. The lattice can be a 1-dimensional spin chain, a 2-dimensional square lattice, a 2-dimensional hexagonal lattice, or any of many other geometries. Variations abound – the coupling can be made asymmetrical in a variety of ways, second- or even third- nearest neighbor couplings can be included, the coupling J can be made to vary with spatial location, and so forth. Sometimes a coupling to an external magnetic field is included. There is a close relation between spin-spin coupling models and various fermion interaction models (see, e.g. the review article by De Raedt and Lagendijk [DeRaedt85]). In particular, the spin ½ AFHM is equivalent to the strong coupling limit of the Hubbard model at half filling. As mentioned in the prologue, there is also a mapping between the Heisenberg model and the non-linear σ-model (see the article by Ian Affleck in [Halley88]).

A priori we have not specified the magnitude of the spin on each lattice site, though physically realized models have small spins (e.g., $1/2$ and 1). In general when I refer to the Heisenberg model in this work I shall mean the spin ½ Heisenberg model, unless explicitly stated otherwise. In this case the operators $\vec{S}_x$ are $\hbar/2$ times the Pauli spin matrices

associated with the site x. Also, we will usually work on a finite lattice of side length L, measured in units of lattice spacing a, and we will usually work in units where $a = 1$.

In general, if the coupling constant J is negative, then the ground state can be expected to have nearest-neighbor spins aligned; because this ordered state models the alignment of spins in a ferromagnet, we say that the model is ferromagnetic. If, on the other hand, J is positive, the system may be frustrated or have staggered order; in any case, we refer to the model as antiferromagnetic for positive J. I shall often use the abbreviation AFHM to refer to the antiferromagnetic Heisenberg model.

Now it can be seen that on the one hand this model is a significant simplification of the many-body interaction in a crystal lattice – in lieu of every site interacting with every other site, we have reduced the interactions to a handful for each site – but we are still left with a very complicated system. Even in the case of a 1-D spin chain, site 1 interacts with site 2, which in turn interacts with site 3, and so on. And the operator that connects 2 and 3 does not commute with the operator that connects 1 and 2. So, despite our simplification, we seem to be left with a model which is very difficult to handle analytically.

As is well-known, early in the history of this model, Hans Bethe wrote down an explicit, exact solution to the model in the special case of the 1-D spin chain, in what has become known as the Bethe ansatz [Bethe31]. The ansatz in itself has been the source of solutions of other difficult problems in field theory (see, e.g. [Andrei83]) but we do not propose to discuss those efforts here.

The Trotter-Suzuki Method

Instead we focus on a class of solutions that are based on the Trotter formula. In 1976 Masuo Suzuki pointed out that the Trotter formula can be used to analyze a variety of models like the AFHM [Suzuki76a].

The basic idea is to subdivide the Hamiltonian into complementary sets of terms, each of which is composed of simple terms that commute with each other. The common example is to analyze the AFHM on a finite 1-D spin chain. We assume that there are N sites, where N is even and we impose periodic boundary conditions. In this case, we can divide the Hamiltonian into even- and odd-indexed terms, i.e.

$$H = H_1 + H_2 = J\sum_m \vec{S}_{2m} \cdot \vec{S}_{2m+1} + J\sum_m \vec{S}_{2m+1} \cdot \vec{S}_{2m+2}$$

Then we can write

$$
\begin{aligned}
Z &= \mathrm{Tr}\left(\exp(-\beta H)\right) \\
&= \lim_{\varepsilon\to 0} \sum_{n} \langle n | \left(\exp\left(-\varepsilon\beta(H_1 + H_2)\right)\right)^N | n\rangle \text{ where } N\varepsilon = 1 \\
&= \lim_{\varepsilon\to 0} \sum_{\substack{\{n_k\} \\ n_N = n_0}} \prod_{k=0}^{N-1} \langle n_k | \exp\left(-\varepsilon\beta(H_1 + H_2)\right) | n_{k+1}\rangle \text{ inserting complete sets of states} \\
&= \lim_{\varepsilon\to 0} \sum_{\substack{\{n_k\} \\ n_N = n_0}} \prod_{k,k'=0}^{N-1} \langle n_k | \exp(-\varepsilon\beta H_1) | n_{k'}\rangle \langle n_{k'} | \exp(-\varepsilon\beta H_2) | n_{k+1}\rangle \text{ applying the Trotter formula}
\end{aligned}
$$

The Trotter formula allows us to split up the exponential, since the error in doing so is $O(\varepsilon^2)$. The complete sets of states are eigenstates $|\pm 1/2\rangle$ of the spin operator. The extra dimension is an inverse temperature or Euclidean time dimension. We exploit these facts to turn the d-dimensional quantum spin problem into a (d+1)-dimensional classical spin problem. For the 1-D spin chain, we wind up with 2N slices in the time direction.

For more complicated lattices, we can usually find a decomposition of the Hamiltonian which enables this decomposition. For example, for the 2-D square lattice, we can divide the nearest-neighbor links on the spatial lattice into 4 sets – explicitly, we split the Hamiltonian up as

$$H = \sum_{\substack{\mu=1,2 \\ k=1,2}} H_{\mu,k}$$

$$H_{\mu,k} = J \sum_{x\in X_{\mu,k}} \vec{S}_x \cdot \vec{S}_{x+\hat{\mu}}$$

with the sets defined by

$$
\begin{aligned}
X_{1,1} &= \{x | x = (2m, n)\} \\
X_{2,1} &= \{x | x = (m, 2n)\} \\
X_{1,2} &= \{x | x = (2m+1, n)\} \\
X_{2,2} &= \{x | x = (m, 2n+1)\}
\end{aligned}
$$

In this case we get 4N time slices in the Trotter-Suzuki decomposition. The time slices are indexed so that the sets $X_{\mu,k}$ are active sequentially; we write

$$
\begin{aligned}
T_{1,1} &= \{t | t \equiv 0 (\mathrm{mod}\, 4)\} \\
T_{2,1} &= \{t | t \equiv 1 (\mathrm{mod}\, 4)\} \\
T_{1,2} &= \{t | t \equiv 2 (\mathrm{mod}\, 4)\} \\
T_{2,2} &= \{t | t \equiv 3 (\mathrm{mod}\, 4)\}
\end{aligned}
$$

This spatial decomposition is pictured in Figure 1-1.

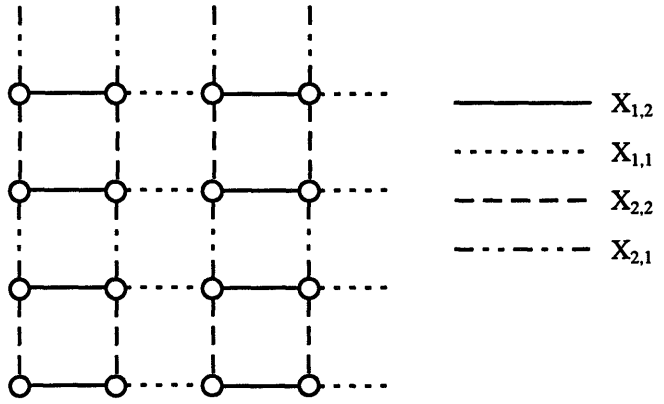


Figure 1-1. Decomposition of nearest-neighbor Hamiltonian into four parts.

The final step is to connect the factors in the transfer matrices with an action. We write

$$Z = \prod_{x,t} \sum_{s(x,t)=\pm 1} \exp(-S).$$

The action is

$$S = \sum_{\substack{\mu=1,2 \\ k=1,2}} \sum_{\substack{x \in X_{\mu,k} \\ t \in T_{\mu,k}}} S\left[s(x,t), s(x+\hat{\mu},t), s(x,t+1), s(x+\hat{\mu},t+1)\right]$$

and the plaquette Boltzmann factors are given by the 4 by 4 matrix

$$\exp\left(-S[s_1,s_2,s_3,s_4]\right) = \left\langle s_1 s_2 \left| \exp\left(-\varepsilon\beta J \vec{S}_x \cdot \vec{S}_{x+\hat{\mu}}\right) \right| s_3 s_4 \right\rangle$$

The matrix in the last equation is called a "transfer matrix" because it connects spin states at time t with those at $t+1$. For the Heisenberg model, the transfer matrix is

$$\left\langle n_k \left| \exp(-\varepsilon\beta H) \right| n_{k+1} \right\rangle = \exp\left(-\tfrac{1}{4}\varepsilon\beta J\right) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}\left(1+e^{\varepsilon\beta J}\right) & \frac{1}{2}\left(1-e^{\varepsilon\beta J}\right) & 0 \\ 0 & \frac{1}{2}\left(1-e^{\varepsilon\beta J}\right) & \frac{1}{2}\left(1+e^{\varepsilon\beta J}\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}_{k,k+1} \leftrightarrow \begin{pmatrix} ++ \\ +- \\ -+ \\ -- \end{pmatrix} \textit{basis}$$

Note there are 6 non-zero elements, corresponding to those plaquettes which preserve the 3-component of spin.

The number N (or sometimes the related quantity $2dN$) is called the "Trotter number". In the conventional numerical treatment, some finite N is selected and the summation is implemented on a computer. There is an error associated with using a finite N, since the Trotter formula is only valid in the limit $N \to \infty$. This error is called the Trotter error and is generally the most severe systematic error in such simulations.

Of course, for any reasonably sized system and finite Trotter number, sampling the complete sets of states exhaustively is out of the question, since the cardinality of the global basis set is 2 raised to the N_t power, where $N_t = 2^d N(L/a)^d$ is the number of lattice sites. Instead we turn to Monte Carlo methods to evaluate the summation.

From the outset it is clear that a brute force Monte Carlo algorithm will not be practicable. That is, randomly changing spin states at random sites, with a Metropolis accept/reject step, will have an overwhelming rejection rate, simply because most configurations in the basis set are "illegal". These illegal configurations have zero weight (i.e. infinite action) because they have one or more plaquettes that violate the conservation of spin component in the 3-direction.

The only way to construct a successful Monte Carlo algorithm for these types of systems is to build the conservation laws into the mapping step that takes one legal configuration to another. Within this constraint, there are a variety of schemes that combine simple local moves with occasional large steps that wrap around in Euclidean time. [DeRaedt85] discusses early attempts to develop effective Monte Carlo routines for fermionic and spin models. A common theme is that closed loops of spins are flipped, in order to respect the conservation of the 3-component of spin. A recent example is the extensive study of Makivic´ and Ding of the 2-D AFHM by such conventional Monte Carlo methods [Makivic91].

Algorithms based mostly on local changes suffer from long autocorrelation times and a phenomenon known as "critical slowing down" – the divergence of the autocorrelation time near the time continuum limit.

Evolution of the loop cluster algorithm

In 1987 Swendsen and Wang introduced a cluster algorithm that showed marked improvement in the dynamical critical exponent that characterizes critical slowing down [Swendsen87]. Their work, which used the Potts Hamiltonian as a model, showed how one could implement an algorithm that generated large clusters of spins and yet used only local information at each step in loop-building. Locally the algorithm is probabilistic in a way that satisfies detailed balance.

Subsequent efforts applied this technique to a variety of models, such as vertex models [Evertz93] and the 1-D and 2-D Heisenberg model [Wiese92 and Wiese94].

Here we discuss the discrete-time cluster algorithm for the 1-D and 2-D spin ½ AFHM. Recall the transfer matrix for the Heisenberg model, presented above. Note that the off-diagonal elements are negative if $J > 0$ – and hence cannot be interpreted as Boltzmann factors. In the special case of bipartite lattices – that is, lattices on which we can define a stagger factor, such as the 1-D chain and 2-D square lattice – we can eliminate this minus sign with a unitary change of basis. The "trick" is to rotate every other spin by 180 degrees about the z axis, effectively exchanging $x \leftrightarrow -x$ and $y \leftrightarrow -y$. The unitary matrix is

$\exp(-i\pi\sigma_3/2)\otimes 1 = \mathrm{diag}(i,-i,i,-i)$, which can be easily verified to change the sign of the off-diagonal elements but leave the others alone. This leaves us with the transfer matrix

$$M = \exp(-\tfrac{1}{4}\varepsilon\beta J)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}\left(e^{\varepsilon\beta J}+1\right) & \frac{1}{2}\left(e^{\varepsilon\beta J}-1\right) & 0 \\ 0 & \frac{1}{2}\left(e^{\varepsilon\beta J}-1\right) & \frac{1}{2}\left(e^{\varepsilon\beta J}+1\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Discrete-time cluster-building rules for the Heisenberg Hamiltonian

If we start from a legal (finite action) configuration, the cluster algorithm by design will sample the configuration space in proportion to the Boltzmann weight of each configuration. The basic idea is to build closed loops of spin sites and then flip all the spins on the loop. The rules for building loops are designed to provide ergodicity and detailed balance (cf. [Evertz93]).

To build a loop, it is necessary to decide how it will flow through each of the different types of plaquettes that have finite action. In general the flow rules will depend on the particular Hamiltonian being investigated. For the spin ½ AFHM, there are 6 different legal plaquettes, but because of spin-up/down symmetry, this reduces to 3 types of flow patterns. Table 1-1 displays the 3 types of legal plaquettes, together with flow patterns that have been determined to provide detailed balance for the AFHM.

The foremost rule is that the flow will move forward in time from a particular site if the spin at that site is currently "up", and backward in time if the current spin is "down" – unless it moves to an adjacent site at constant time. This rule guarantees that the resulting loop will always locally satisfy the constraint on the 3-component of spin. The loop is constructed piecewise until it closes. The loop-building rules ensure that the loops always close.

The first pattern in Table 1-1, with "like" spins on all four corners of the plaquette, forces the flow to proceed in the time direction. The second pattern, with alternating spins, forces the flow to move to the adjacent spin site at constant time. The third pattern, with "unlike" spins at adjacent spin sites, is probabilistic. The first time a loop encounters such a plaquette, the flow is sent in the time-direction with probability p. If the flow happens to revisit the plaquette, it is forced to conform to the flow direction that was chosen at the initial visit. These 3 types of plaquettes are referred to as "forced continuation," "forced transition," and "optional decay" respectively.

Detailed balance is verified by examining a plaquette type, flipping the spins on one flow line, and seeing which plaquette type results. The product of the transition probability and the Boltzmann weight must be the same in each direction. For example, the forced continuation plaquette has Boltzmann weight $\exp(-\varepsilon\beta J/4)$; flipping the spins on one flow line (with probability 1) will send it to an optional decay type plaquette with Boltzmann weight $\exp(-\varepsilon\beta J/4)(1+\exp(\varepsilon\beta J))/2$. The transition probability in one direction is unity,

and in the other direction is $p = 2/(1+\exp(\varepsilon\beta J))$, and obviously this case satisfies detailed balance. It is easily verified that the other transitions also satisfy detailed balance.

Table 1-1. Summary of Plaquette Flow Rules. Solid circles indicate spin-up sites; open circles are spin-down. The time direction is horizontal. Flow rules for inverse plaquettes are analogous.

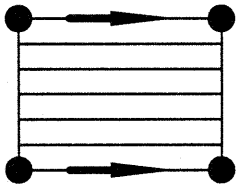
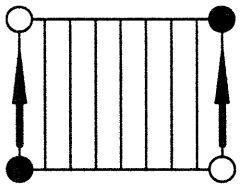
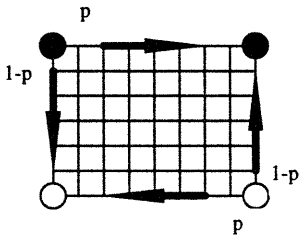
	Discrete Time
Forced continuation	
Forced transition	
Optional decay, $p = 2/(1+\exp(\varepsilon\beta J))$ and $\lambda = \lim_{\varepsilon\to 0}\frac{1-p}{\varepsilon\beta} = J/2$	

Figure 1-2 shows a typical situation in the building of a loop in 1 + 1 dimensions with discrete time. Here we show the neighborhood of site i, for a number of time slices. Filled circles indicate spin up sites; open circles, spin down. The hatched rectangles are the plaquettes resulting from the Trotter-Suzuki decomposition. In this figure, as in Table 1-1, the hatching is indicative of the flow rules associated with each type of plaquette: horizontal for forced continuation, vertical for forced transition, and cross-hatched for optional decay. A loop is shown entering this diagram at t_1. It is forced to proceed forward in time for two plaquettes, then encounters a series of optional decay plaquettes. It is shown succumbing to the temptation to move to site $i-1$ at time t_7. Note that if it had reached time t_{10}, it would have been forced to move to site $i+1$.

The cluster algorithm connects configurations with different total 3-component of spin because loops can wrap around in the time direction. This helps to provide ergodicity for the algorithm. In fact it is possible to prove ergodicity of the loop cluster algorithm for both 1-D chains and 2-D square lattices with even numbers of spin sites. In outline, we define ± 1

stagger factors for alternating spin sites, and then we define a spin "domain" as a connected region of spin-time space in which the stagger times the spin state has a particular sign. Loops are confined to domains but can fill them. Any two spin configurations differ by a finite number of domains and can be connected by a finite number of loop flips.

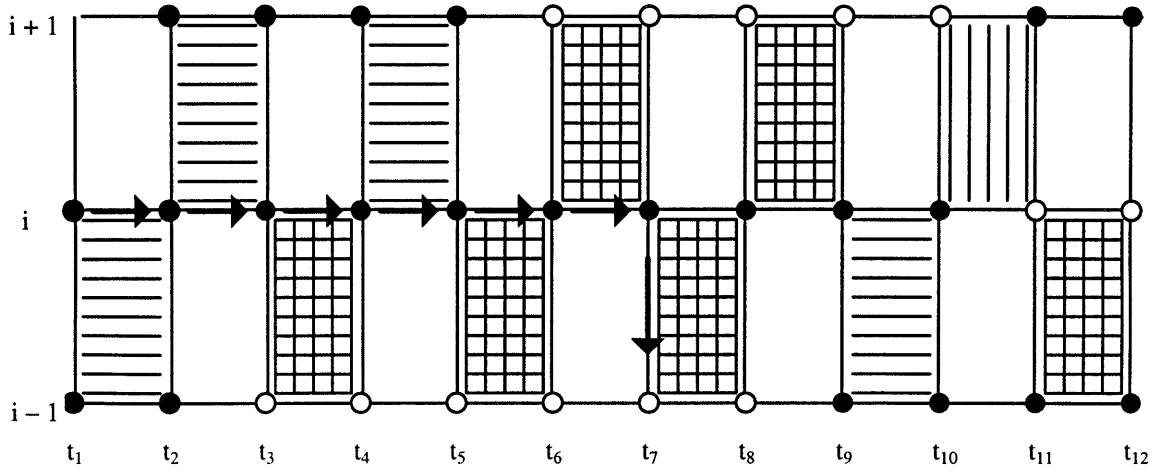


Figure 1-2. Typical situation in the building of a path in discrete time and 1 spatial dimension. Time is in the horizontal direction; the immediate neighborhood of spin site i is depicted in the vertical direction. Solid circles denote spin-up sites; open circles denote spin down.

This in essence is the discrete-time cluster algorithm. We use a probabilistic local algorithm to build closed loops of spins in such a way as to guarantee detailed balance and ergodicity. The loops are flipped, a measurement is taken, and then the process is repeated. Generally we repeat for roughly 10^5 to 10^7 measurements. The discussion of how we implement observables is deferred to Chapter 3. First we discuss how we take this algorithm to the continuum limit.

"What then is time? If no one asks me,
I know what it is. If I wish to explain it
to him who asks, I do not know."
– St. Augustine

Chapter 2. Generalization to continuous time

At this point a certain insight comes into play. It is easily understood that since the state on each spin site is discrete, either +1/2 or -1/2, any representation of the time-dependence of the state necessarily involves only finite segments with discrete transitions between the two states. In the Trotter-Suzuki implementation, this means that a given spin site is in a state for a certain number of time slices, then switches to the other state for some number of time slices, and so forth. We recognize that increasing granularity in time (i.e. increasing Trotter number) leads to a continuum picture with finite segments and sporadic discrete transitions between states, as in Figure 2-1.

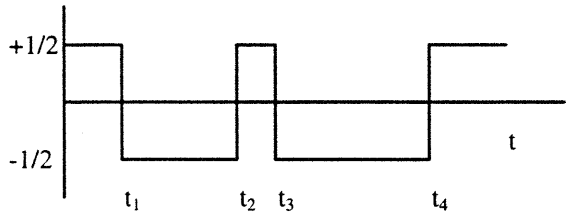


Figure 2-1. Variation of spin state versus time in the continuous-time picture.

Dr. Wiese credits P. Grassberger with suggesting that the transition times can be manipulated directly, rather than relying on the storage of spin state information at each time slice. As we shall see, this is a very fruitful idea. Indeed, the implementation of the loop cluster algorithm in continuous time completely eliminates the most severe systematic error afflicting these kinds of simulations – that of finite Trotter number.

The Farhi-Gutmann formula

Before we discuss the translation of the loop cluster algorithm to continuous time, however, it is worthwhile to review a very closely related procedure, developed recently by Ed Farhi and Sam Gutmann [Farhi92].

In their paper, Farhi and Gutmann lay out a procedure for calculating matrix elements of the time evolution operator for Hamiltonians operating on a discrete basis. This discussion follows parts of their paper closely. The operator that evolves states in time is defined by

$$|\psi(t)\rangle = \hat{U}(t,t_0)|\psi(t_0)\rangle$$

and is related to the Hamiltonian through

$$i\frac{d}{dt}\hat{U}(t,t_0) = \hat{H}(t)\hat{U}(t,t_0).$$

We also define

$$\hat{U}(t,t_0) = 1.$$

A countable orthonormal basis $|i\rangle$ is selected. Farhi and Gutmann show that a functional integral expression can be written for the matrix element

$$U_{ji}(t,t_0) = \langle j|\hat{U}(t,t_0)|i\rangle.$$

They use the following notation for the modulus and phase of the matrix elements of the Hamiltonian (note the factor of i in the definition):

$$\rho_{jk}\exp(i\theta_{jk}) \equiv -i\langle j|\hat{H}|k\rangle$$

where ρ_{jk} is real and nonnegative and θ_{jk} is real. Define the sum of the off-diagonal moduli

$$\rho_k \equiv \sum_{j\neq k}\rho_{jk}$$

and the total weight

$$W_k \equiv \langle k|\hat{H}|k\rangle + i\rho_k.$$

Then the matrix element above is identically equal to the functional integral

$$F_{ji}(t,t_0) \equiv \int [dq]_{ji}\exp\left(-i\int_{t_0}^{t} ds W_{q(s)}(s)\right)\prod_{l\to k}\exp(i\theta_{kl}(\cdot))$$

where the following interpretation is given to this symbology. A path is defined as a mapping $q(s)$ from the interval $[t_0,t]$ to the (countable) basis space. A measure for the summation over these paths is defined by rules for generating a random path. The basic rule is this: if a path has the value k at time s, then with a probability per unit time $\rho_{lk}(s)$ the path will switch from k to l. The measure is defined by taking all paths that begin at i and end at j. The weight $W_{q(s)}(s)$ has the meaning given above. The term $\prod_{l\to k}\exp(i\theta_{kl}(\cdot))$ is read as a prescription to multiply by $\exp(i\theta_{kl}(s))$ if the path switches from l to k at time s.

We reproduce Figure 1 from [Farhi92] as Figure 2-2 below to clarify this discussion. The characteristic that defines this picture is that the paths comprise finite segments with sporadic discrete transitions.

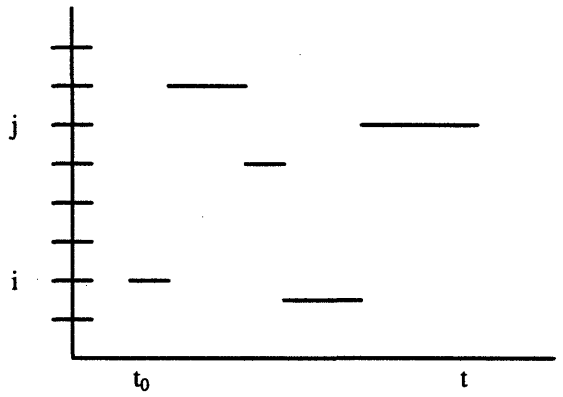


Figure 2-2. A typical path from *i* to *j*.

[Farhi92] contains a proof of the proposition that this functional integral is equivalent to the matrix element of the time evolution operator, as well as a number of enlightening examples. For our purposes it is enough to reiterate the central mechanism behind this path integral: the notion that the path changes state according to some probability per unit time that is determined by the matrix elements of the Hamiltonian in question.

Now the relation of the Farhi-Gutmann procedure to the current work is as follows: first, we specialize to Euclidean time. Then, instead of computing individual matrix elements, we are concerned with the partition function, that is, a trace over all the basis states. Lastly, we seek a practical, efficient algorithm for approximating the trace. It is this last condition that forces us to part company with the Farhi-Gutmann procedure, for it is, as written, an impractical procedure for numerical analysis because so many configurations are discarded for each configuration accepted.

Adding granularity to the discrete-time picture

So, returning to our loop cluster algorithm, we proceed first in a heuristic fashion. We imagine the lattice in space-time as derived by the Trotter-Suzuki method. We imagine a given lattice configuration, that is, a certain specification of spin states for every site in the spin time lattice. Then we add granularity in time, that is, we insert time slices between the existing slices. We find that the state of any given spin site as a function of time ultimately resembles Figure 2-1. The time variation of the spin state at a given site comprises segments in a fixed state for a finite time, with sporadic discrete transitions between states. We build loops in the same way: loop movement from one spin site to a neighbor is sporadic and of measure zero, and most of the (now infinitesimal) plaquettes are continuations in time.

Continuous-time cluster-building rules

Table 2-1 resembles Table 1-1 with the addition of a column for the continuum-time flow rules. Solid lines represent continuously spin up states; dotted lines, spin down. The flow rule for the forced continuation plaquettes has, in the continuum limit, become a rule that the flow cannot move to a neighbor that is in the same spin state. It is the forced transition plaquettes which represent the discontinuous change in spin state; we call the resulting

jumps "bonds" because the changes in state at a given site are always accompanied by complementary changes in a neighbor. The continuum limit of the flow rule for the forced transition plaquettes is the rule that, if the flow reaches a bond, it must move along a bond to the neighbor and reverse direction (because the flow direction is dictated by the spin state).

The rule for the optional decay plaquettes, which is probabilistic, becomes a rule that the flow will jump to a neighbor in an opposite state with a fixed probability per unit time. The resulting exponential distribution of segment lengths is identical to the familiar distribution of lifetime of a radioactive nucleus, which retrospectively justifies our use of the term "decay" to describe this plaquette type. The decay constant is simply

$$\lambda = \lim_{\varepsilon \to 0} (1 - p)/\varepsilon\beta = J/2.$$

It is not coincidental that this decay constant is also the off-diagonal transfer-matrix element of the AFHM Hamiltonian. As we suggested above, the fact that the Hamiltonian is the generator of time translations is closely related to the structure of algorithms that represent path integrals in continuous time.

Table 2-1. Summary of Plaquette Flow Rules. Solid circles and solid lines indicate spin-up sites; open circles and dotted lines are spin-down. The time direction is horizontal. Flow rules for inverse plaquettes are analogous.

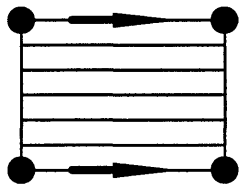
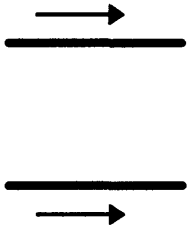
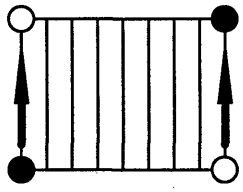
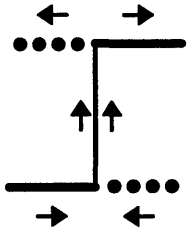
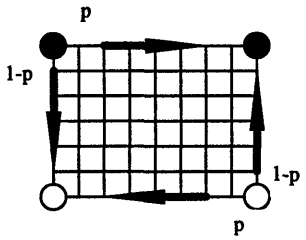
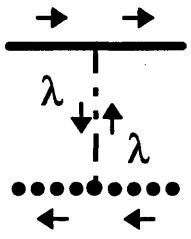
	Discrete Time	Continuous Time
Forced continuation		
Forced transition		
Optional decay, $p = 2/(1+\exp(\varepsilon\beta J))$ and $\lambda = \lim_{\varepsilon \to 0} \frac{1-p}{\varepsilon\beta} = J/2$		

Figure 2-3 shows a typical situation in the building of a loop in 1 + 1 dimensions with continuous time. Again we show the neighborhood of site i. Solid lines indicate spin up sites; dotted lines, spin down. A loop is shown entering the diagram at t_1. The probability per unit time that it will move to a neighboring site is proportional to the number of neighbors that are of opposite spin. The total decay constant λ is shown varying with time in the graph in Figure 2-3. For example, between t_2 and t_3 there are no available "decay channels" and the flow is forced to move forward in time in that interval. In the situation shown here, the flow survives up to some point between times t_4 and t_5, when it decays to site $i-1$. Note that if the flow had reached time t_6, it would have been forced to traverse the bond and move to site $i+1$. Path building proceeds in this way until the loop closes.

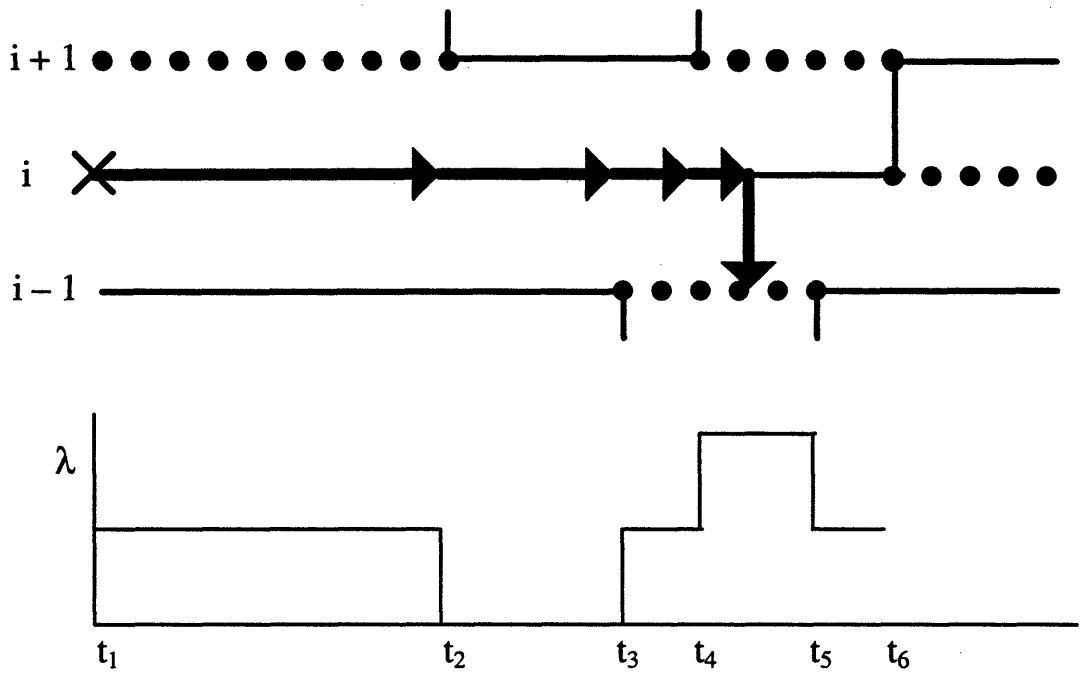


Figure 2-3. Typical situation in the building of a path in continuous time and 1 spatial dimension. Solid lines denote spin up sites in continuous time; dotted lines, spin down. λ is the probability per unit time that the path will jump to an adjacent site, and is proportional to the number of neighboring sites that have opposite spin.

The Code LATTMP

The continuous-time cluster algorithm was implemented in a Fortran code, designated LATTMP. A recent version of the code is provided in Appendix A.

From a computational standpoint, the essential difference between the discrete-time and continuous-time implementation is that the former requires us to store spin state information for each of the points of the space-time lattice, while the latter requires us only to store the transition times for each spin site (plus an extra bit to record the state at $t=0$). As outlined above, the time evolution of the path is also handled differently; instead of a

point-by-point crawl through the lattice, we race through time because we know the decay times are exponentially distributed. In the code developed here, all the transition times were maintained in a large array. A cyclic double-linked-list storage technique was implemented to ensure that the overhead associated with inserting and deleting time values was minimized.

A significant advantage of the loop cluster algorithm is that it allows for the implementation of improved estimators. We found that the improved estimators for susceptibility, staggered susceptibility, and internal energy density all have easily determined continuum limits, although the internal energy density estimator was found to suffer from poor numerical behavior.

The intention of the code was to provide an extensible "neighbor structure." However, it turns out that a naïve extension of the techniques that work for the 1-D chain and 2-D square lattices does not work for, say, the 2-D hexagonal lattice. See Chapter 6 for more discussion of this limitation.

From the outset we adhered to a design goal that the algorithm would work with floating point numbers up to machine precision, that is, with no arbitrary "tolerances" specified. The result is a code that is highly portable and is credibly faithful to the spirit of the continuous-time cluster algorithm.

In the judgment of this author, this kind of code is moderately more difficult to write and to debug than a conventional plaquette-oriented code. Working with continuous time within the constraints of finite-precision computer arithmetic introduces a number of subtleties related to the fact that certain phenomena cannot be relied upon to be of measure zero. Accommodation to this fact was implemented in a number of specific tests for equality conditions that would never occur in a hypothetical infinite-precision machine. A careful reading of the code in Appendix A will reveal these "seams" but we elide such details here.

Thermalization studies

Previous studies of the discrete-time cluster algorithm showed that the lattice configuration "thermalizes" rapidly, that is, given a random start, in relatively few steps the configuration becomes indistinguishable from any configuration far into a simulation.

We explored this phenomenon in somewhat more detail than previous studies. Results are outlined in Appendix D. Based in part on the results of the thermalization study, our "standard" run consisted of a thermalization period of 1,000 configurations followed by 100,000 measurements.

Validation

The output of the code was checked against known exact results for 2-spin and 4-spin systems; against previous discrete-time cluster simulations of the 1-D ferromagnetic and both 1-D and 2-D antiferromagnetic Heisenberg models [Wiese92, Wiese 94]; and against selected parameters in the literature for other methods of modeling the AFHM. In all cases the results were consistent with or better than previous studies.

Table 2-2. Comparison of discrete- and continuous-time cluster algorithm uniform susceptibility results for 1-D AFHM. Discrete-time results are from [Wiese92]. The predictions of the continuous-time code agree quite well with the apparent continuum limit of the predictions of the discrete-time code.

β	L	2N(disc.)	χ_u (disc.)	χ_u (cont.)
1	32	32	0.1354(4)	0.1362(4)
1	32	64	0.1362(4)	"
1	32	128	0.1374(4)	"
1	32	256	0.1362(4)	"
2	128	16	0.1435(6)	0.1456(6)
4	128	32	0.1251(5)	0.1257(7)
8	128	64	0.1173(5)	0.1181(8)
16	128	16	0.0714(6)	0.1103(9)
16	128	32	0.0985(6)	"
16	128	64	0.1087(7)	"
16	128	128	0.1120(7)	"

Tables 2-2 and 2-3 compare the simulation to previous results for the 1-D and 2-D AFHM [Wiese92, Wiese94]. Figure 2-4 shows the internal energy density versus temperature for various methods, including the continuous-time cluster algorithm [Wiese94, Bonner64, Lyklema82, Cullen83, Wiese92].

The continuum-time method completely eliminates the most severe systematic error in this type of calculation. It also obviates the need to conduct the several runs of successively finer time-granularity needed for extrapolation to the continuum limit, thus eliminating a costly dimension in the simulation procedure. It should also be noted that this method directly solves the problem (first demonstrated by Barma and Shastry [Barma78]) that taking the Trotter number $N \to \infty$ and $\beta \to \infty$ limits in the wrong order gives erroneous ground-state observables.

Storage and Time Requirements

It is somewhat tricky to compare the discrete-time cluster algorithm and continuous-time cluster algorithm in terms of their storage requirements and speed. A priori it is not clear what granularity (i.e. Trotter number) to assign the discrete-time cluster algorithm in such a comparison. In one sense, the discrete-time cluster algorithm requires an infinite amount of storage and time to achieve the "same" result as the continuous-time cluster algorithm. There is also the question of inherent tradeoff of storage and speed that all computer programs face. The two approaches have fundamentally different storage schemes and time consumption patterns. In the face of these imponderables, we provide no formal comparison of storage and speed for the continuous and discrete cases.

On the other hand, from a practical standpoint, one runs the discrete-time cluster algorithm at successively finer granularities, ultimately settling on a Trotter number that gives satisfactory results. Informally, we find that the continuous-time cluster algorithm requires

Table 2-3. Comparison of discrete and continuous time cluster algorithm results for 2-D AFHM. Uniform and staggered susceptibilities are shown. Discrete results are from [Wiese94]. Both susceptibilities agree quite well.

β	L	4N(disc.)	χ_u (disc.)	χ_u (cont.)	χ_s (disc.)	χ_s (cont.)
5	6	256	0.0482(3)	0.0488(3)	9.67(3)	9.65(3)
5	8	256	0.0514(3)	0.0512(3)	16.08(5)	16.13(5)
5	10	256	0.0527(3)	0.0525(3)	23.73(7)	23.54(7)
5	12	256	0.0530(3)	0.0532(3)	32.3(1)	32.3(1)
5	14	256	0.0519(3)	0.0527(3)	41.7(1)	41.7(1)
5	16	256	0.0531(3)	0.0528(3)	52.6(2)	52.3(2)
5	18	256	0.0528(3)	0.0530(3)	64.3(2)	63.8(2)
5	20	256	0.0535(3)	0.0528(3)	76.3(3)	76.2(3)
10	6	512	0.0268(3)	0.0268(3)	14.65(5)	14.53(5)
10	8	512	0.0406(3)	0.0404(3)	27.5(1)	27.42(9)
10	10	512	0.0442(3)	0.0448(3)	42.9(2)	42.6(1)
10	12	512	0.0460(3)	0.0475(3)	60.6(2)	60.4(2)
10	14	512	0.0469(3)	0.0473(3)	81.2(3)	81.1(3)
10	16	512	0.0476(3)	0.0478(3)	103.1(4)	103.0(3)
10	18	512	0.0480(3)	0.0480(3)	129.0(4)	128.3(4)
10	20	512	0.0477(3)	0.0478(3)	156.2(5)	156.7(5)

much less storage, in some cases by more than an order of magnitude, than a roughly equivalent discrete-time cluster algorithm. CPU time also seems to be less, although not by as large a margin.

Notwithstanding reluctance to compare the discrete-time cluster algorithm and continuous-time cluster algorithm, an understanding of storage and time requirements for the latter is needed for setting array sizes (and project schedules!). Figures 2-5 through 2-7 show key storage and CPU time trends. "`Mused`" is the number of transitions in the spin-time lattice. "`Nseg`" is the maximum number of segments in a single cluster. In these plots both are normalized by the lattice volume βL^2. "`That`" is defined as CPU time per 1000 configurations examined. The plots show trends for the range of simulations run for the chiral perturbation theory study discussed below.

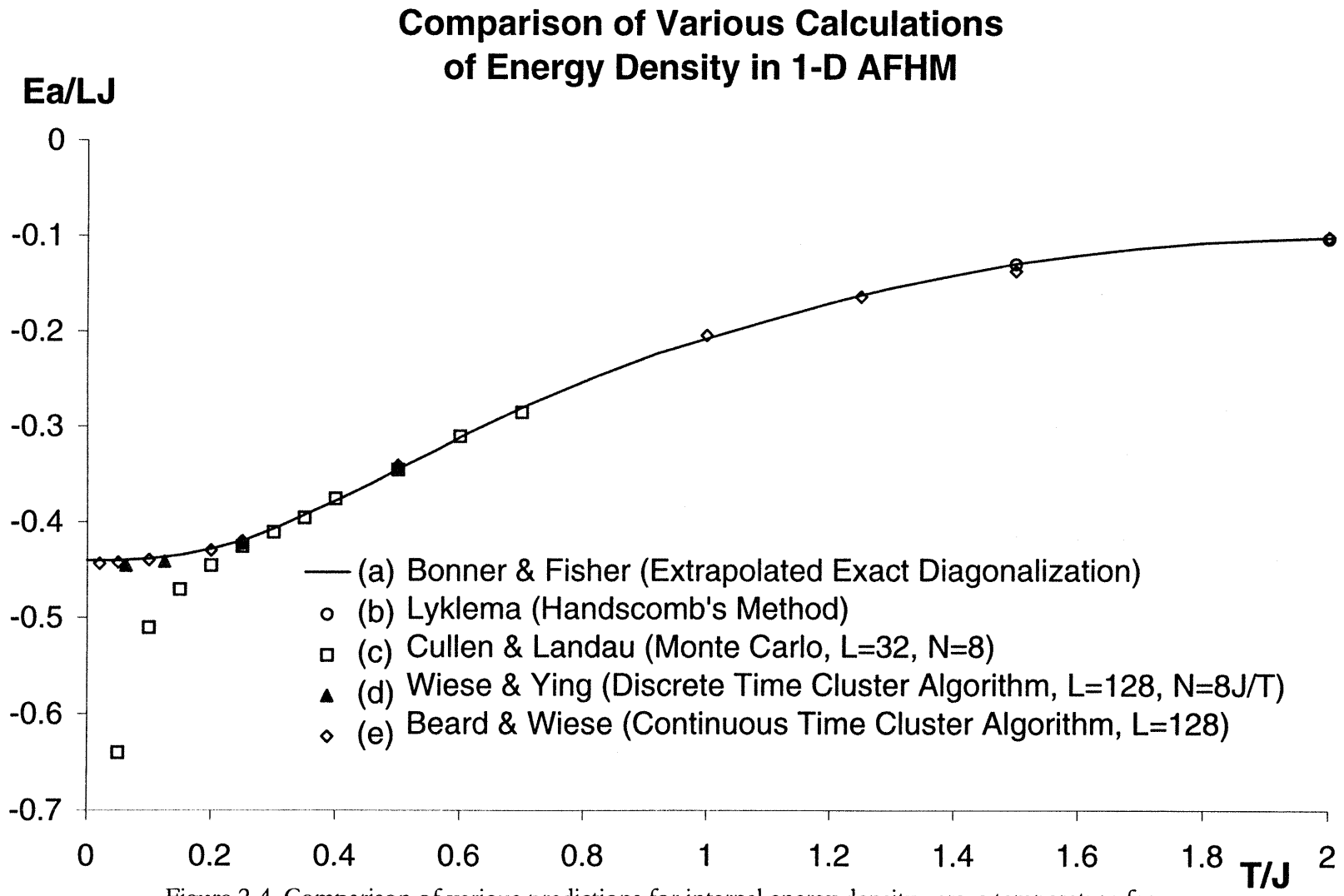


Figure 2-4. Comparison of various predictions for internal energy density versus temperature for the 1-D AFHM. The continuous-time cluster algorithm provides an excellent fit with the exact result

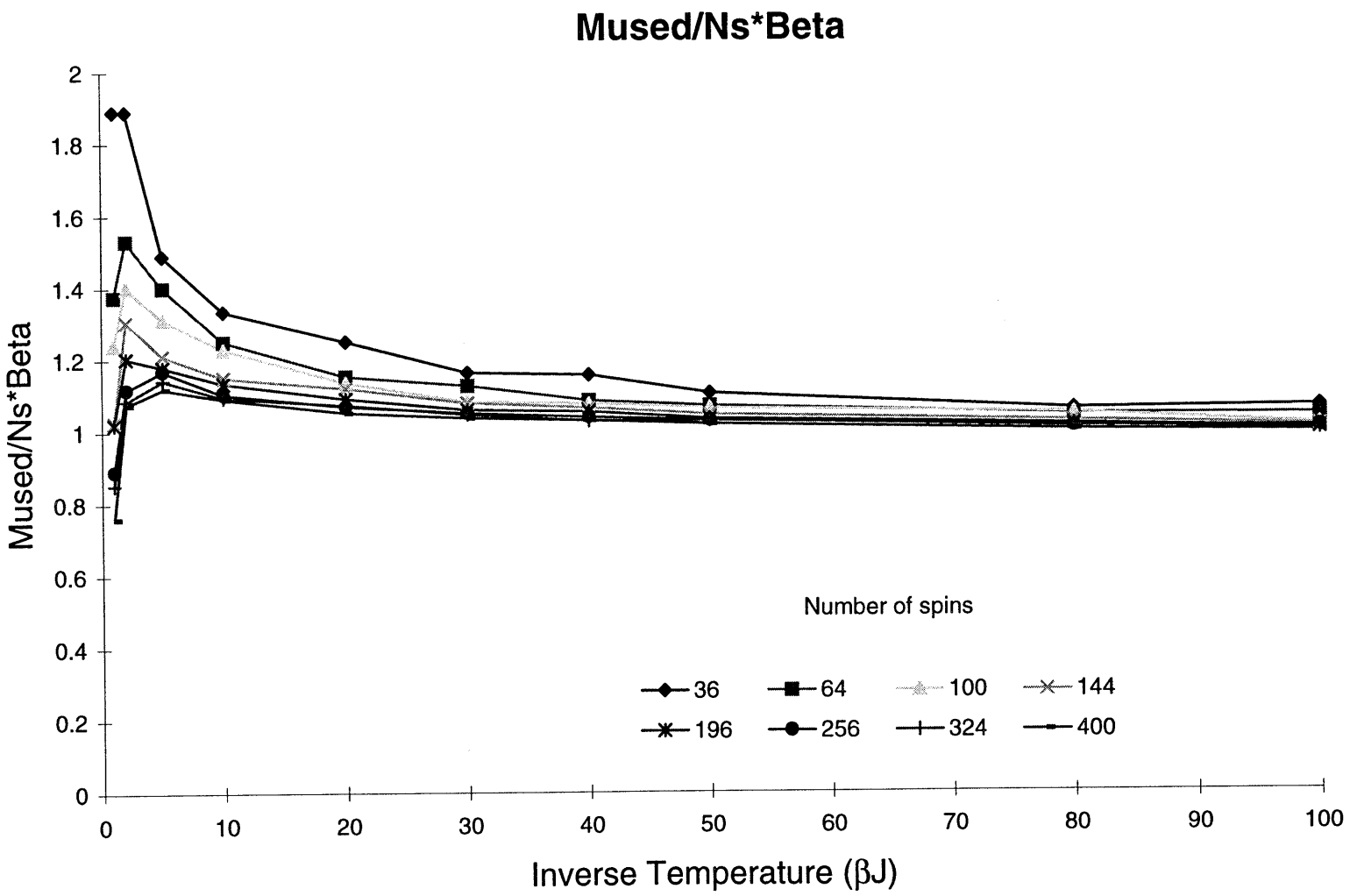


Figure 2-5. Mused normalized by the lattice volume. Mused is the maximum number of transitions in the lattice configuration encountered during a simulation.

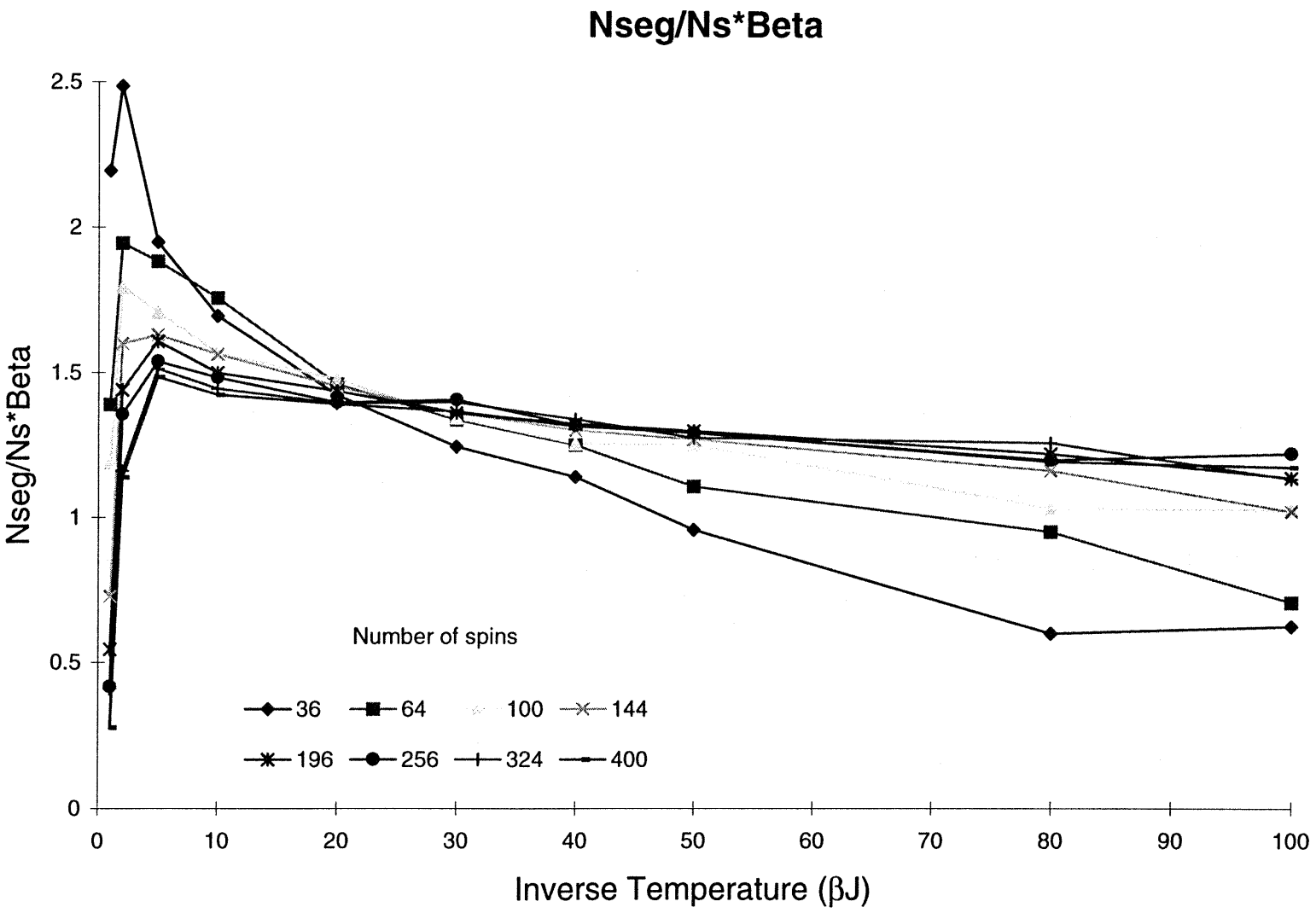


Figure 2-6. Nseg normalized by lattice volume. Nseg is the maximum number of path segments in a single loop encountered during a simulation.

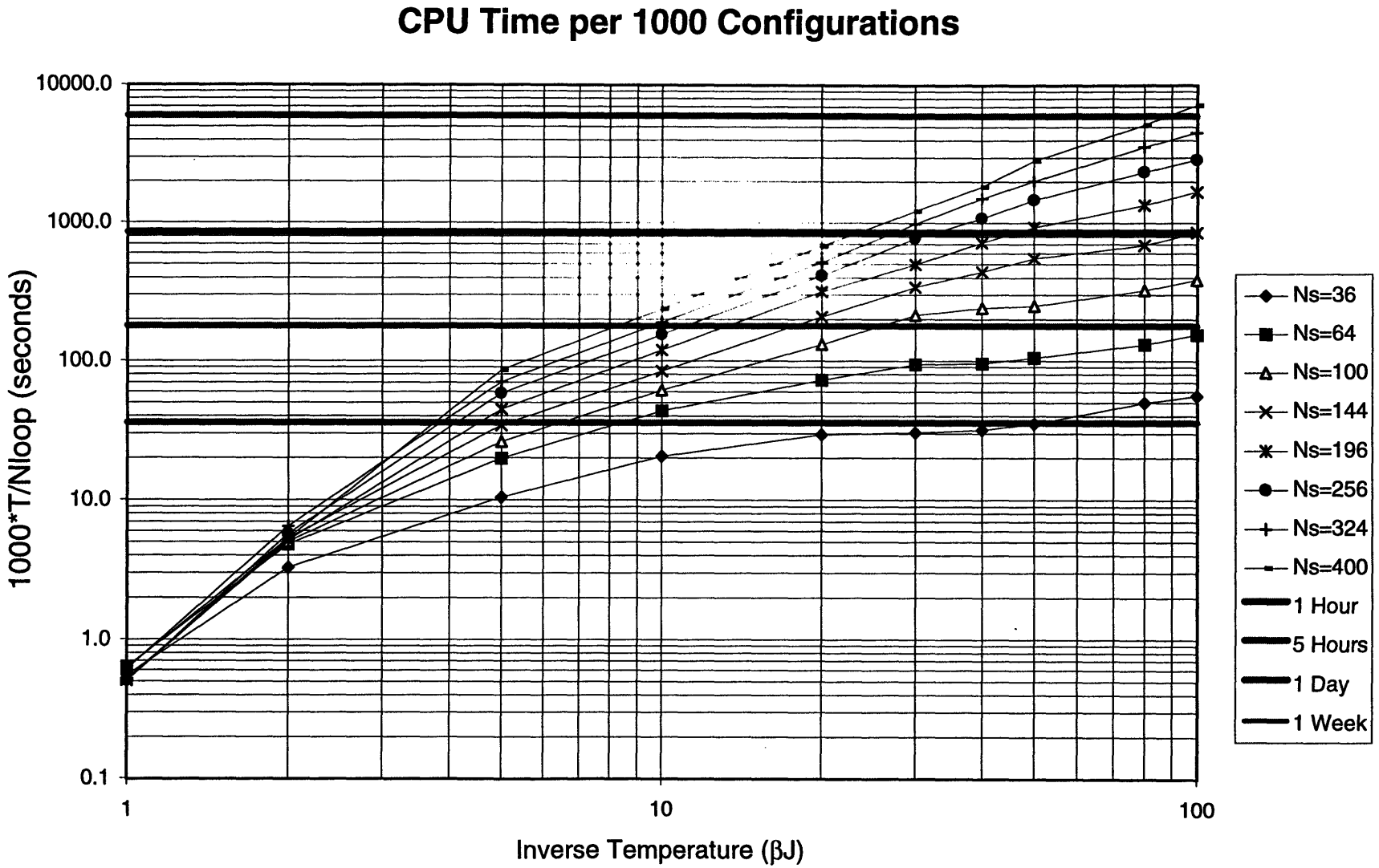


Figure 2-7. CPU Time per 1000 configurations for a range of volumes and temperatures. All data were run on a DEC Alpha workstation. Horizontal guides give total time for 100,000 configurations.

Chiral perturbation theory predictions for the AFHM

Hasenfratz and Niedermayer used chiral perturbation theory (CPT) to develop an extensive set of predictions for the susceptibilities and energies for the spin ½ AFHM [Hasenfratz93]. Chiral perturbation theory is a powerful tool for analyzing quantum systems. It was originally developed to model pion interactions in quantum chromodynamics, but has since been applied to a variety of systems with spontaneously broken symmetries. In CPT, one starts with the most general effective local action that satisfies whatever symmetries are present, and expresses the observables in a theory in terms of a handful of empirical, undetermined coefficients of terms in the action. One is left with a theory with some number of undetermined parameters but with otherwise strong predictions for dynamics.

In the chiral perturbation theory treatment, the observables of the spin ½ AFHM are expressed in terms of the following parameters:

- ground state energy density e_0
- infinite-volume staggered magnetization: M_s
- spin-wave velocity: $\hbar c$
- spin stiffness: ρ_s

The data for the discrete-time cluster algorithm presented in Table 2-3 were extracted from a study that carried out a high-precision fit for the parameters of the chiral perturbation theory treatment of the spin ½ AFHM [Wiese94]. This study generated data for uniform and staggered susceptibilities and internal energy in the so-called "cubical" temperature regime (where $TL/\hbar c \cong 1$) and fit the functional forms from [Hasenfratz93] to these data. This study found

- $e_0 \doteq -0.6693(1)\,J/a^2$
- $M_s \doteq 0.3074(4)/a^2$
- $\hbar c \doteq 1.68(1)Ja$
- $\rho_s \doteq 0.186(4)J$

An independent fit for the parameters of CPT/AFHM

Chiral perturbation theory makes explicit predictions about the low temperature properties of the AFHM. In the ground state of the 2-D AFHM the staggered magnetization develops an expectation value, and hence the $O(3)$ spin rotational symmetry gets spontaneously broken down to $O(2)$. The low energy excitations of the systems are spinwaves (so-called magnons) which are the Goldstone bosons of the spontaneously broken $O(3)$ symmetry.

The predictions of CPT at extremely low temperatures were checked using a range of square volumes with side length L/a = 6,8...20 and a range of inverse temperatures βJ = 1, 2, 5, 10, 20, 30, 40, 50, 80, 100. Note that the very small temperatures T = 0.01 J are inaccessible to

the discrete-time code, largely due to storage limitations. In every case 100,000 configurations were generated after a thermalization period of 1000 configurations.

Several of the key predictions of CPT were directly verified with the continuous-time cluster algorithm. In particular, the energy spectrum is that of an $O(3)$ rotor, with energy levels characterized by an integer spin j, having degeneracy $2j+1$ and energy levels distributed as $j(j+1)$. Finite volume effects are computed as expansions in $\hbar c/\rho_s L$, in units where the lattice spacing $a=1$. The energy spectrum is computed in [Hasenfratz93] to be

$$E_j - E_0 = j(j+1)\frac{(\hbar c)^2}{2\rho_s L^2}\left[1-\frac{\hbar c}{\rho_s L}\frac{3.900265}{4\pi}+O\left(\frac{1}{L^2}\right)\right]$$

Including only the leading term in the partition function, the uniform susceptibility approaches the form

$$\chi_u \xrightarrow[T\to 0]{} \frac{6}{L^2 T}\exp\left(-\frac{(\hbar c)^2}{\rho_s L^2 T}\right)$$

and the staggered susceptibility goes to the temperature-independent form

$$\chi_s \xrightarrow[T\to 0]{} \frac{2M_s^2\rho_s}{(\hbar c)^2}L^4\left[1+3\frac{\hbar c}{\rho_s L}\frac{3.900265}{4\pi}+O\left(\frac{1}{L^2}\right)\right].$$

These functional forms in the large β limit, including both volume and temperature dependence, were verified for uniform and staggered susceptibility. In addition, the staggered susceptibility is shown to plateau at large β as predicted by CPT. An independent fit for the CPT parameters M_s, $\hbar c$, and ρ_s gives excellent agreement with the results of [Wiese94], as shown in Table 2-4. This fit required that the partition function of the $O(3)$ rotor be included in its entirety, instead of including just the leading term, as the limiting forms in the equations above employ. In addition, only inverse temperatures $\beta J \geq 10$ were used in the fit, since the rotor approximation is valid only for very small temperatures. In order to reproduce the accuracy of the fit in [Wiese94], it was necessary to find fitted values for the coefficients of the quadratic terms in the expressions for energy and staggered susceptibility, that is, the coefficients of the terms $(\hbar c/\rho_s L)^2$ in the expressions above.

Note that the agreement between the current work and reference [Wiese94] is particularly remarkable because they are derived for different volume-temperature regimes with entirely different functional forms for χ_u and χ_s. Reference [Wiese94] was concerned with the "cubical" regime $TL/\hbar c \cong 1$ while the current study focuses on the "cylindrical" regime $TL/\hbar c << 1$. Quadratic coefficients for the cubical regime are not included in Table 2-4 because they are not comparable to those in the cylindrical regime. The five parameter values in Table 2-4 resulted in a goodness-of-fit $\chi^2/\text{d.o.f.} = 1.50$. Figures 2-8 and 2-9 show the fit for uniform and staggered susceptibility, respectively. Solid lines represent the fitted

Table 2-4. Comparison of Fitted Parameters for CPT/AFHM

	[Wiese94]	Current
Spin stiffness ρ_s	0.186(4)	0.185(2)
Spin-wave velocity $\hbar c$	1.68(1)	1.68(1)
Staggered magnetization M_s	0.3074(4)	0.3083(2)
Quadratic coeff: Energy	-	0.068(1)
Quadratic coeff: Stag. Mag.	-	0.338(7)

CPT result. Circles and error bars are displayed at each simulation point. The fit is quite good for $\beta J \geq 10$.

It should be reiterated that we see here the convergence of theory, experiment, and numerical simulation. Chiral perturbation theory provides a framework for understanding the physics of materials like La_2CuO_4. Numerical simulations are capable of delivering high-precision estimates of the parameters of the theory. Laboratory experiments, the grounding of all this activity, verify that the parameters and theory are consistent with the behavior of real materials and hence show that we have progressed in our understanding of high-temperature superconductors.

Summary

To summarize the important concepts of this chapter: a continuous-time limit of the discrete-time cluster rules was found to be quite natural, and is related to a prescription for path integrals by Farhi and Gutmann. The continuous-time cluster algorithm has substantial computational advantages over the discrete-time cluster algorithm. These advantages are exploited to confirm the predictions of chiral perturbation theory in the extreme low temperature limit.

The astute reader will have noted that the current work does not make a fit for the ground-state internal energy density. And therein lies a tale. The next chapter discusses the development of improved estimators for the observables in the AFHM, and in particular the poor numerical behavior of the energy.

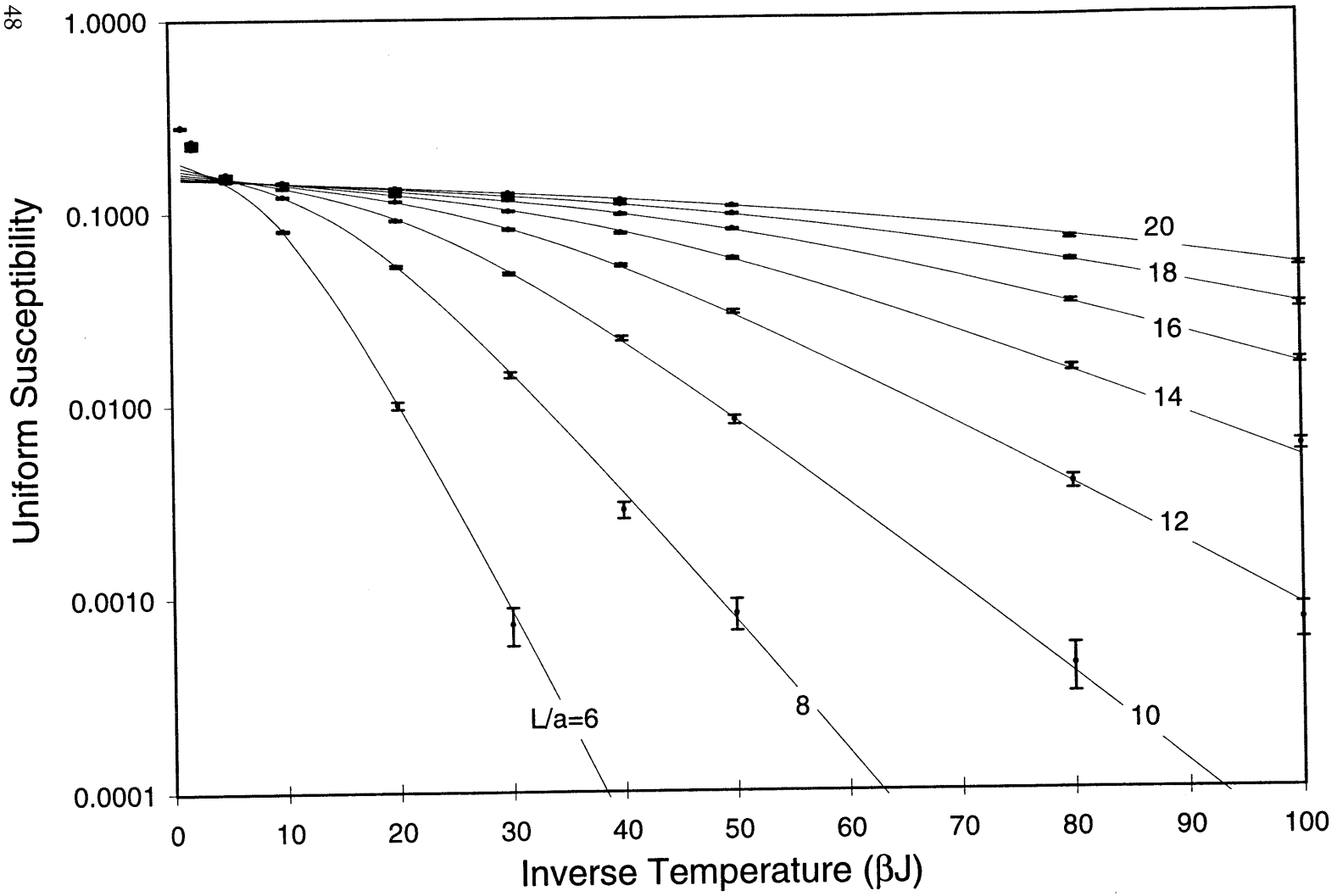


Figure 2-8. Uniform susceptibility versus inverse temperature for various volumes. Solid lines are predictions from chiral perturbation theory with fitted parameters; circles with error bars are lattice simulations with continuous time. The fit is very good for $\beta J > 10$.

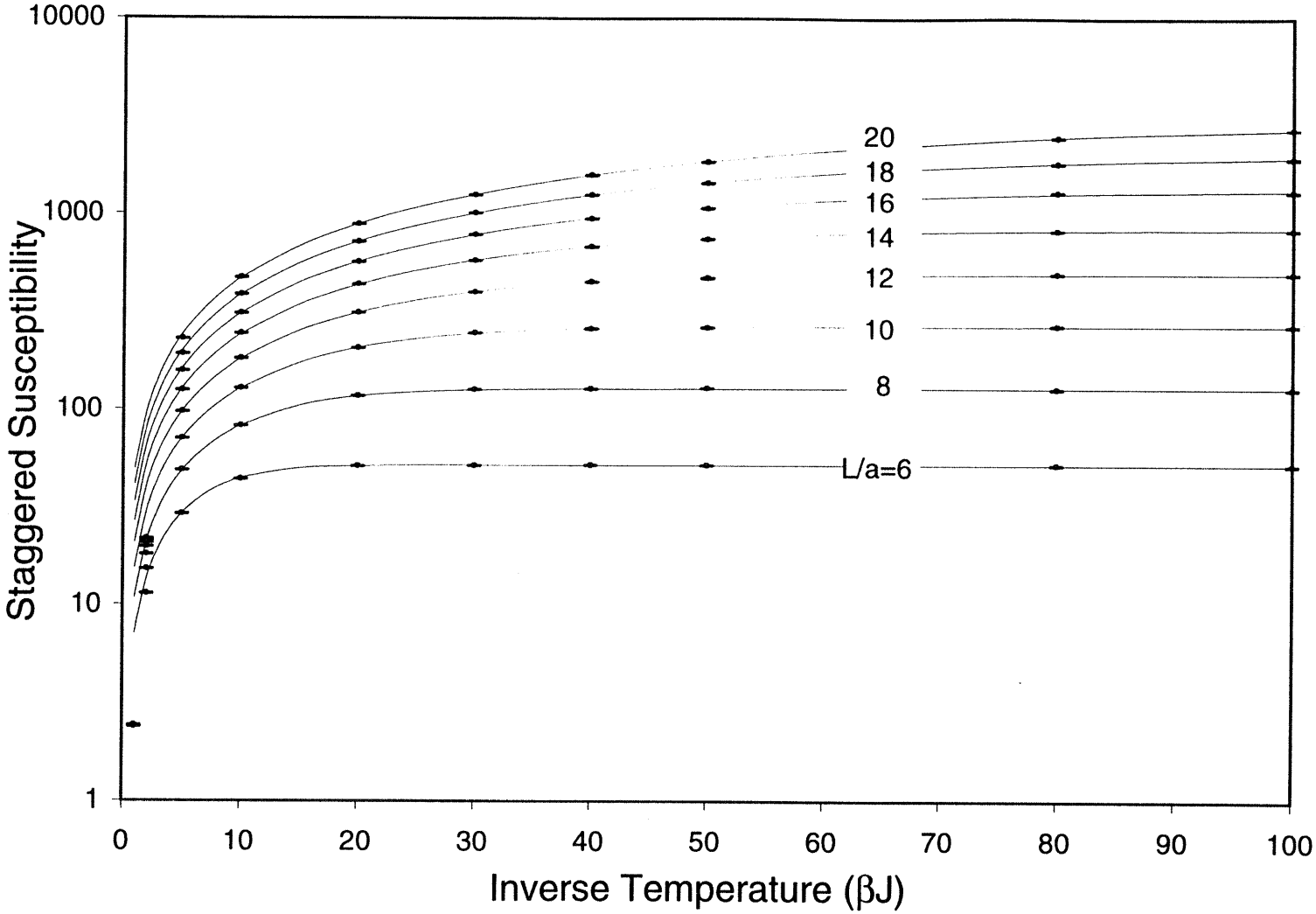


Figure 2-9. Staggered susceptibility versus inverse temperature for various volumes. Solid lines are predictions from chiral perturbation theory with fitted parameters; circles with error bars are lattice simulations with continuous time. The fit is very good for $\beta J > 10$.

"All progress is precarious, and the solution of one problem brings us face to face with another problem."
– Martin Luther King, Jr.

Chapter 3. Improved Estimators

One of the principal advantages of the loop cluster algorithm is that it allows for the possibility of improved estimators for observables. We find that the discrete-time improved estimators for uniform and staggered susceptibility, and for internal energy density, are readily generalized to the continuous-time cluster algorithm. However, the improved estimator for the internal energy density is found to be poorly behaved numerically in the continuous-time limit.

Overview

So far we have only described the "single-cluster" loop cluster algorithm, in which we build and flip one cluster at a time. The generalization of this is the "multi-cluster" algorithm, in which we cover the spin-time lattice exhaustively with some number N_C of disjoint clusters. In the multi-cluster algorithm, we can measure an observable for a given configuration, and flip clusters in all combinations to make subsequent measurements. For one covering we can thus sample 2^{N_C} independent configurations.

The expectation value of an observable A is a summation over configurations

$$\langle A \rangle = \frac{1}{Z} \int DC \, A \exp(-S[C])$$

and the corresponding multi-cluster estimator is

$$\langle A \rangle = \frac{1}{2^{N_c}} \sum_{\mathrm{sgn}(C)=\pm} A_C .$$

Note this formula works only if the algorithm that generates the clusters is ergodic and satisfies detailed balance, as ours does.

We derive a single-cluster estimator by identifying the contribution to an observable measured only for the points in a given cluster.

$$\langle A \rangle = \left\langle \frac{A_C}{|C|} \right\rangle$$

Note we must divide by the cluster size $|C|$ to account for the fact that all clusters are weighted equally in the multi-cluster algorithm, but are selected in proportion to their size in the single-cluster version.

Susceptibilities

The uniform susceptibility is given by the estimator

$$\chi_u = \frac{\beta}{L^d}\left\langle M^2 \right\rangle = 2dN\beta\left\langle \frac{M_C^2}{|C|} \right\rangle$$

where M_C is the magnetization of a cluster [Wiese92]. Note that in the discrete-time cluster algorithm, M_C is the summation of the spin eigenvalues for each lattice point in the cluster (and hence must be divided by the Trotter number in the continuum limit). Because the cluster moves forward in time when the spin is up and backward in time when the spin is down, it is not hard to see that M_C will be identically zero unless the cluster wraps around the time direction. In fact, we identify M_C (suitably normalized) with the winding number of the cluster in the time direction.

In our implementation of the continuous-time cluster algorithm, we use a version of this formula that works directly with the winding number W_C:

$$\chi_u = \frac{3\beta}{4}\left\langle \frac{W_C^2}{|C|} \right\rangle .$$

It should be noted that this definition of susceptibility gives answers that are an exact factor of 3 greater than the definitions used in [Wiese92] and [Wiese94]. This corresponds to taking all three spin components into account in the definition of susceptibility, rather than just the 3-component. Note the 4 in the denominator simply accounts for the spin being s=½.

The cluster size in the discrete-time cluster algorithm is simply the number of lattice points in the cluster. In our implementation of the continuous-time cluster algorithm (in which we define periodic time to run from 0 to 1), the cluster size is defined so that a maximal cluster would have magnitude $|C| = L^d \equiv N_s$.

The staggered susceptibility is similarly defined, except that there is no easy identification with the winding number. Instead we sum up the staggered spin for each lattice site in the discrete-time cluster (and each loop segment in the continuous-time cluster). Happily, because the stagger factor changes whenever the loop jumps from one site to a neighbor, the staggered magnetization of a cluster is identically equal to its size, and we find a direct relationship between the staggered susceptibility and the average cluster size:

$$\chi_s = \frac{3\beta}{4}\left\langle \frac{\chi_{s,C}^2}{|C|} \right\rangle = \frac{3\beta}{4}\left\langle \frac{|C|^2}{|C|} \right\rangle = \frac{3\beta}{4}\langle |C| \rangle .$$

This relation strongly suggests that the clusters themselves have physical meaning.

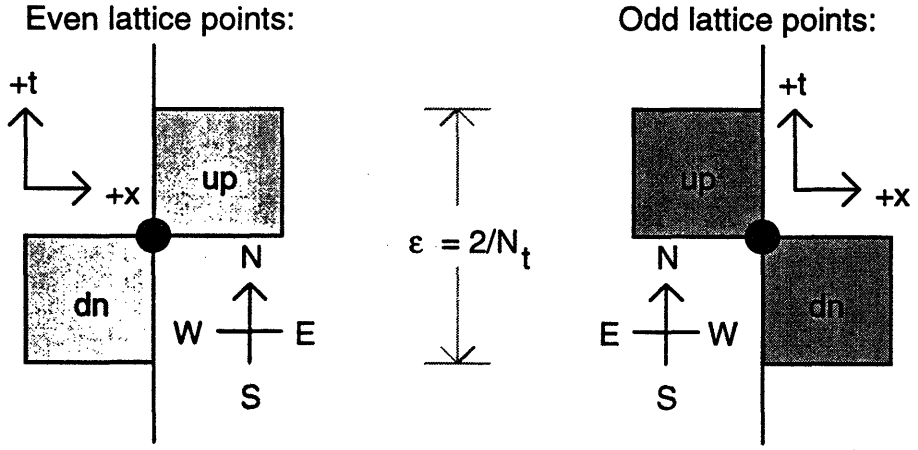
Internal energy density

The internal energy density is related to the partition function by

$$e = -\frac{1}{L^d}\frac{d\ln Z}{d\beta}.$$

An improved estimator for the internal energy density was developed and implemented for the discrete-time cluster algorithm some time ago [Wiese92]. Because this estimator has not been documented, we present an exhaustive summary of the energy estimator rules here.

The basic geometry of alternating plaquettes (with time in the vertical direction) is



The "energy matrix" in the discrete cluster algorithm, with corresponding plaquettes, is

$$\left.\begin{aligned} energy(0,0,0) &= \varepsilon J/16 = energy(1,1,1) \\ energy(0,0,1) &= (\varepsilon J/16)\left(\frac{1-3\exp(\varepsilon\beta J)}{1+\exp(\varepsilon\beta J)}\right) = energy(1,1,0) \\ energy(0,1,0) &= (\varepsilon J/16)\left(\frac{1+3\exp(\varepsilon\beta J)}{1-\exp(\varepsilon\beta J)}\right) = energy(1,0,1) \end{aligned}\right\} \Leftrightarrow \left\{\begin{array}{cc|cc} - & - & + & + \\ - & - & + & + \\ \hline - & + & + & - \\ - & + & + & - \\ \hline - & + & + & - \\ + & - & - & + \end{array}\right.$$

Other entries (*energy*(0,1,1) and *energy*(1,0,0)) are zero and are not used.

The logic for the improved estimator follows. We accumulate deltas to the energy based on the status of the current plaquette. The energy estimator is re-zeroed for each path and accumulated external to the path-building logic.

Note $\boxed{x \equiv \varepsilon\beta J}$ throughout. Also the notation 1=spin up and 0=spin down is employed.

IF CURRENT=1 THEN (with CURRENT=①)

IF NORTH=EAST THEN

IF N=E=1 THEN $\begin{pmatrix} 1 & 1 \\ \Uparrow & \Uparrow \\ ① & 1 \end{pmatrix}$ OR $\begin{pmatrix} 1 & 1 \\ \Uparrow & \Uparrow \\ 1 & ① \end{pmatrix}$ (continue on in time)

IF NEW VISIT THEN

$$\Delta ene = energy(1,1,1) + energy(1,1,0)$$
$$= (\varepsilon J/16)\left(1+\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1-e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{x^2}{16}\right)$$

ELSE IF PREVIOUSLY VISITED THEN

$$\Delta ene = 3\cdot energy(1,1,1) - energy(1,1,0)$$
$$= (\varepsilon J/16)\left(3-\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1+3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(\frac{x}{4}+\frac{x^2}{16}\right)$$

ENDIF

ELSE IF N=E=0 THEN $\begin{pmatrix} 0 & \Rightarrow & 1 \\ & & \\ ① & \Rightarrow & 0 \end{pmatrix}$ OR $\begin{pmatrix} 1 & \Leftarrow & 0 \\ & & \\ 0 & \Leftarrow & ① \end{pmatrix}$ (forced transition)

IF NEW VISIT THEN

$$\Delta ene = energy(0,1,0) + energy(1,1,0)$$
$$= (\varepsilon J/16)\left(\frac{1+3e^{\varepsilon\beta J}}{1-e^{\varepsilon\beta J}}+\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1+3e^{2\varepsilon\beta J}}{1-e^{2\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{1}{4}-\frac{x}{8}-\frac{x^2}{12}\right)$$

ELSE IF PREVIOUSLY VISITED THEN

$$\Delta ene = 3\cdot energy(0,1,0) - energy(1,1,0)$$
$$= (\varepsilon J/16)\left(3\frac{1+3e^{\varepsilon\beta J}}{1-e^{\varepsilon\beta J}}-\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1+8e^{\varepsilon\beta J}+3e^{2\varepsilon\beta J}}{1-e^{2\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{3}{4}-\frac{x}{8}\right)$$

ENDIF

ENDIF

ELSE IF NORTH≠EAST THEN $\begin{pmatrix} 1 & \overset{1-p}{\Leftarrow} & 0 \\ \Uparrow & & \Downarrow \\ ① & \Rightarrow & 0 \end{pmatrix} p$ OR $\begin{pmatrix} 0 & \overset{1-p}{\Rightarrow} & 1 \\ \Downarrow & & \Uparrow \\ 0 & \Leftarrow & ① \end{pmatrix} p$ (continuation | decay)

IF CONTINUATION (p) THEN

IF NEW VISIT THEN

$$\Delta ene = energy(1,1,0) + energy(1,1,1)$$
$$= (\varepsilon J/16)\left(\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}+1\right) = 2(\varepsilon J/16)\left(\frac{1-e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) \xrightarrow[\varepsilon\to 0]{} \frac{1}{\beta}\left(-\frac{x^2}{16}\right)$$

ELSE IF PREVIOUSLY VISITED THEN

$$\Delta ene = 3\cdot energy(1,1,0) - energy(1,1,1)$$
$$= (\varepsilon J/16)\left(3\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}-1\right) = 2(\varepsilon J/16)\left(\frac{1-5e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) \xrightarrow[\varepsilon\to 0]{} \frac{1}{\beta}\left(-\frac{x}{4}-\frac{3x^2}{16}\right)$$

ENDIF

ELSE IF DECAY ($1-p$) THEN

IF NEW VISIT THEN

$$\Delta ene = energy(1,1,0) + energy(0,1,0)$$
$$= (\varepsilon J/16)\left(\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}+\frac{1+3e^{\varepsilon\beta J}}{1-e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1+3e^{2\varepsilon\beta J}}{1-e^{2\varepsilon\beta J}}\right) \xrightarrow[\varepsilon\to 0]{} \frac{1}{\beta}\left(-\frac{1}{4}-\frac{x}{8}-\frac{x^2}{12}\right)$$

ELSE IF PREVIOUSLY VISITED THEN

$$\Delta ene = 3\cdot energy(1,1,0) - energy(0,1,0)$$
$$= (\varepsilon J/16)\left(3\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}-\frac{1+3e^{\varepsilon\beta J}}{1-e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1-8e^{\varepsilon\beta J}+3e^{2\varepsilon\beta J}}{1-e^{2\varepsilon\beta J}}\right) \xrightarrow[\varepsilon\to 0]{} \frac{1}{\beta}\left(\frac{1}{4}-\frac{x}{8}-\frac{x^2}{6}\right)$$

ENDIF

ENDIF

ENDIF

ELSE IF CURRENT=0 THEN (with CURRENT=⓪)

IF SOUTH=WEST THEN

IF S=W=0 THEN $\begin{pmatrix} ⓪ & 0 \\ \Downarrow & \Downarrow \\ 0 & 0 \end{pmatrix}$ OR $\begin{pmatrix} 0 & ⓪ \\ \Downarrow & \Downarrow \\ 0 & 0 \end{pmatrix}$ (continue on in time)

IF NEW VISIT THEN

$$\Delta ene = energy(0,0,0) + energy(0,0,1)$$
$$= (\varepsilon J/16)\left(1+\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1-e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{x^2}{16}\right)$$

ELSE IF PREVIOUSLY VISITED THEN

$$\Delta ene = 3\cdot energy(0,0,0) - energy(0,0,1)$$
$$= (\varepsilon J/16)\left(3-\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1+3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(\frac{x}{4}+\frac{x^2}{16}\right)$$

ENDIF

ELSE IF S=W=1 THEN $\begin{pmatrix} ⓪ & \Rightarrow & 1 \\ & & \\ 1 & \Rightarrow & 0 \end{pmatrix}$ OR $\begin{pmatrix} 1 & \Leftarrow & ⓪ \\ & & \\ 0 & \Leftarrow & 1 \end{pmatrix}$ (forced transition)

IF NEW VISIT THEN

$$\Delta ene = energy(1,0,1) + energy(0,0,1)$$
$$= (\varepsilon J/16)\left(\frac{1+3e^{\varepsilon\beta J}}{1-e^{\varepsilon\beta J}}+\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1+3e^{2\varepsilon\beta J}}{1-e^{2\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{1}{4}-\frac{x}{8}-\frac{x^2}{12}\right)$$

ELSE IF PREVIOUSLY VISITED THEN

$$\Delta ene = 3\cdot energy(1,0,1) - energy(0,0,1)$$
$$= (\varepsilon J/16)\left(3\frac{1+3e^{\varepsilon\beta J}}{1-e^{\varepsilon\beta J}}-\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1+8e^{\varepsilon\beta J}+3e^{2\varepsilon\beta J}}{1-e^{2\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{3}{4}-\frac{x}{8}\right)$$

ENDIF

ENDIF

ELSE IF SOUTH≠WEST THEN $\begin{pmatrix} ⓪ & \xrightarrow{1-p} & 1 \\ \Downarrow & & \Uparrow \\ 0 & \xleftarrow{} & 1 \end{pmatrix}_{p}$ OR $\begin{pmatrix} 1 & \xleftarrow{1-p} & ⓪ \\ \Uparrow & & \Downarrow \\ 1 & \xrightarrow{} & 0 \end{pmatrix}_{p}$ (continuation | decay)

IF CONTINUATION (p) THEN

IF NEW VISIT THEN

$$\Delta ene = energy(0,0,1) + energy(0,0,0)$$

$$= (\varepsilon J/16)\left(\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}+1\right) = 2(\varepsilon J/16)\left(\frac{1-e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{x^2}{16}\right)$$

ELSE IF PREVIOUSLY VISITED THEN

$$\Delta ene = 3\cdot energy(0,0,1) - energy(0,0,0)$$

$$= (\varepsilon J/16)\left(3\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}-1\right) = 2(\varepsilon J/16)\left(\frac{1-5e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{x}{4}-\frac{3x^2}{16}\right)$$

ENDIF

ELSE IF DECAY ($1-p$) THEN

IF NEW VISIT THEN

$$\Delta ene = energy(0,0,1) + energy(1,0,1)$$

$$= (\varepsilon J/16)\left(\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}+\frac{1+3e^{\varepsilon\beta J}}{1-e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1+3e^{2\varepsilon\beta J}}{1-e^{2\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(-\frac{1}{4}-\frac{x}{8}-\frac{x^2}{12}\right)$$

ELSE IF PREVIOUSLY VISITED THEN

$$\Delta ene = 3\cdot energy(0,0,1) - energy(1,0,1)$$

$$= (\varepsilon J/16)\left(3\frac{1-3e^{\varepsilon\beta J}}{1+e^{\varepsilon\beta J}}-\frac{1+3e^{\varepsilon\beta J}}{1-e^{\varepsilon\beta J}}\right) = 2(\varepsilon J/16)\left(\frac{1-8e^{\varepsilon\beta J}+3e^{2\varepsilon\beta J}}{1-e^{2\varepsilon\beta J}}\right) \xrightarrow{\varepsilon\to 0} \frac{1}{\beta}\left(\frac{1}{4}-\frac{x}{8}-\frac{x^2}{6}\right)$$

ENDIF

ENDIF

ENDIF

ENDIF

Now the distillation of all this logic for the continuum-time algorithm is quite simple: essentially we take the $\varepsilon \to 0$ limit, keeping in mind that summing over some plaquettes (such as forced continuations) becomes an integration in Euclidean time, while for other plaquettes (such as forced transitions) we find a finite sum. So our rules are:

- Any Δene contribution for continuous plaquettes (forced continuations or optional decays that have been declined) is truncated to order $x \equiv \varepsilon\beta J$, because the higher-order contributions, $O(\varepsilon^2)$, disappear upon integration with respect to Euclidean time.
- Any Δene contribution for sporadic plaquettes (forced transitions or optional decays that have been accepted) is truncated to order 1, because a finite sum of terms $O(\varepsilon)$ disappears in the continuum limit.

Working through the logic, we find that the 4-page flow-chart above turns into a handful of rules:

For every...	**add...**
• new free transition,	$-1/4\beta$
• cancellation of a new free transition,	$+1/4\beta$
• cancellation of an existing transition,	$-1/4\beta$
• unit of segment length traversed next to a **previous** segment of the same path,	$-J/4$ for each masked neighbor

The rules here are couched in slightly modified terminology that is more natural for the continuum algorithm. The somewhat ominous-sounding reference to "masked neighbors" simply means those nearest neighbors that have already hosted a path segment in the current cluster. They are "masked" in the sense that the path cannot jump to those sites because they have already selected continuation-in-time as their flow choice. A "free transition" is the same as an "optional decay" that has been accepted. It is "new" on the first visit by a path and is "canceled" by a revisit. A "cancellation of an existing transition" means the path has hit a forced transition.

The energy bug

Now, as it happens, this is a fairly easily implemented set of rules. Fairly far into the exploration of the cylindrical temperature regime of the spin ½ AFHM, we discovered that this estimator has poor numerical behavior.

The crux of the matter can be seen in Figure 3-1. This plot shows the internal energy of a 4-spin system as predicted by the continuous-time cluster algorithm with the improved estimator for internal energy, compared with the exactly known result. We see that there is excellent agreement for high temperatures (small β), but a systematic error that arises when the temperature approaches zero (high β).

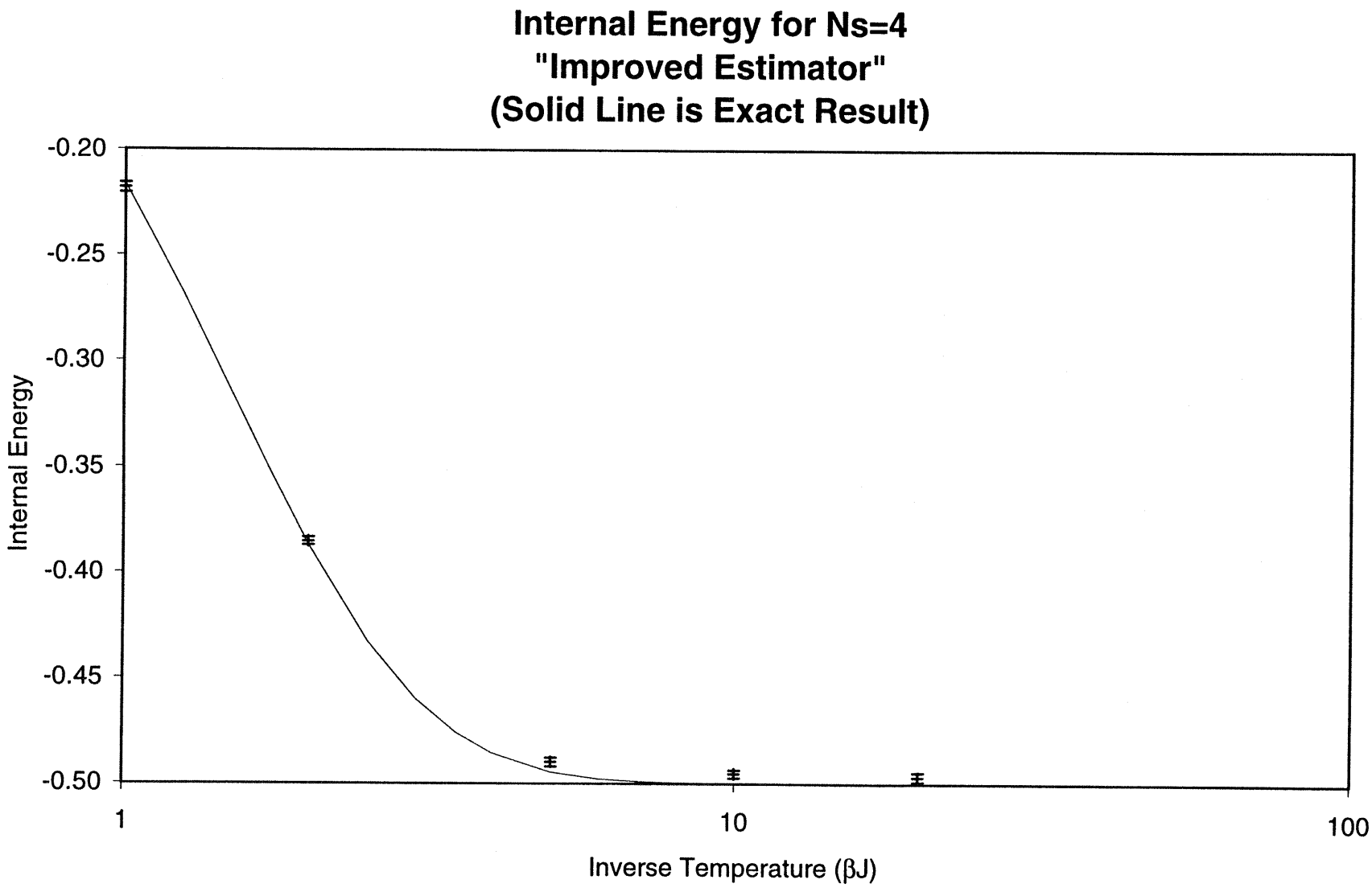


Figure 3-1. Internal energy versus β for the 4 spin system. The improved estimator has a systematic error at large β.

This discrepancy is mildly perplexing, not least because the energy estimator rules as constructed above are so simple that programming bugs can be eliminated fairly quickly as an explanation.

We believe we have tracked the discrepancy to the nature of the statistical distribution of the energy estimator. Figure 3-2 shows a comparison, for $\beta = 20$ and the 4 spin system, of the binning distribution of the energy estimator for both the improved estimator and the (now inappropriately named) unimproved estimator. The latter is the estimator based on measuring the energy on the entire spin-time lattice after every configuration update. The meaning of the binning distribution is as follows: after each configuration, a contribution *ene* to the internal energy is computed. For the improved estimator, this contribution is based on the latest cluster. For the unimproved estimator, the contribution is based on the entire spin-time lattice. Based on the size of the contribution, we tally a count in a bin $[x, x + dx)$ where $x = -ene$ and $dx = 0.05$.

The binning distribution for the energy estimator is clearly more spread out for the improved estimator. Actually, the situation is far worse than apparent in the first chart: Figure 3-3 shows the binning distribution on a logarithmic scale. It is at once apparent that the unimproved estimator is closely Gaussian while the improved estimator has a very long tail. (Note that there were 175 configurations out of 100,000 that produced $-ene > 5$ – these show up as a spike at $x = 5$ in this plot). The exact result for the internal energy is -0.50000. We see that the unimproved estimator gives results consistent with the exact result

$$\mu_{\text{unimproved}} = -0.50013 \pm 0.00014$$

while the improved estimator misses the mark by several error bars:

$$\mu_{\text{improved}} = -0.49305 \pm 0.00195$$

Also shown in Figure 3-3 is the best-fit Gaussian for the unimproved estimator and a best-fit ansatz functional form for the tail of the improved estimator. The latter is of the form

$$\log f(x) = A - B(x - C)^k.$$

We find

$$A = 21.95$$
$$B = 19.49$$
$$C = 0.1955$$
$$k = 0.0606$$

as best-fit parameters for this case. It is the small value of k that produces the long tail of this distribution.

With this motivation, we investigated a family of distributions of this type both analytically and with a Monte Carlo procedure with the goal of demonstrating that this long-tailed behavior is problematical.

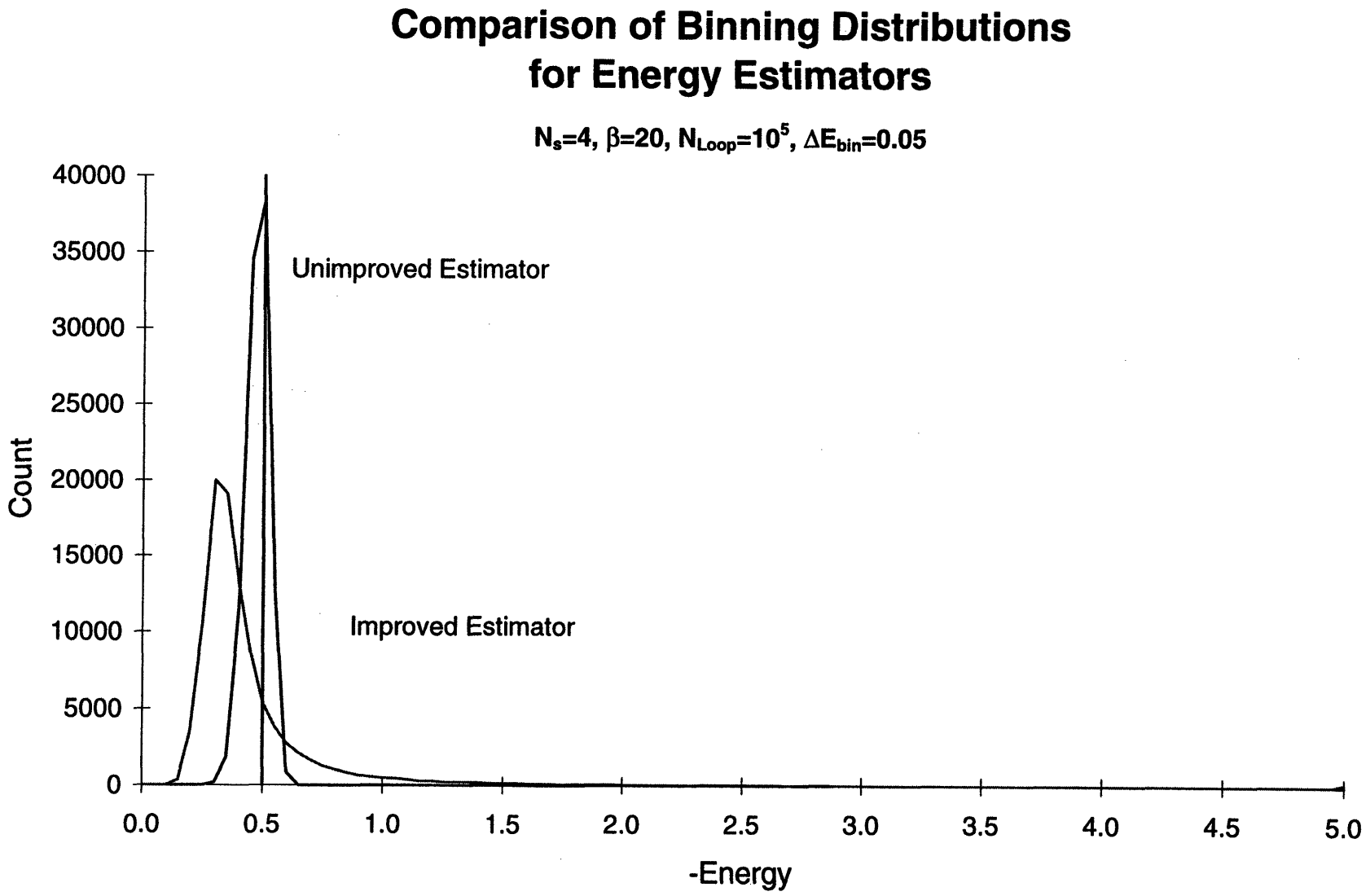


Figure 3-2. Binning distribution for improved and unimproved estimators for internal energy density. The improved estimator has a long tail.

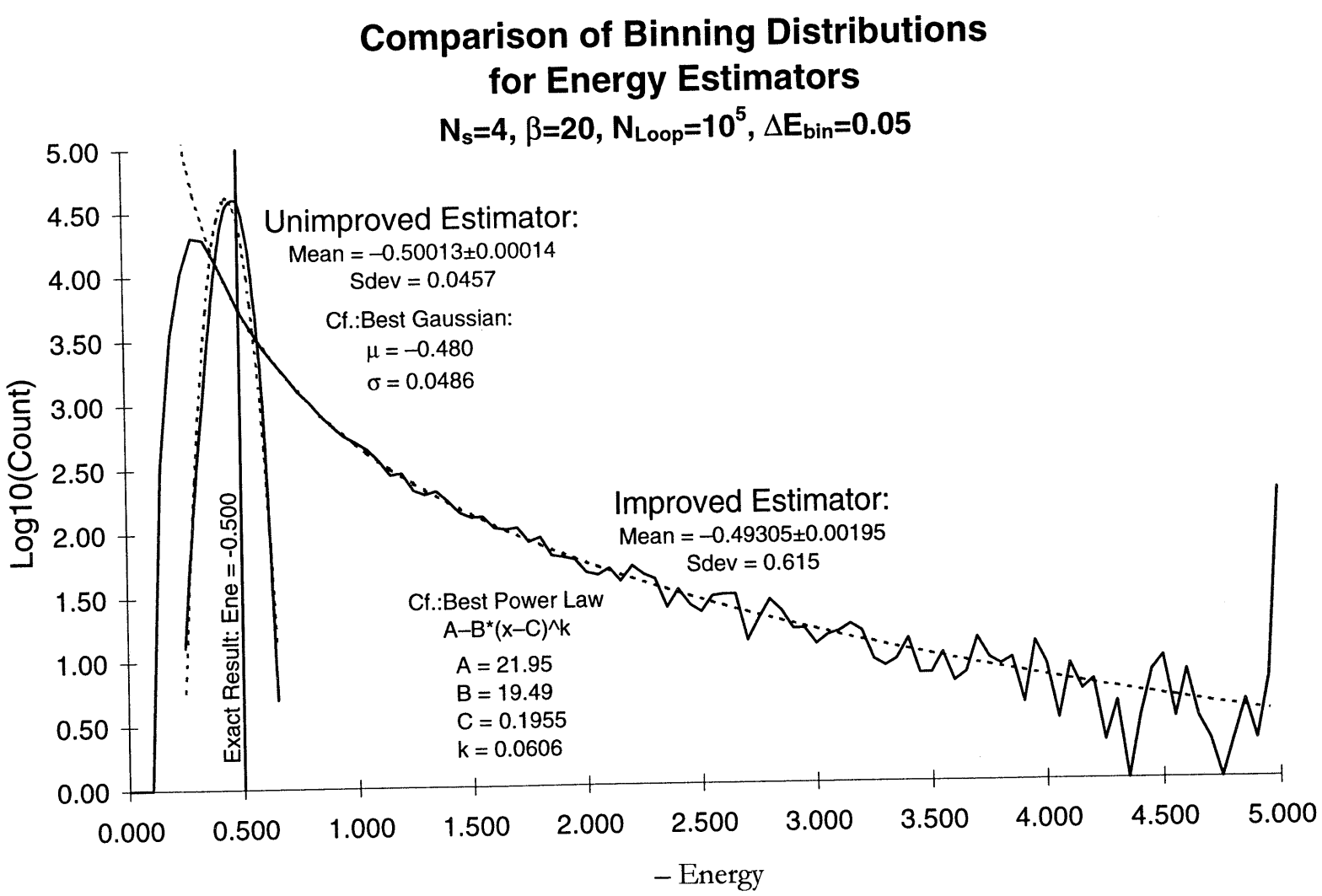


Figure 3-3. Binning distribution for energy estimators. Best-fit Gaussian and power-law exponential are shown for the estimators.

Poorly behaved Monte Carlo estimates

Let us assume that we have a random variable $x \in [0,\infty)$ whose distribution function is given by

$$f(x) = A\exp\left(-\alpha x^{p}\right).$$

The normalization constant A is given by [1]

$$A^{-1} = \int_0^\infty \exp\left(-\alpha x^{p}\right)dx = \frac{\Gamma(1/p)}{p\alpha^{1/p}}.$$

The mean of this distribution is

$$\mu \equiv \int_0^\infty x f(x)dx = \frac{\Gamma(2/p)}{p\alpha^{2/p}} \Big/ \frac{\Gamma(1/p)}{p\alpha^{1/p}} = \frac{\Gamma(2/p)}{\Gamma(1/p)}\alpha^{-1/p}$$

and its variance is

$$\begin{aligned}
\sigma^2 &\equiv \int_0^\infty (x-\mu)^2 f(x)dx \\
&= \int_0^\infty x^2 f(x)dx - \mu^2 \\
&= \frac{\Gamma(3/p)}{\Gamma(1/p)}\alpha^{-2/p} - \mu^2 \\
&= \frac{\Gamma(3/p)}{\Gamma(1/p)}\alpha^{-2/p} - \left(\frac{\Gamma(2/p)}{\Gamma(1/p)}\alpha^{-1/p}\right)^2 \\
&= \alpha^{-2/p}\left(\frac{\Gamma(3/p)}{\Gamma(1/p)} - \left(\frac{\Gamma(2/p)}{\Gamma(1/p)}\right)^2\right)
\end{aligned}$$

In terms of $\lambda \equiv \alpha^{-1/p}$ (which has the same dimensions as x), we have

$$\mu/\lambda = \frac{\Gamma(2/p)}{\Gamma(1/p)}$$

and

$$\sigma/\lambda = \sqrt{\frac{\Gamma(3/p)}{\Gamma(1/p)} - \left(\frac{\Gamma(2/p)}{\Gamma(1/p)}\right)^2}$$

---

[1] It is useful to recall the handy identity $\int_0^\infty x^n \exp\left(-\alpha x^{p}\right)dx = \frac{\Gamma((n+1)/p)}{p\alpha^{(n+1)/p}}$ for $n > -1$ and $p, \alpha > 0$.

Of course, we know that for large argument the gamma function can be approximated by Sterling's formula

$$\Gamma(x) \approx x^x e^{-x} \sqrt{\frac{2\pi}{x}}\left(1+\frac{1}{12x}+\frac{1}{288x^2}-\frac{139}{51840x^3}-\frac{571}{2488320x^4}\cdots\right)$$

which implies that for small p,

$$\frac{\Gamma(2/p)}{\Gamma(1/p)} \approx \frac{(2/p)^{2/p} e^{-2/p}}{(1/p)^{1/p} e^{-1/p}} \sqrt{\frac{1/p}{2/p}} = \frac{1}{\sqrt{2}} e^{-1/p} (4/p)^{1/p}$$

so that

$$\ln(\mu/\lambda) \doteq \frac{1}{p}\left(\ln\frac{1}{p}-1\right)-\frac{1}{2}\ln 2$$

Similarly

$$\frac{\Gamma(3/p)}{\Gamma(1/p)} \approx \frac{(3/p)^{3/p} e^{-3/p}}{(1/p)^{1/p} e^{-1/p}} \sqrt{\frac{1/p}{3/p}} = \frac{1}{\sqrt{3}} e^{-2/p} \left(\sqrt{27}/p\right)^{2/p}$$

so

$$\sigma/\lambda = \sqrt{\frac{1}{\sqrt{3}} e^{-2/p} \left(\sqrt{27}/p\right)^{2/p} - \frac{1}{2} e^{-2/p} (4/p)^{2/p}}$$

$$= e^{-1/p} (1/p)^{1/p} \sqrt{\frac{1}{\sqrt{3}} (27)^{1/p} - \frac{1}{2} (16)^{1/p}}$$

To facilitate a numerical analysis, we would like to generate random x following the distribution $f(x)$. For this we need the integral form of this distribution

$$F(x) \equiv \int_0^x f(x')dx'$$

Fortunately, a relatively manageable integral form can be found for the special cases $p = 1/m$ where m is an integer. In this case

$$F(x) = \int_0^x A \exp\left(-\alpha x'^{1/m}\right) dx'$$

and the normalization factor is

$$A^{-1} = \frac{\Gamma(m)}{\frac{1}{m}\alpha^m} \Rightarrow A = \frac{\alpha^m}{m!}$$

Then

$$F(x) = \frac{\alpha^m}{m!} \int_0^x \exp\left(-\alpha x'^{1/m}\right) dx' = 1 - \exp\left(-\alpha x^{1/m}\right) \sum_{k=0}^{m-1} \frac{\left(\alpha x^{1/m}\right)^k}{k!}$$

from a substitution $u = \alpha x^{1/m} \Rightarrow \frac{m}{\alpha^m} u^{m-1} du = dx$ and repeated integration by parts. It is easily verified that this expression satisfies the expected properties $F(0) = 0$ and $F(\infty) = 1$.

This function must be inverted numerically, but the task is facilitated by the relation $F'(x) = f(x)$, which helps in the construction of a Newton-Raphson iterative scheme.

It will facilitate matters further to work with a distribution for which $\mu = 1$; in other words we will select

$$\lambda = \frac{\Gamma(m)}{\Gamma(2m)} \Rightarrow \alpha = \lambda^{-1/m} = \left(\frac{\Gamma(2m)}{\Gamma(m)} \right)^{\frac{1}{m}}$$

Also, the standard deviation is related to the mean via

$$\sigma/\mu = \sqrt{\frac{\Gamma(3m)\Gamma(m)}{\Gamma(2m)^2} - 1}$$

Now, armed with these relations, we performed a series on Monte Carlo simulations with increasing m – that is, with increasing "tail length". In each case we generated 100,000 numbers according to the distribution $f(x)$, and computed the mean and standard error of the sample population. Figure 3-4 shows the comparison of the empirical mean and error with the exact mean (unity) and error from the analysis above.

We see that the longer the tail of the distributions in this family, the poorer the estimate of the mean. Also note that the standard error is severely underestimated by this procedure. We believe this is the root of the problem with the improved estimator for internal energy.

Comparison with the discrete case

The question naturally arises whether this phenomenon was seen in the case of the discrete-time cluster algorithm. We believe not. An examination of the source of the long tail contributions to the improved estimator for the internal energy reveals that large $-ene$ numbers come entirely from extremely small clusters. That is, some fraction of the clusters generated in the continuum time algorithm consist of very small loops where two new transitions are created. Since the estimators in the single-cluster algorithm have cluster size in the denominator, and the numerator of these contributions has a fixed $2(-1/4\beta)$, we get arbitrarily large contributions. We do not see such huge contributions in the discrete case because the finite Trotter number in effect provides an ultraviolet cutoff – the smallest cluster size is $1/Nd$, that is, $1/2N$ for the 2-D AFHM. The tail of the distribution is cut off for the discrete-time cluster algorithm.

The silver lining is that the "unimproved" estimator is easy to implement and costs very little computation time compared to the rest of the algorithm. Unfortunately, the internal energy bug was not discovered until late in the study of the AFHM in the cylindrical temperature

regime, so an independent reduction for the ground-state energy density parameter e_0 was not possible.

The moral of this story is that one must be mindful of removing cutoff effects even when the apparent fallout seems benign. Quite possibly there are other cases of observables whose numerical behavior degrades rather than improves in the continuum limit.

In the meantime, we have gone on to apply the cluster algorithm to higher spin systems, which is the subject of the next chapter.

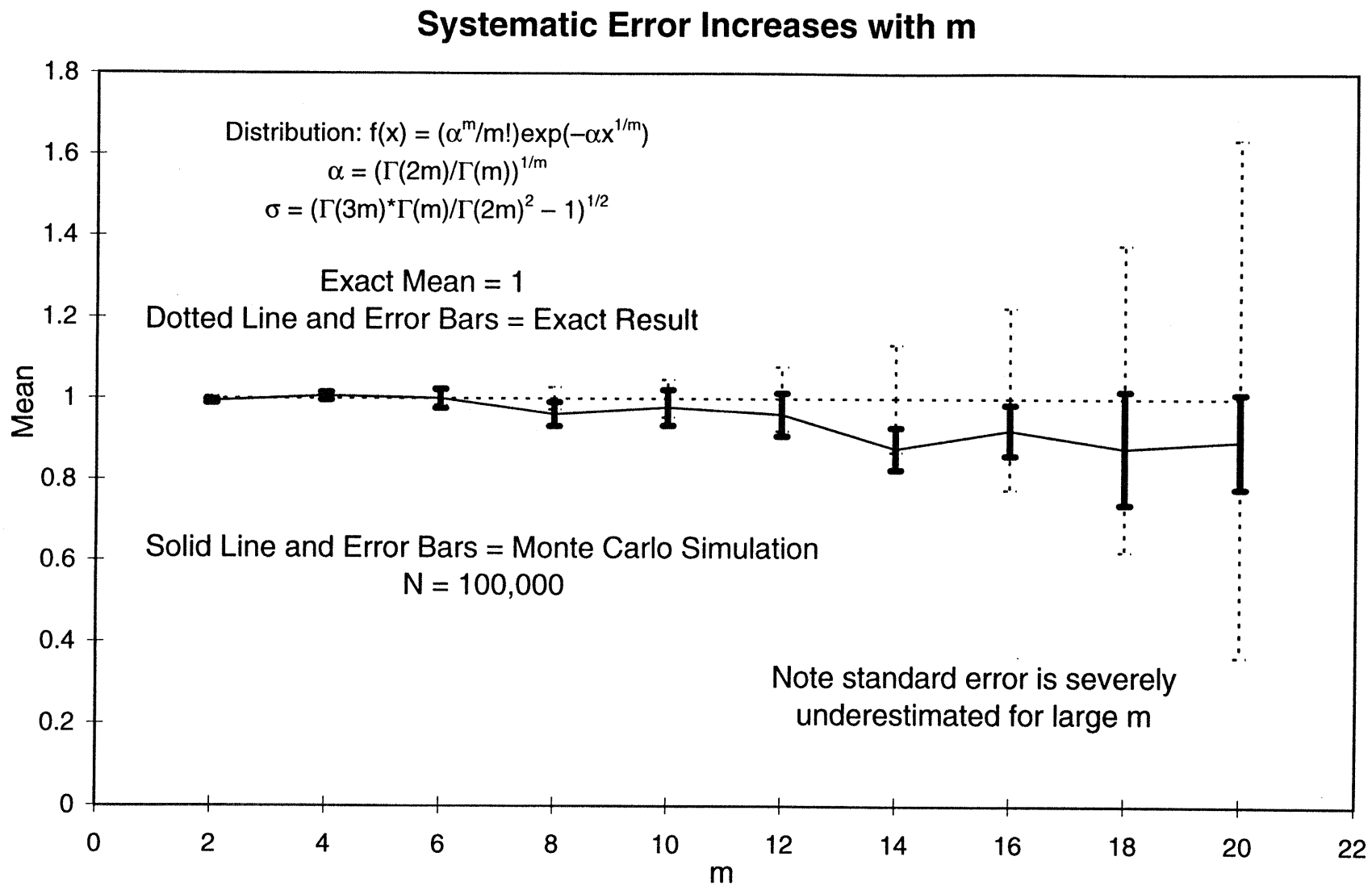


Figure 3-4. Systematic error for a family of long-tailed distributions. The estimate for the mean and standard error gets worse for longer-tailed distributions.

"We are like dwarves upon the shoulders of giants, and so able to see more and farther than the ancients."
– Bernard of Chartres

Chapter 4. Generalization to Higher Spin Systems

Some physical systems, such as K_2NiF_4, seem to be well-modeled by the Heisenberg model for higher spins [Greven95]. We would like to be able to apply the cluster algorithm for such systems. We can begin with a Trotter-Suzuki analysis, just as in the spin ½ case, and work in a 2+1-dimensional classical system.

The first conceptual hurdle is that these systems have three states at each lattice site, corresponding to -1, 0, and +1 components of spin in the 3-direction. It is not immediately clear how we should proceed to define a "spin flip" for sites belonging to a cluster. Further, we relied on the state of the system to dictate the direction of the cluster build at any given time; with 3 states, it is again not immediately clear how we should choose a path direction.

Spin 1 AFHM in Discrete Time

After some consideration, Martin Greven and Uwe-Jens Wiese came up with a scheme for implementing a loop cluster algorithm in discrete time. The scheme is described in detail in our preprint dealing with the spin 1 AFHM [Beard96b]; the basic idea is reviewed here in order to provide a foundation for discussing the continuous-time version.

Our approach to tackling the spin 1 system is to view every site as a pair of spin ½ subsites. We ensure that only the triplet state is counted by explicitly inserting projection operators between the usual transfer matrices of the Trotter-Suzuki method. This projection turns out to be equivalent to assuming that an infinitely strong *ferromagnetic* coupling exists between the two spin ½ subsites.

Just as described for the spin ½ system in Chapter 1, the spin 1 Hamiltonian is first decomposed into four terms

$$H = \sum_{\substack{\mu=1,2 \\ k=1,2}} H_{\mu,k}$$

$$H_{\mu,k} = J \sum_{x \in X_{\mu,k}} \vec{S}_x \cdot \vec{S}_{x+\hat{\mu}}$$

where

$$X_{1,1} = \{x | x = (2m, n)\}$$
$$X_{2,1} = \{x | x = (m, 2n)\}$$
$$X_{1,2} = \{x | x = (2m+1, n)\}$$
$$X_{2,2} = \{x | x = (m, 2n+1)\}$$

Figure 1-1 in Chapter 1 shows this basic decomposition.

The partition function is then

$$Z = \lim_{N\to\infty} \mathrm{Tr}\left[\exp\left(-\varepsilon\beta H_{1,1}\right)\exp\left(-\varepsilon\beta H_{2,1}\right)\exp\left(-\varepsilon\beta H_{1,2}\right)\exp\left(-\varepsilon\beta H_{2,2}\right)\right]^N$$

as usual.

The spins at each site are decomposed

$$\vec{S}_x = \vec{s}_{1,x} + \vec{s}_{2,x}$$

where $\vec{s}_{i,x}$ is now a spin ½ operator with $\vec{s}_{i,x}^{\,2} = 3/4$. The nearest neighbor interaction is

$$\vec{S}_x \cdot \vec{S}_{x+\hat{\mu}} = \vec{s}_{1,x} \cdot \vec{s}_{1,x+\hat{\mu}} + \vec{s}_{2,x} \cdot \vec{s}_{2,x+\hat{\mu}} + \vec{s}_{1,x} \cdot \vec{s}_{2,x+\hat{\mu}} + \vec{s}_{2,x} \cdot \vec{s}_{1,x+\hat{\mu}}$$

Each of the four Hamiltonians $H_{\mu,k}$ is then subdivided further into "direct" and "crossed" parts incorporating the first two and last two terms of the above expression.

$$H_{\mu,k,1} = J \sum_{x\in X_{\mu,k}} \left(\vec{s}_{1,x} \cdot \vec{s}_{1,x+\hat{\mu}} + \vec{s}_{2,x} \cdot \vec{s}_{2,x+\hat{\mu}}\right)$$

$$H_{\mu,k,2} = J \sum_{x\in X_{\mu,k}} \left(\vec{s}_{1,x} \cdot \vec{s}_{2,x+\hat{\mu}} + \vec{s}_{2,x} \cdot \vec{s}_{1,x+\hat{\mu}}\right)$$

Figure 4-1 makes this notation clearer.

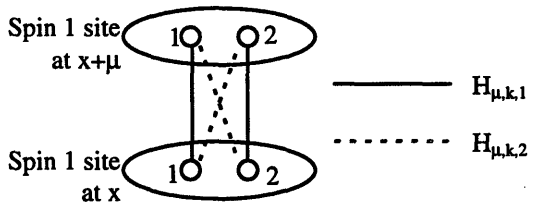


Figure 4-1. Splitting up the spin 1 interaction into two spin ½ terms

At this point the partition function has eight terms in each sandwich. We still need to project out the spin 1 states at every Euclidean time step. This we do with the projection operator

$$P = \prod_x P_x = \prod_x \frac{1}{2}\left(\vec{s}_{1,x} + \vec{s}_{2,x}\right)^2$$

As mentioned above, this projection is equivalent to switching on an infinitely strong ferromagnetic interaction between the two spins $\vec{s}_{1,x}$ and $\vec{s}_{2,x}$.

Finally, the partition function can be written

$$Z = \lim_{N\to\infty} \mathrm{Tr}[\exp(-\varepsilon\beta H_{1,1,1})P\exp(-\varepsilon\beta H_{1,1,2})P\exp(-\varepsilon\beta H_{2,1,1})P\exp(-\varepsilon\beta H_{2,1,2})P$$
$$\exp(-\varepsilon\beta H_{1,2,1})P\exp(-\varepsilon\beta H_{1,2,2})P\exp(-\varepsilon\beta H_{2,2,1})P\exp(-\varepsilon\beta H_{2,2,2})P]^N$$

Complete sets of spin ½ states are now inserted between each factor. The action is defined by the identification of the partition function with the expression

$$Z = \prod_{x,t} \sum_{s_i(x,t)=\pm 1/2} \exp(-S)$$

Our direct and crossed Hamiltonians have distinct expressions for the action: For the direct terms we have

$$\langle s_i(t)|\exp(-\varepsilon\beta H_{\mu,k,1})|s_i(t+1)\rangle = \prod_{x\in X_{\mu,k}} \exp(-S_A)\exp(-S_B)$$
$$S_A = S[s_1(x,t)s_1(x+\hat{\mu},t)s_1(x,t+1)s_1(x+\hat{\mu},t+1)]$$
$$S_B = S[s_2(x,t)s_2(x+\hat{\mu},t)s_2(x,t+1)s_2(x+\hat{\mu},t+1)]$$

and for the crossed terms we have

$$\langle s_i(t)|\exp(-\varepsilon\beta H_{\mu,k,2})|s_i(t+1)\rangle = \prod_{x\in X_{\mu,k}} \exp(-S_A)\exp(-S_B)$$
$$S_A = S[s_1(x,t)s_2(x+\hat{\mu},t)s_1(x,t+1)s_2(x+\hat{\mu},t+1)]$$
$$S_B = S[s_2(x,t)s_1(x+\hat{\mu},t)s_2(x,t+1)s_1(x+\hat{\mu},t+1)]$$

Just as in the spin ½ case, there are only 6 non-vanishing Boltzmann factors. The transfer matrix is the same. For bipartite lattices, we can use the same trick of including 180° rotation of staggered sites in our definition of the global basis. The transfer matrix is then

$$M = \exp(-\tfrac{1}{4}\varepsilon\beta J)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}(e^{\varepsilon\beta J}+1) & \frac{1}{2}(e^{\varepsilon\beta J}-1) & 0 \\ 0 & \frac{1}{2}(e^{\varepsilon\beta J}-1) & \frac{1}{2}(e^{\varepsilon\beta J}+1) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The matrix elements of the projection operator are

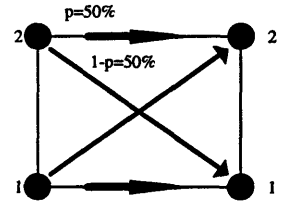
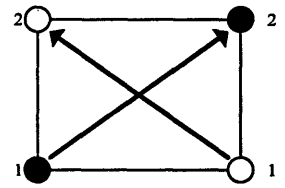
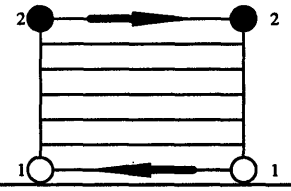
$$\langle s_i(t)|P|s_i(t+1)\rangle = \prod_x P(s_1(x,t), s_2(x,t), s_1(x,t+1), s_2(x,t+1))$$

or

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Leftrightarrow \begin{pmatrix} ++ \\ +- \\ -+ \\ -- \end{pmatrix} \text{basis}$$

The discrete-time cluster algorithm proceeds exactly as in the spin ½ case described in Chapter 1, except we now have a 3+1 dimensional problem, where the "z" dimension has a thickness equal to 2 spin sites, and now some plaquettes are governed by the transfer matrix P instead of M. For the latter, the flow rules are the familiar AFHM rules. For the P-type plaquettes, the flow rules are simple.

Table 4-1. Summary of Plaquette Flow Rules for projector P plaquettes. Solid circles indicate spin-up sites; open circles are spin-down. The time direction is horizontal. Flow rules for inverse plaquettes are analogous.

	Discrete Time
Optional layer shift, $p = 50\%$	
Forced layer shift	
Forced continuation	

Consider, for example, the optional layer shift. If the current spin is in layer 1, i.e. is a spin s_1, then with 50% probability the flow will move to layer 2 and with 50% probability the flow will remain in layer 1. The analogous rule holds if the current spin is in layer 2. The other rules are self-explanatory. It can be easily shown that these rules satisfy detailed balance for the transfer matrix P above.

In this way a loop is built site by site until it closes, just as in the familiar spin ½ AFHM.

This discrete-time cluster algorithm was implemented in a Fortran code and a range of volumes (up to $L^2 = 400$) and temperatures (up to $\beta = 15$) were run. For these cases we set the total number of time slices to $N_t = 160\beta$. Since there are 16 distinct layers in the time sandwich for the spin 1 algorithm, i.e. $N_t = 16N$, this means that the Trotter number was $N = 10\beta$. In each case we allowed 10,000 configurations for thermalization followed by 100,000 measurements.

Figures 4-2 and 4-3 show the uniform and staggered susceptibility for the simulations described above.

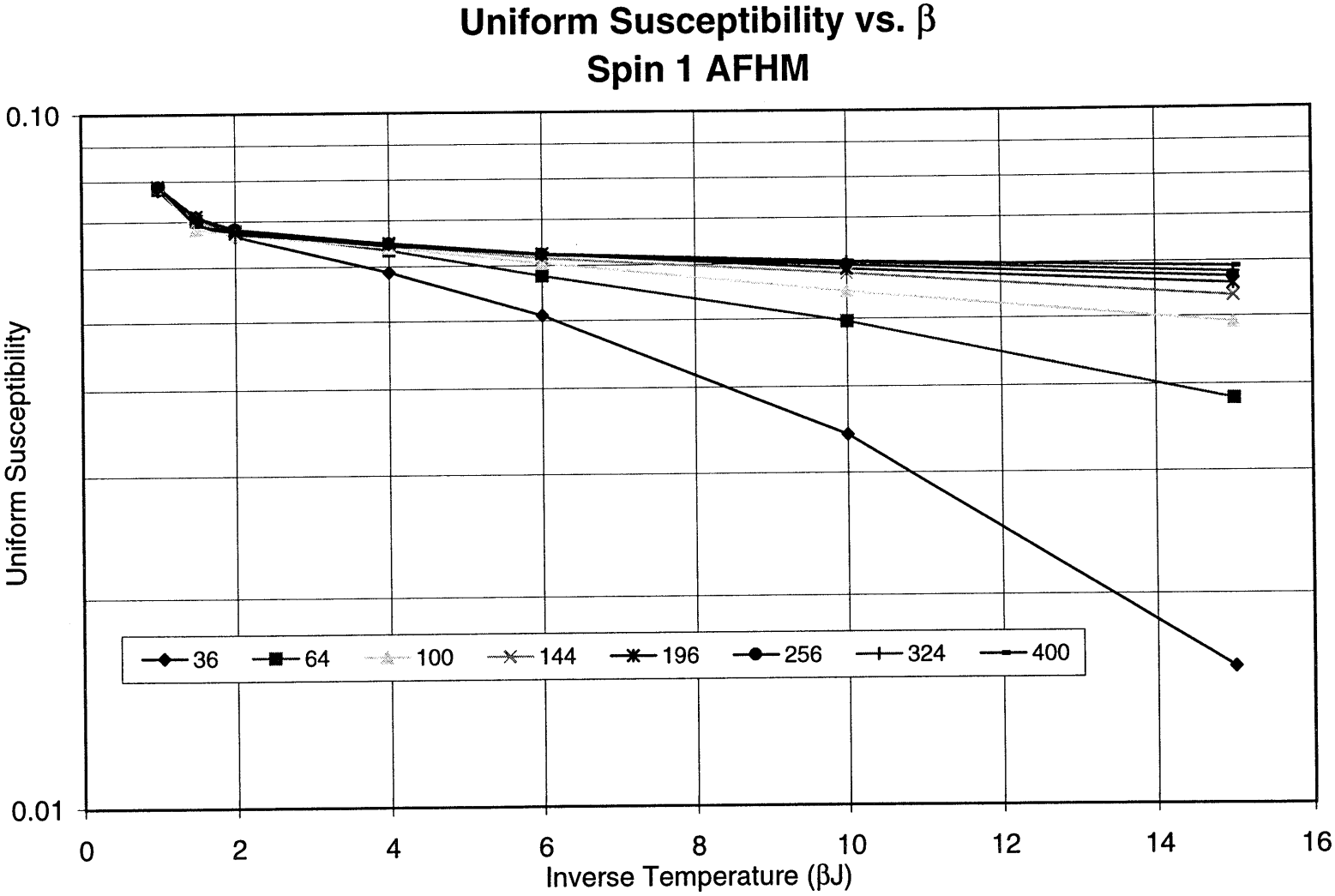


Figure 4-2. Uniform Susceptibility for the Spin 1 AFHM.

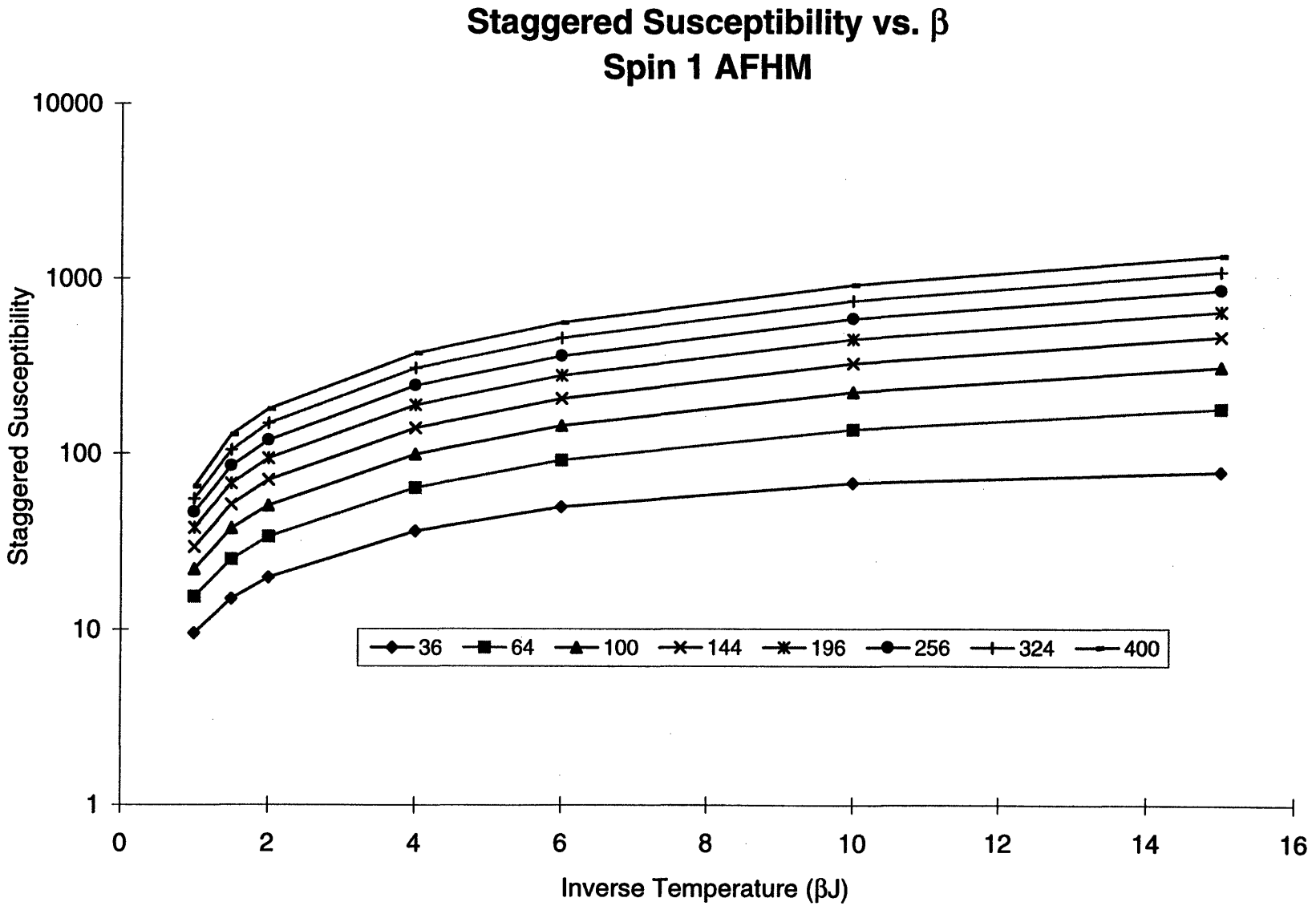


Figure 4-3. Staggered Susceptibility for the Spin 1 AFHM.

Proposal for Continuous-time Spin 1 Model

Armed with the discrete-time version of a loop cluster algorithm for the spin-1 AFHM, we proceed to outline a continuous-time algorithm. Time has not permitted us to carry out this plan; however, we hope to do so in the near future.

The basic idea of the decomposition above is that each spin 1 spin site is actually composed of two spin ½ particles, they appear as a single spin 1 object because they are strongly coupled ferromagnetically. We propose that the natural generalization of the continuous-time cluster algorithm to higher spins is to use the three-state spin 1 basis $\{m_s = -1,0,+1\}$ but to allow the cluster path to traverse each spin-time location up to two times.

We visualize that the flow at any point exists in one of two "constituent-spin" layers; the flow direction is determined by the spin state in that layer. If the spin state is +1, then both constituent spins are +½, so the flow will move forward in time on both first and second visits. Similarly, if the spin state is -1, both spins are -½ so the flow must move backward in time for both visits. If the spin state is 0, then it is +½ half the time and -½ the rest of the time, so the flow can move in either direction initially but must change direction on its second visit.

The coupling between spin sites is the familiar spin ½ coupling, except that we must take into account four of these couplings for every pair of spins. This means that adjacent sites that are both spin +1 do not allow a path jump, so they are forced continuations. If adjacent sites are +1 and 0, then half the time they are optional decay type and half the time forced transitions. Other rules for optional transitions would be similar.

When a bond exists between adjacent sites (the forced transition plaquettes in the spin ½ model) the path would be forced to move along the bond probabilistically. Suppose the bond connects a spin state +1 and a spin state 0 to a 0, +1. If we are moving forward in time on the +1 site and encounter the bond, then with 50% probability we continue forward along the spin 0 section, and with 50% probability we move along the bond, reverse direction, and continue backward in time along the spin 0 section. In this way a consistent set of rules can be implemented.

A "flip" would be defined within this picture by the operation of flipping the constituent-spin. Flipping a path segment moving forward in time on a spin +1 site would change the state to 0; flipping a path moving forward (backward) in time on a spin 0 site would change it to spin -1 (+1). Flipping a double-visit segment to a +1 site would change the spin to -1.

Clearly this algorithm would be more complex than the spin ½ case, but presumably the benefits would be the same or greater – complete elimination of the Trotter error, and the consequent elimination of technically esoteric extrapolation to the continuum limit.

In the meantime, we have been engaged in other aspects of the spin ½ and spin 1 models – in particular, the computation of the correlation length and comparison to the predictions of chiral perturbation theory. These are the subject of the next chapter.

"The medium is the message. This is merely to say that the personal and social consequences of any medium – that is, of any extension of ourselves – result from the new scale that is introduced into our affairs by each extension of ourselves, or by any new technology."
– Marshall McLuhan

Chapter 5. Studies Related to Correlation Length

Both the discrete-time cluster algorithm and the continuous-time cluster algorithm are well-suited for measurement of the correlation length ξ. In the AFHM the correlation length measures the distance dependence of the two-point correlation of the staggered magnetization.

It is convenient to project the staggered magnetization into one dimension to compute the correlation function. We sum over the x_2 dimension

$$\vec{M}_{s,x_1} = \sum_{x_2} (-1)^{x_1+x_2} \vec{S}_x .$$

The staggered correlation function is then defined as

$$C(x_1) = \int_0^\beta dt \left\langle \vec{M}_{s,0}(0) \cdot \vec{M}_{s,x_1}(t) \right\rangle$$

or equivalently,

$$C(x_1 - x_1') = \frac{1}{\beta} \int_0^\beta dt' \int_0^\beta dt \left\langle \vec{M}_{s,x_1'}(t') \cdot \vec{M}_{s,x_1}(t) \right\rangle$$

The correlation function has an exponential dependence on the separation x_1:

$$C(x_1) \propto \exp(-x_1 / \xi)$$

The length ξ is called the correlation length.

Implementation in discrete time

The basic computational strategy is to use an improved estimator for the correlation function. The determination of the correlation function is a two-stage process. For each cluster, we first accumulate the staggered magnetization projected onto the x_1 dimension in a one-dimensional array. When the cluster is finished, all pairs of x_1 coordinates are examined, the product of summed magnetizations is computed and divided by the cluster size, and contributions to $C(x_1 - x_1')$ are thus accumulated. These contributions, averaged

over the course of the Monte Carlo simulation, are the estimate for the correlation function. Standard errors are computed in the usual way.

Let us be explicit about the coding technique. The inner loop of the discrete-time cluster algorithm marches from spin site to spin site. At each site the x_1-coordinate `ix` can be projected from the spin index `is` with a simple modular relation:

```
ix = MOD(is-1,nx)+1
```

where `nx` is the dimension of the lattice in the x_1 direction. Then the staggered spin state is accumulated in an array:

```
lc1dsum(ix) = lc1dsum(ix) + istag(is)*(2*ising(is)-1)
```

Here `istag(is)` is a function which delivers the ± 1 stagger factor associated with spin site `is`. The spin state is stored as 1 or 0 in the array `ising(is)` and so must be affinely mapped as shown.

After a loop is built, the contribution of the loop to the correlation function is computed. All pairs of coordinates `ix` are examined to build up the contribution. To save time, only the elements of `lc1dsum` that are non-zero are examined; the number of non-null entries is `nnon` and they are stored in `nonnullx`. The inner loops of the module that computes the contribution to $C(x_1 - x_1')$ resemble the following:

```
      DO j1=1,nnon
        DO j2=1,nnon
        ixd = nonnullx(j2) - nonnullx(j1)
        IF (ixd.LT.0) ixd = ixd + nx ! "forward difference"
c
         val(ixd) = val(ixd) +
     &         (FLOAT(lc1dsum(nonnullx(j1)))/FLOAT(nt))*
     &         (FLOAT(lc1dsum(nonnullx(j2)))/FLOAT(ncsize))
     &         /FLOAT(ny*nz)
        ENDDO
      ENDDO
c
      DO ixd=0,nx-1
      c1dsum(ixd) = c1dsum(ixd) + val(ixd)
      c1dsm2(ixd) = c1dsm2(ixd) + val(ixd)*val(ixd)
      ENDDO
```

This fragment is extracted from the spin 1 AFHM discrete-time cluster algorithm. Here `ny` is the lattice size in the x_2 direction and `nz` (=2) is the lattice size in the spin layer direction. Also, `ncsize` is the cluster size (the total number of spin sites in the cluster). The contribution to the correlation function (and its square, for computing the standard error) is stored in `c1dsum` (respectively, `c1dsm2`).

Note we choose here to normalize by `1/(ny*nz*nt**2)` and `1/(ncsize/nt)`. The factors of `nt` keep the magnitude finite in the time continuum limit. The `1/ncsize` is required because we compensate as usual for cluster size in the single-cluster algorithm.

Implementation in continuous time

The same idea applies in the continuous-time cluster algorithm, except that the algorithm already accumulates cluster segment lengths – so the time integration is already done. Also, for the antiferromagnetic model, a) the cluster path reverses direction when it moves to a neighboring spin site, and b) neighbor sites have opposite stagger factor, so it is sufficient to store the accumulated segment lengths in the first stage of the computation. The code for implementing the estimator for correlation function is given in entirety in the modules `corraccum1d` and `docorr1d` in the code `LATTMPAS` given in Appendix A, so we decline to present detail here.

Extracting the correlation length

There remains the residual issue of how to extract the correlation length. In a finite volume, due to the periodicity of the boundary conditions, the exponential form of the correlation function is symmetrized to yield a hyperbolic cosine. The direct method of estimating the correlation length is simply to compute the best-fit parameters for a function of the form

$$f(x;A,\xi) = A\cosh\left(\frac{x - L/2a}{\xi}\right)$$

The fit proceeds as a least-squared-error fit, which produces an error estimate together with a χ^2 goodness-of-fit statistic.

However, the estimate for ξ using this procedure is biased. It underpredicts the correlation length by a small amount because the correlation function actually contains higher-mass modes (corresponding to small correlation lengths). While these contaminants can be effectively eliminated by discarding the first few elements of the correlation function (i.e. the correlation function at small separations $\Delta x_1 = 0,1\ldots$), we choose to use an alternative definition of the correlation length, which gives better results [Cooper82].

The so-called second moment method of computing the correlation length uses as input the Fourier transform of the correlation function. The definition we use is

$$\xi = \frac{L}{2\pi}\sqrt{\frac{\tilde{C}(0)}{\tilde{C}(2\pi/L)} - 1}$$

Here the Fourier transform of the correlation function $C(x)$ is simply

$$\tilde{C}(p) = \sum_{x_1} \exp(ipx_1)C(x_1)$$

The first moment, corresponding to $p = 0$, is simply the integral of the correlation function, and simply yields the staggered susceptibility (in fact we use this as a check for the correctness of the algorithm). The second moment, corresponding to the momentum

$p = 2\pi/L$, appears in the definition of ξ. In practice we multiply the correlation function by a cosine and sum the result to get the second moment.

It is a simple exercise to verify that the second moment method gives exactly the same result as the best-fit method for a correlation function that is precisely a hyperbolic cosine. The second-moment method suppresses the higher mass terms and gives a more accurate estimate for ξ.

Comparison with predictions of chiral perturbation theory

Chakravarty, Halperin, and Nelson used perturbative renormalization group arguments to predict the exponential temperature dependence of the correlation length [Chakravarty88 and Chakravarty89]. Hasenfratz and Niedermayer computed the temperature dependence with a first-order correction as well as the specific prefactor that accompanies the functional form, using the techniques of chiral perturbation theory in the asymptotic scaling regime [Hasenfratz91]. The functional form they predict is

$$\xi(\beta) = \frac{e}{8}\frac{\hbar c}{2\pi\rho_s}\exp(2\pi\rho_s\beta)\left[1 - \tfrac{1}{2}(1/2\pi\rho_s\beta) + O\left((1/2\pi\rho_s\beta)^2\right)\right]$$

We refer to this formula as the CH_2N_2 formula. The parameters $\hbar c$ and ρ_s are the same as those defined in Chapter 2. This formula should give the correct temperature dependence of the correlation length for the low temperature regime where asymptotic scaling is expected to set in.

We attempt to validate this formula, using the cluster algorithms we have developed. We discuss the 2-D spin ½ AFHM, for which we have a continuous-time cluster algorithm. The basic strategy is to run simulations for a variety of temperatures, enlarging the volume at each temperature until finite-size effects disappear. Figure 5-1 shows a comparison of CH_2N_2 versus the correlation lengths from the continuous-time cluster algorithm for a range of inverse temperatures up to $\beta = 4$. At the smallest β, a lattice side length $L = 20$ suffices to eliminate finite-size effects. At $\beta = 4$, we explore lattice sizes up to $L = 320$. Clearly the exponential part of the temperature dependence is basically correct.

As an aside, we note that Makivic´ and Ding explored this regime in a 1991 paper, using a conventional Monte Carlo technique [Makivic91]. The current results are consistent with the [Makivic91] results, considering the Trotter error associated with discrete-time techniques. Makivic´ and Ding carried out a reduction for the exponential prefactor, neglecting the linear and higher order corrections that the CH_2N_2 formula asserts. Their estimate is roughly 70% of the correct prefactor, for reasons that will become clear momentarily.

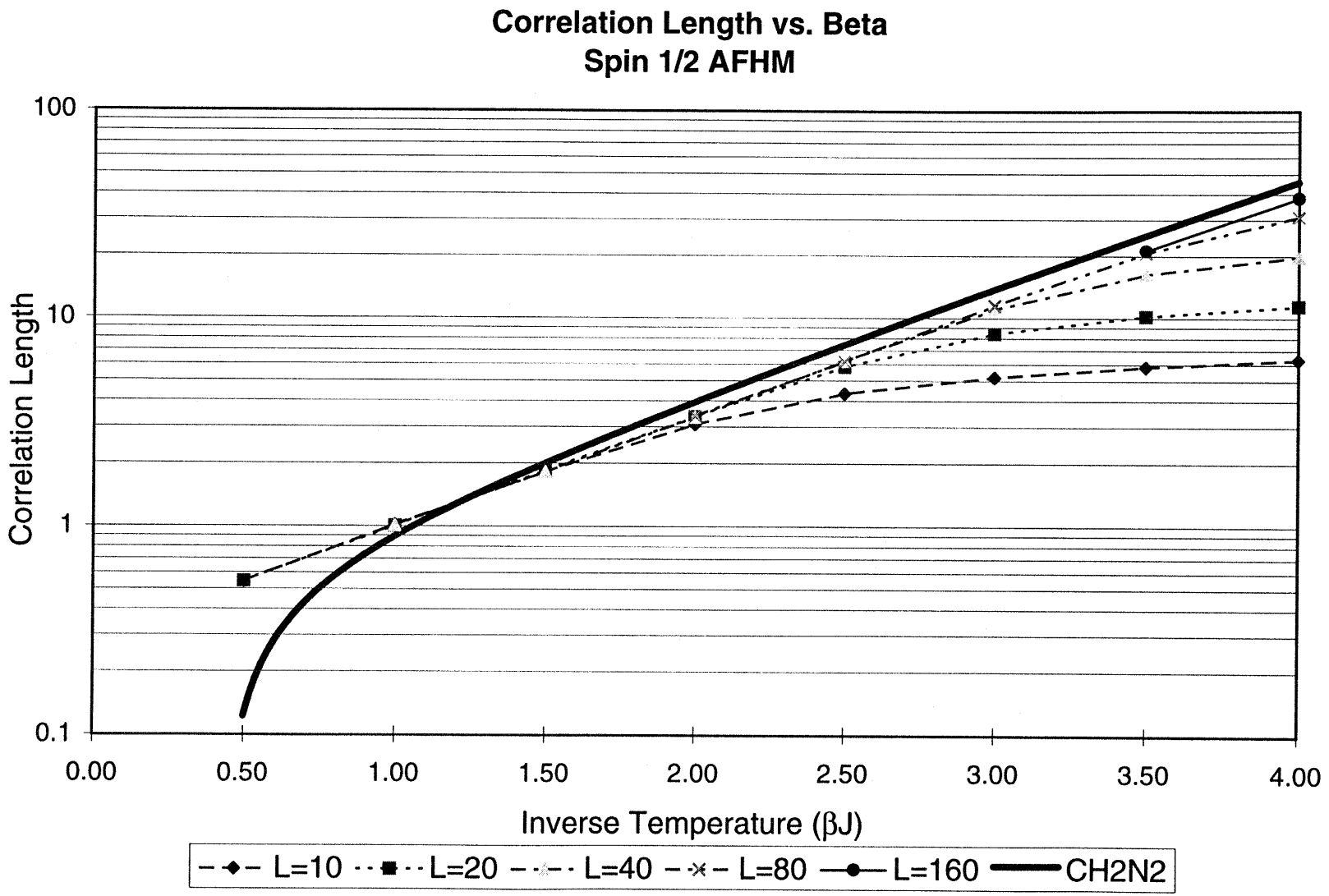


Figure 5-1. Correlation Length Comparison. Heavy solid line is prediction of chiral perturbation theory, other lines are various volumes up to the point where finite-size effects disappear.

Finite Size Scaling of the Correlation Length

We quickly run into a problem with the direct measurement of correlation length. Accurate direct measurement of the correlation length requires working with a lattice of side length $L > 5\xi$. With the correlation length growing exponentially with inverse temperature, it quickly becomes impractical to measure ξ directly. In fact, we are unable to demonstrate consistency with chiral perturbation theory in the range of β available to us if we confine our study to direct measurements alone.

Fortuitously, recent theoretical developments are of direct assistance in this problem. In a series of papers, Drs. Caracciolo, Edwards, Ferreira, Pelissetto, and Sokal (hereinafter, the "CEFPS" collaboration) set forth a technique for extrapolating correlation lengths from finite-size lattices to the infinite-volume limit [Caracciolo95a, Caracciolo95b, Caracciolo95c, Caracciolo95d]. Jae-Kwon Kim has developed a related technique that gives results consistent with CEFPS, but for reasons of space and time we concentrate on the former technique [Kim93,,Kim94a,Kim94b].

In the following discussion we must keep in mind the distinction between the correlation length measured on a lattice of side length L, denoted $\xi(L)$, and the infinite-volume correlation length $\xi \equiv \xi(\infty)$.

The basic insight of CEFPS is that $\xi(2L)/\xi(L)$ is a universal function of $\xi(L)/L$, for a given class of models. The CEFPS collaboration carried out an exhaustive analysis of the three-state 2-D Potts antiferromagnetic model and the $O(3)$ non-linear σ model (which is equivalent to the AFHM). We are indebted to Drs. Caracciolo et al. for providing us the function that they derived from their studies.

Figure 5-2 shows the scaling function from CEFPS as a solid line, and specific points derived from the continuous-time cluster algorithm data presented in Figure 5-1. Clearly we are in basic agreement with the scaling theory of CEFPS, with some possible residual finite-size variation due to the very small lattice sizes with which we started the analysis.

The scaling function can be used to extrapolate finite size data to the infinite-volume limit, by a simple iteration scheme. Figure 5-3 shows the mapping from $\xi(L)/L$ to $\xi(2^n L)/L$, clearly demonstrating the convergence of the iteration scheme.

We use the CEFPS technique to extrapolate correlation length data for $\beta = 4,5,6,7,8$ to the infinite-volume limit. To highlight the deviation of the correlation length from asymptotic scaling (i.e. CH_2N_2), we refer to the plot presented in Figure 5-4. This presentation takes out the exponential part of CH_2N_2 from the correlation length, and plots the result versus $1/2\pi\rho_s\beta$. The CH_2N_2 result is the dotted line of slope -½ shown in the plot.

This plot reflects scaled data for two different starting volumes: $L = 20$ for small β (from 0.5 to 2.0) and $L = 80$ for large β (from 2 to 8). In principle one might expect that starting with any volume would give the same $\xi(\infty)$ if the scaling law is exact, but deviations from exact

scaling are known to occur if the lattice size L is not much greater than the lattice spacing. In practice one finds that starting with the largest practicable volume minimizes the magnification of the error.

On the other hand, for small β, scaling hardly comes into play at all; we might as well have plotted the $\xi(\infty)$ data directly measured for large volumes for those cases. We chose to plot the scaled data for $L = 20$ for small β to make the interpretation of these plotted data more consistent.

There seems to be a hopeful trend in the $\beta = 5$ and 6 extrapolations, but the data at $\beta = 7$ and 8 take a suspicious downturn, albeit with increasing error. The agreement between CPT and these correlation length data is tenuous at best. Clearly more work is indicated. An ongoing collaboration is studying this problem.

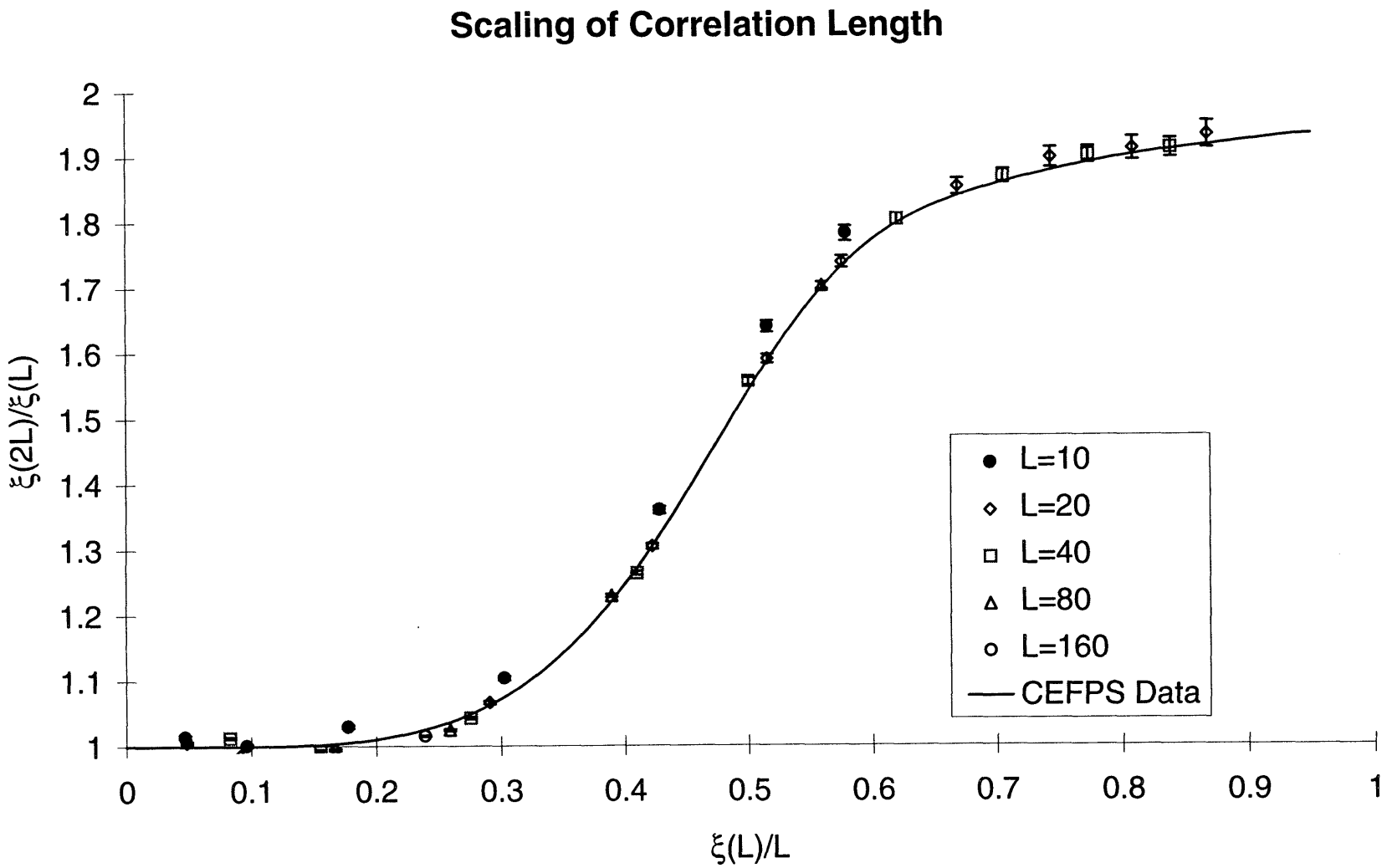


Figure 5-2. Finite-Size Scaling Function. Solid line is from CEFPS; data points are taken from the current study for β in the range 0.5 to 8. Except for L=10, the data generally fall quite close to the CEFPS curve.

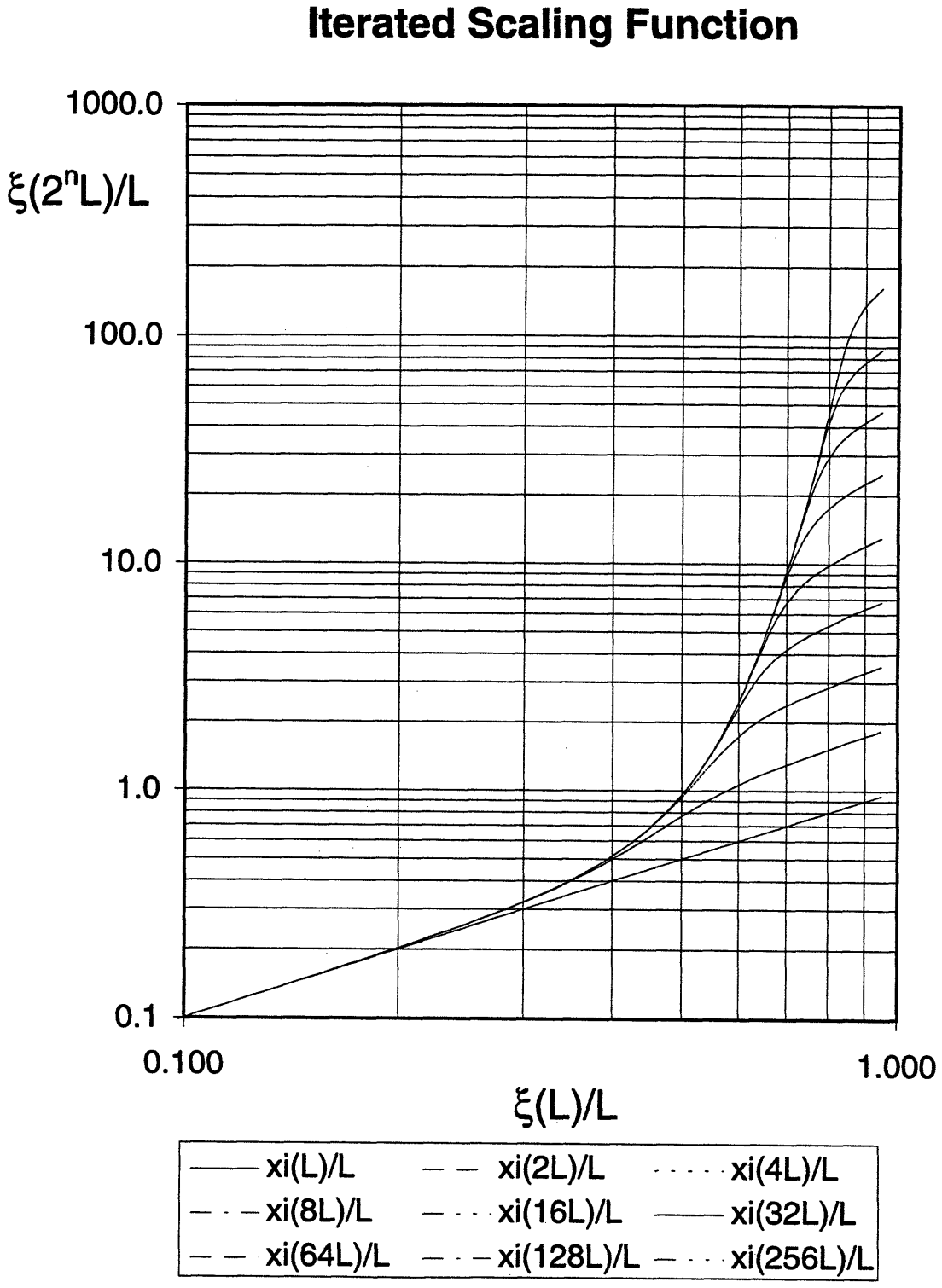


Figure 5-3. Iterated version of the CEFPS scaling function.

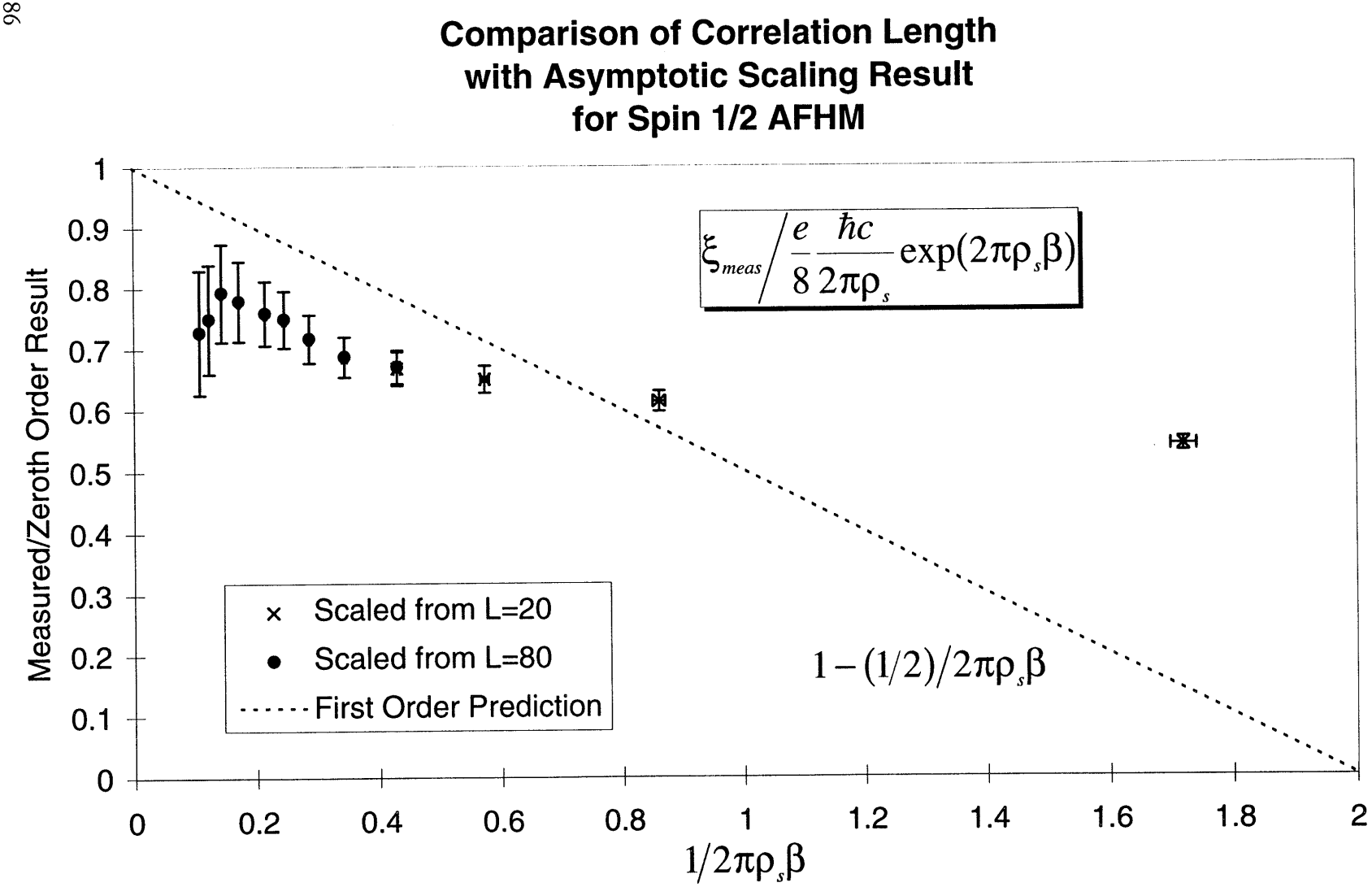


Figure 5-4. Comparison of measured, scaled correlation length with the CH_2N_2 prediction for the asymptotic scaling region. Agreement is tenuous and further simulation at lower temperatures is called for.

"Research is the process of going up alleys to see if they are blind"
– Marston Bates

Chapter 6. The Negative Sign Problem

At this point we consider systems which suffer from the so-called "negative-sign problem" or NSP. In this class fall such venerable puzzles as the Hubbard model, as well as various varieties of "frustrated systems." We will concentrate on the AFHM on the hexagonal lattice, although the significance of the NSP should be understood to encompass many more types of models.

AFHM on a hexagonal lattice

Unlike the situation on the 1-D chain or 2-D square lattice, there is no way to define a (real) stagger factor on a hexagonal lattice. Thus there is apparently no way for a quantum system to develop the staggered order the characterizes the antiferromagnet for bipartite systems. Figure 6-1 displays a typical distribution of spins on a hexagonal lattice. Solid circles are spin up, open circles, spin down.

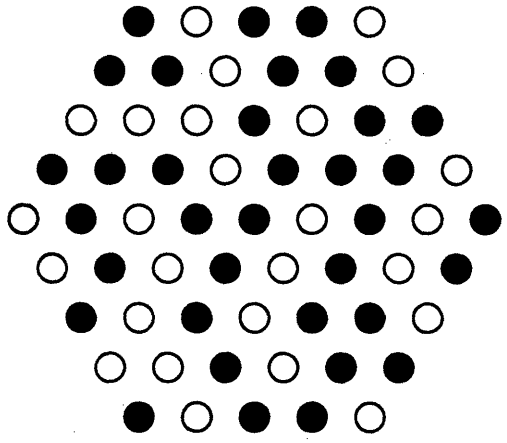


Figure 6-1. Typical set of spin states on a hexagonal lattice

Interestingly, there is an ordered ground state for the *classical* antiferromagnet on a hexagonal lattice. If the spin directions are allowed to lie in the plane, then one can readily demonstrate that there is a repeating pattern where spin directions differ by the maximal $\pm 120°$ for nearest neighbors. In essence the problem with the quantum antiferromagnet on a hexagonal lattice is that the additional x and y component operators are not diagonal in the basis which mimics the classical ground state.

When the basis-change trick doesn't work

So what is the problem with the AFHM on a hexagonal lattice? Recall that the transfer matrix we found for the spin ½ AFHM originally had negative signs in the off-diagonal elements. We had to use a trick to get rid of those minus signs.

Let us recall the trick from Chapter 1 through which we neatly eliminated the minus signs in the transfer matrix . The transfer matrix obtained directly from the Trotter-Suzuki decomposition is

$$M = \exp(-\tfrac{1}{4}\varepsilon\beta J)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}\left(1+e^{\varepsilon\beta J}\right) & \frac{1}{2}\left(1-e^{\varepsilon\beta J}\right) & 0 \\ 0 & \frac{1}{2}\left(1-e^{\varepsilon\beta J}\right) & \frac{1}{2}\left(1+e^{\varepsilon\beta J}\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

For $J > 0$, the off-diagonal elements are negative and cannot be directly interpreted as Boltzmann factors. For the 1-D chain and 2-D square lattices, however, we find that rotating every other spin by 180° resolves this problem, giving a transfer matrix with all real, non-negative elements.

In fact, we could have implemented *any* unitary change of basis – it just happened that $U = \mathrm{diag}(i,-i,i,-i)$ got rid of the minus signs and is equivalent to the staggered 180° rotation. The key observation for the 2-D square lattice is that U has the property $U^4 = 1$, allowing one to return to the same basis when traversing the sides of a unit square, as is necessary to include all nearest-neighbor couplings. (In fact, U incidentally satisfies the more severe constraint $U^2 = 1$).

So the question naturally arises whether there exists some change of basis U which eliminates the minus signs from the off-diagonal elements of M. U must satisfy two additional constraints: all the elements of $UMU^\dagger$ must be real as well as non-negative, and we require $U^3 = 1$. The latter condition guarantees that the change of basis can be propagated consistently throughout the hexagonal lattice.

It is instructive to work through the orbit of $UMU^\dagger$ to see why the basis-change trick doesn't work.

Because we must close a triangular loop when we propagate the change of basis, we want to leave one spin alone and rotate its neighbor. Hence we look for a matrix $U = 1 \otimes u$, where $u \in SU(2)$ acts on one spin. (Since constant factors cancel in the operator transformation $O' = U^\dagger O U$, we will require $\det(u) = 1$.)

One convenient parametrization of $SU(2)$ is

$$u = \begin{pmatrix} \exp(i\alpha)\cos\theta & \exp(i\beta)\sin\theta \\ -\exp(-i\beta)\sin\theta & \exp(-i\alpha)\cos\theta \end{pmatrix}$$

where α, β, and θ are real parameters in the domain $(-\pi, \pi]$.

Some algebra shows that the cube of this matrix has the following elements of $v \equiv u^3$:

$$\begin{aligned}
\mathrm{Re}(v_{1,1}) &= \cos\alpha\cos\theta(4\cos^2\alpha\cos^2\theta - 3) = \mathrm{Re}(v_{2,2}) \\
\mathrm{Re}(v_{1,2}) &= \cos\beta\sin\theta(2\cos\alpha\cos\theta - 1)(2\cos\alpha\cos\theta + 1) = -\mathrm{Re}(v_{2,1}) \\
\mathrm{Im}(v_{1,1}) &= \sin\alpha\cos\theta(2\cos\alpha\cos\theta - 1)(2\cos\alpha\cos\theta + 1) = -\mathrm{Im}(v_{2,2}) \\
\mathrm{Im}(v_{1,2}) &= \sin\beta\sin\theta(2\cos\alpha\cos\theta - 1)(2\cos\alpha\cos\theta + 1) = \mathrm{Im}(v_{2,1})
\end{aligned}$$

We want $u^3 = 1$, so we want all these expressions to be zero except for $\mathrm{Re}(v_{1,1}) = \mathrm{Re}(v_{2,2})$, which we require to be unity. We factor $4x^3 - 3x - 1 = (2x+1)^2(x-1)$, so we conclude that either $\cos\alpha\cos\theta = 1$ or $\cos\alpha\cos\theta = -1/2$. The first case gives the trivial transformation $u = 1$, because the values of $\sin\alpha$ and $\sin\theta$ are forced to be zero.

The second case is non-trivial, although it includes the obvious case $u = \mathrm{diag}(\exp(\pm 2\pi i/3), \exp(\mp 2\pi i/3))$. The condition $\cos\alpha\cos\theta = -1/2$ guarantees that the other terms in u are automatically zero, so the value of β is unrestricted. The locus of parameters for $u^3 = 1$, is, as we expect, a two-parameter manifold, depicted in Figure 6-2. The contours for the function $\cos\alpha\cos\theta$, and the resulting locus for $\cos\alpha\cos\theta = -1/2$ in α, θ, and β space, are as follows.

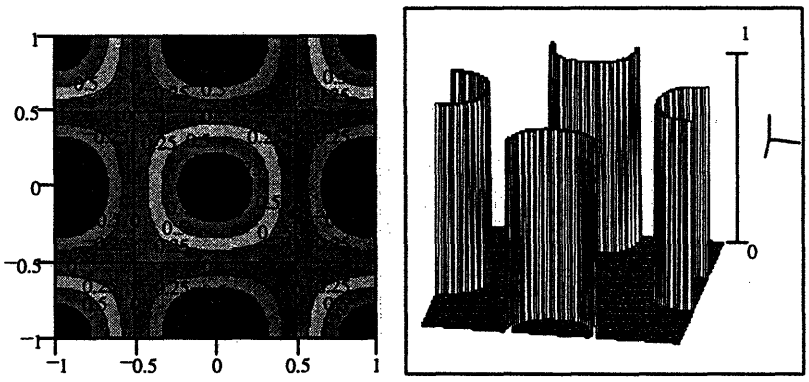


Figure 6-2. Contours for $\cos\alpha\cos\theta$ and the locus $\cos\alpha\cos\theta = -1/2$.

The remaining question is whether any of the transformations $U = 1 \otimes u$ can repair the transfer matrix. Again we write

$$u = \begin{pmatrix} \exp(i\alpha)\cos\theta & \exp(i\beta)\sin\theta \\ -\exp(-i\beta)\sin\theta & \exp(-i\alpha)\cos\theta \end{pmatrix}$$

and we write the shorthand for the transfer matrix

$$M = \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & -z & 0 \\ 0 & -z & y & 0 \\ 0 & 0 & 0 & x \end{pmatrix}$$

where x, y, and z are real, positive expressions that are generally unequal. Some calculation reveals that the matrix $M' = U^{\dagger} M U$ has the following entries:

$$\begin{aligned}
M'_{1,1} &= x\cos^2\theta + y\sin^2\theta = M'_{4,4} \\
M'_{2,2} &= x\sin^2\theta + y\cos^2\theta = M'_{3,3} \\
M'_{1,2} &= (x-y)\cos\theta\sin\theta\exp(-i\alpha + i\beta) = M'^{\,*}_{2,1} = -M'_{3,4} = -M'^{\,*}_{4,3} \\
M'_{1,3} &= z\cos\theta\sin\theta\exp(i\alpha + i\beta) = M'^{\,*}_{3,1} = -M'_{2,4} = -M'^{\,*}_{4,2} \\
M'_{1,4} &= z\sin^2\theta\exp(2i\beta) = M'_{4,1} \\
M'_{2,3} &= -z\cos^2\theta\exp(2i\alpha) = M'_{3,2}
\end{aligned}$$

It is here that the agenda stumbles. The first two conditions are automatically satisfied, but in order for the elements listed in the third and fourth lines to be real and non-negative, we immediately conclude $\cos\theta\sin\theta = 0$.

If $\sin\theta \neq 0$, so that $\cos\theta = 0$ and $\sin^2\theta = 1$, then the condition on $M'_{1,4}$ requires $\exp(2i\beta) = 1$ so $\beta = 0$ or π though α is unrestricted. However, by the reasoning in the first part of this section, the only non-trivial transformations u have $\cos\alpha\cos\theta = -1/2$, so clearly $\cos\theta = 0$ is not in this locus.

Conversely, if $\cos\theta \neq 0$, so that $\sin\theta = 0$ and $\cos^2\theta = 1$, then the condition on $M'_{2,3}$ requires $\exp(2i\alpha) = -1$ so $\alpha = \pm\pi/2$ though β is unrestricted. But $\alpha = \pm\pi/2 \rightarrow \cos\alpha = 0$; again this condition is incompatible with $\cos\alpha\cos\theta = -1/2$.

Therefore we conclude that there is no unitary transformation on the tripartite lattice that will provide us with a transfer matrix which has all real and non-negative elements.

Treating the sign as an observable

We turn to another technique for dealing with the NSP. The idea is to treat the sign as an observable, which we then apply in the evaluation of other observables. We extract the sign from the weight of a configuration by defining a revised action $S'[C]$

$$\exp(-S[C]) = \mathrm{Sign}[C]\exp(-S'[C])$$

where the factor $\exp(-S'[C])$ is the real, positive weighting factor that we want to work with. Observables are expressed in terms of the expectation of the sign

$$\langle A\rangle = \frac{\int DCA[C]\exp(-S[C])}{\int DC\exp(-S[C])} = \frac{\int DCA[C]\mathrm{Sign}(C)\exp(-S'[C])}{\int DC\mathrm{Sign}(C)\exp(-S'[C])} = \frac{\langle A\ \mathrm{Sign}\rangle_{S'}}{\langle \mathrm{Sign}\rangle_{S'}}$$

If we look at the partition functions defined for the two actions S and S', we find that their ratio gives the sign estimator:

$$\frac{Z_S}{Z_{S'}} = \frac{\int DC\exp(-S[C])}{\int DC\exp(-S'[C])} = \frac{\int DC\mathrm{Sign}(C)\exp(-S'[C])}{\int DC\exp(-S'[C])} = \langle \mathrm{Sign}\rangle_{S'}$$

Expectation of sign for the 3-spin system

In order to illustrate the decomposition of the partition function into positive and negative contributions, we analyze explicitly the system composed of 3 spin-½ particles. From an eigenvalue analysis of the 3-spin system the partition function is

$$Z = 4\exp(-\tfrac{3}{4}\beta J) + 4\exp(\tfrac{3}{4}\beta J).$$

It turns out that it is possible to construct explicitly the sign estimator for this simple system. A detailed derivation is given in Appendix C. In this chapter we only quote the result:

$$\langle \mathrm{Sign}\rangle = \frac{Z^+ - Z^-}{Z^+ + Z^-} = \frac{4\exp(\tfrac{3}{4}\beta J) + 4\exp(-\tfrac{3}{4}\beta J)}{2\exp(\tfrac{5}{4}\beta J) + 4\exp(-\tfrac{1}{4}\beta J) + 2\exp(-\tfrac{3}{4}\beta J)}$$

This function starts at unity for $\beta = 0$, and asymptotically goes to $\frac{1}{2}\exp(-\beta J/2)$.

This accords with heuristic considerations. We expect that the sign estimator will basically go as $\langle \mathrm{Sign}\rangle \propto \exp(-\beta V(f - f'))$, where $f - f'$ is the difference in free energy density for the systems with and without the sign included in the action. This in fact is the heart of the problem with the sign estimator – it exponentially goes to zero as the temperature is lowered. Since the estimator is the result of cancellations between configurations of positive and negative sign, it becomes very difficult to extract a sign estimator without the vanishing signal being swamped by statistical uncertainty.

Some approaches to putting Trotter-Suzuki plaquettes on a hexagonal lattice

The general strategy at this point it to plow forward with the Trotter-Suzuki analysis of the AFHM on a hexagonal lattice. Our hope is first that we will be able to find an improved estimator for the sign observable, essentially by collecting and canceling groups of positive and negative configurations via a multi-cluster algorithm. Secondarily we hope to find an algorithm that has a continuum limit, so that we can gain the advantages of eliminating the Trotter error.

Since each spin in the hexagonal lattice has six neighbors, we first look to a decomposition of the AFHM Hamiltonian into six pieces

$$H = H_1 + H_2 + H_3 + H_4 + H_5 + H_6.$$

Modulo rotations and reflections, a single layer in the Trotter-Suzuki sandwich, analogous to the $X_{\mu,k}$ of Chapters 1 and 4, must resemble the pairing scheme shown in Figure 6-3.

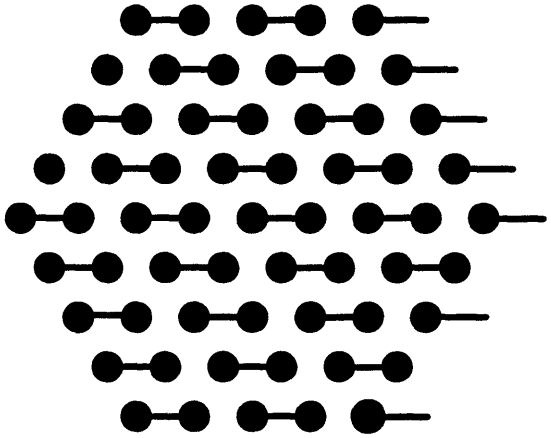


Figure 6-3. Pairing within one layer of the Trotter-Suzuki sandwich on a hexagonal lattice

One stacks six such layers, rotated by multiples of 60°, to form the Trotter-Suzuki sandwich. Some thought reveals that there are actually a number of inequivalent ways to sort the layers, and that for each such sorting, there are 4 distinct types of sites. For visualization of some of the different layerings, you are urged to visit the World-Wide Web site `http://ctpa02.mit.edu/~bbbeard/notes/hexlatt.htm`, where some 256-color GIFs are available.

Potentially this degeneracy is a source of concern. It may happen, for example, that different layerings give slightly different results for observables, due to the order in which a site is paired with its neighbors. This could be disastrous for the sign estimator, where teasing a tiny signal from a noisy background is the name of the game.

An alternative approach – the so-called "prismatic" technique – for splitting up the AFHM Hamiltonian is discussed in Appendix E. The idea behind the prismatic method is to group the spin sites into triangles and treat the triangles as the basic object in the decomposition. This approach has the virtue that the resulting 4-layer sandwich is unique up to rotations and reflections.

A Problem with Ergodicity

It became apparent midway through this study that our intention to put the algorithm of Chapter 1 onto either a 6-layer conventional sandwich or 4-layer prismatic sandwich was confounded by a simple problem with ergodicity. The problem can be explained as follows: essentially the sign of a configuration is just the parity of the number of bonds (forced transition plaquettes) in the configuration. If there are an even number of bonds, the sign is positive; if an odd number, negative. The AFHM cluster algorithm as currently constituted always builds clusters that have an even number of jumps from one spin site to another. Each jump can either create or annihilate a bond. *But any cluster leaves the parity of the number of bonds unchanged.* Hence if the configuration starts in a odd-parity state, it will remain odd for the duration of the simulation, and similarly for even parity.

Potentially there are still some avenues of recourse. The problem with the AFHM cluster algorithm is that the cluster path switches direction whenever it jumps from one site to a neighbor. One can explore ways to modify the action so as to allow jumps that continue in the original time direction, and then manually correct the estimators ex post facto. Our explorations of such strategies was unable to progress very far.

The Galli technique

In a recent preprint, Andrea Galli proposed a technique for eliminating the NSP in a wide variety of Monte Carlo problems [Galli96a]. Galli demonstrated that the technique is able to "cure" the NSP for the trivial fermion model dissected in [Wiese93].

A detailed discussion of the Galli method is beyond the scope of this dissertation. Readers are urged to consult [Galli96a] and [Galli96b]. The core idea is to map a configuration to a partner with (usually) opposite sign, such that the sum of their weights is positive, and to update both configurations simultaneously. The method relies on a clever hashing technique to provide the desired mapping. This is fundamentally different from the improved estimator strategy, and looks very promising.

Conclusions

Regrettably, the explorations described in this chapter were not as fruitful as the rest of the work documented in this treatise. Although some insights were gained, we did not really progress very far in the solution of the negative sign problem. Our hope is that further research, perhaps along the lines suggested by Dr. Galli, will be fruitful.

"The past is of no importance.
The present is of no importance.
It is with the future that we have to deal."
– Oscar Wilde

Chapter 7. Conclusions and Outline for Further Research

Significant progress in the development of loop cluster algorithms has been covered in this dissertation. Not only has a continuous-time cluster algorithm been developed and applied to a problem of current research interest, but also progress has been made in our understanding of cluster algorithms for higher-spin systems. The techniques presented here are gaining the interest of a wider audience. We propose a number of research avenues.

The continuous-time cluster algorithm for higher spin systems

Notably, the current work did not implement the continuous-time cluster algorithm for the spin 1 system. Our intention is to pursue this project as a high priority in the next year. It remains to be seen which translations of the discrete-time cluster algorithm for spin 1 provide the most efficient correct algorithm in continuous time. A target project for the spin 1 AFHM continuous-time cluster algorithm would be to refine the estimates for correlation length in the asymptotic scaling region.

Other applications of the continuous-time cluster algorithm have been suggested recently. Application to a random-coupling spin-chain would be straightforward. Merging this algorithm with a practical NSP-solver would be commendable.

The negative sign problem

The technique proposed by Galli has generated a great deal of interest among those who study lattice systems. Applications of this algorithm to the various lattice models such as the t-J model, the triangular-lattice AFHM, and the Hubbard model are urgently required. Our plan is to attack some model systems with the Galli technique and then move forward onto more significant territory.

Algorithmic improvements

Several issues of a practical nature remain outstanding. The continuous-time cluster algorithm as implemented uses a cyclic double-linked-list storage technique built on a static memory model. Some study of the benefit of implementing a dynamic memory model is warranted. With many Fortran 90 compilers coming on to the market, this task should be tackled in the near term. Other storage techniques, such as hashing functions or tree storage, should be considered to determine if they offer significant time savings. Now that the requirements of the continuous-time algorithm are better understood, an analysis should be performed with the goal of producing an object-oriented code.

"What is now proved was once only imagined."
– William Blake

Epilogue

Advances in both theory and computer systems have made accessible the advances described in this dissertation. If there is a lesson to be learned from all this work, it is that it always benefits us to ask if there is a better way to do what we have been doing. My hope is that I have helped to provide a better way to do the kinds of problems that have traditionally been solved by less elegant means.

"I think C is a disaster"
– Jerry Pournelle

Appendix A. The Code LATTMP

Rendered here is a version of the production code for the continuous-time cluster algorithm for the spin ½ antiferromagnetic Heisenberg model. The code is implemented in FORTRAN 77. As a result, array sizes may need to be manually adjusted, as detailed in the notes below. We plan at some future date to upgrade the code to Fortran 90 to take advantage of dynamic memory management, once F90 compilers become more widely available.

The code base which implements the continuous-time cluster algorithm is named `LATTMP` (M for "multi-spin" and P for "production"). Variants of this code have been implemented for assorted special applications. This version is the latest variant, called `LATTMPAS`, and it is set up to handle asymmetric 2-D lattices. It computes and outputs uniform and staggered susceptibilities, internal energy, correlation function, and correlation length. By the time this dissertation is published, this code should be available from the Web URL

```
http://ctpa02.mit.edu/~bbbeard/lattmp/lattmp.htm
```

Here are some notes on the code:

- There are 5 files that are included in this package. 3 are very small and contain timing code; they must be called "`cpufrag1.for`", "`cpufrag2.for`", and "`cpufrag3.for`". There is also a header file "`lattmpas.hdr`" which must be edited for each run to specify `beta`, number of spins, etc. The main code is in the file "`lattmpas.for`", which should run as is. Note that `lattmpas.for` uses FORTRAN `INCLUDE` statements to grab the other files, so you cannot change the file names without changing the `INCLUDE` statements.

- Although this is "production" code, it has numerous diagnostic routines "plugged in but not turned on". These routines present very little time or space penalty, since they are mostly skipped in compilation, but they do make the code a little harder to read.

- The version of the code provided herein runs without modification (except for the header parameters) on the VAX and the DEC Alpha. and with very slight modification under Windows 3.1 or 95 on a PC. The only platform-specific changes that are required are concealed in the `cpufrag*.for` files. The `cpufrag*.for` files I provide here are for the DEC Alpha.

- Porting to the Cray is slightly tricky; the Cray does not support `ACCESS='APPEND'` keywords in `OPEN` statements so these must be modified. More critically, the Cray CF77 compiler does not support the FORTRAN `STRUCTURE` statement, so all the `path.*`

elements must be converted to separate arrays. I have written a small utility that carries out exactly this conversion. It is available at the LATTMP Web site.

- Array sizes may need to be adjusted manually between runs. Specifically, the parameters "nm" and "nsegmax" in the header file (respectively, the maximum number of transitions in the lattice and the maximum number of segments in a cluster) are empirically determined and we haven't found a way to make them efficiently automatic. The supplied formulas are not too bad but don't work (i.e. aren't big enough) for small beta (<1.0) and are a little wasteful for large beta. See Chapter 2 for some helpful plots of maximum storage space.

- I usually recompile the code between runs. On a DEC Alpha machine running OSF/1, the appropriate FORTRAN command turns out to be

```
f77 -o lattmpas -warn nounreachable lattmpas.for
```

  The "-warn nounreachable" flag simply turns off the error messages about unreachable lines (which occur because the header file contains flags declared as parameters). This command produces the executable "lattmpas".

- The main output file is lattmpas.dat. This is formatted for easy reading. There is also an output file lattmpas.sum, which has a number of variables written out in a single line. The intention is to concatenate the *.sum files and input the data more directly into a spreadsheet or a FORTRAN program.

- Because of the numerical difficulty associated with the improved estimator for internal energy density (described in Chapter 3), this version of the code uses the "unimproved" estimator for energy.

- The hexagonal lattice option does not work AT ALL. Only nntype = 0 or 1 work.

- The times are stored as single precision (REAL*4) so when running on a machine with 32-bit singles (like the VAX, the Alpha, or the PC) the program can be expected to get a little flaky at extremely high beta (>175). I haven't run anything above beta=100 (and have seen no problems) but this is a known shortcoming of finite precision arithmetic. If you are adventuresome you could try changing all the REAL*4 to REAL*8 and running at those betas to see what would happen.

The file `cpufrag1.for`:

```
      REAL*4 dtime,delta,tarray(2) !TIME: Alpha
      EXTERNAL dtime               !TIME: Alpha
```

The file `cpufrag2.for`:

```
      delta = dtime(tarray)        !TIME: Alpha
```

The file `cpufrag3.for`:

```
      delta = dtime(tarray)                                  !TIME: Alpha
      delta = delta/(FLOAT(nloop)/1000.)                     !TIME: Alpha
      WRITE(2,'(1h ,a,f12.4,2(/1h ,a,f12.4))')               !TIME: Alpha
     &'dtime() Time per 1000 configs = ',delta,              !TIME: Alpha
     &'                         user : ',tarray(1)/(FLOAT(nloop)/1000.),
     &'                       system : ',tarray(2)/(FLOAT(nloop)/1000.)
      WRITE(*,'(1h ,a,f12.4,2(/1h ,a,f12.4))')               !TIME: Alpha
     &'dtime() Time per 1000 configs = ',delta,              !TIME: Alpha
     &'                         user : ',tarray(1)/(FLOAT(nloop)/1000.),
     &'                       system : ',tarray(2)/(FLOAT(nloop)/1000.)
```

The file `lattmpas.hdr`:

```
c
c Header file for multi-spin continuous-time cluster algorithm
c
c Everything here should be either a PARAMETER or in a COMMON block
c
c Note the "big" arrays are those with size "nm"
      IMPLICIT NONE
c
c platform-specific format descriptors
      CHARACTER*4 zfmt         ! hex output format descriptor
      PARAMETER (zfmt='Z8.8')  ! Change for DOS ('Z8') <=> VAX ('Z8.8')
      CHARACTER*2 nocr         ! to suppress carriage return
      PARAMETER (nocr=',$')    ! Change for DOS ('\') <=> VAX (',$')
c
c values for simulation parameters
      REAL*4 beta
      PARAMETER (beta=4.0)
      REAL*4 xj
      PARAMETER (xj=1.0)
c
c for defining nearest-neighbor structure
      INTEGER*4 nntype
      PARAMETER (nntype=1)     ! 0=1D chain, 1=2D square, 2=2D hexagonal
      INTEGER*4 nn             ! number of nearest neighbors...
      PARAMETER (nn=2*(1+nntype))     ! ...changes with lattice type
      INTEGER*4 nnsave
      PARAMETER (nnsave=1+nntype*(nntype-1)) ! 0=>1, 1=>1, 2=>3
c
c basic array-dimensioning parameters
      INTEGER*4 nx
      PARAMETER (nx=20) ! side length
      INTEGER*4 ny
```

```
      PARAMETER (ny=20) ! side length
      INTEGER*4 ns
      PARAMETER (ns=nx*ny) ! set ny to 1 manually for 1D chain
      INTEGER*4 nmqs
      PARAMETER (nmqs=beta+1)    ! empirical guess
      INTEGER*4 nm
      PARAMETER (nm=1.25*beta*ns) ! maximum number of transitions
      INTEGER*4 nsegmax
      PARAMETER (nsegmax=3*beta*ns)
c                              ! maximum number of segments
c                              ! this is just a guess... can be fiddled
c
c number of iterations (configurations)
      INTEGER*4 nloop
      PARAMETER (nloop=110000) ! number of configurations
      INTEGER*4 nexclude
      PARAMETER (nexclude=10000)  ! excluded to mask startup transient
c
      INTEGER*4 itick
      PARAMETER (itick=1000)  ! spacing of tickler output
c
c for selecting extended output
      LOGICAL   ext_out
      PARAMETER (ext_out=.FALSE.)
c
c for appending output cumulatively
      LOGICAL   append_mode
      PARAMETER (append_mode=.TRUE.)
c
c for selecting step-by-step execution
      LOGICAL   step_by_step
      PARAMETER (step_by_step=.FALSE.)
c
c for selecting staggered start
      LOGICAL   stag_start
      PARAMETER (stag_start=.FALSE.)
c
c for selecting random start (overrides stag_start)
      LOGICAL   rand_start
      PARAMETER (rand_start=.TRUE.)
c
c to output binning table
      LOGICAL   do_table
      PARAMETER (do_table=.FALSE.)
c
c for random number generator
c change from zero to specify starting seed
      INTEGER*4 inputseed
      PARAMETER (inputseed=0)
      INTEGER*4 iseed      ! "current" seed (used only in init-random)
      INTEGER*4 iseedstart ! stores actual starting seed value
      COMMON /probdata/ iseed,iseedstart
c
c stores sqrt(ns) for nntype=1, sqrt(ns/3) for nntype=2
      INTEGER*4 nside
c
c for storing reference spins
      INTEGER*4 isref(ns)
```

```
c
c for storing transition time data
      INTEGER*4 mtrans(ns)     ! records the number of transitions
      REAL*4    trans(nm)      ! records all the transition times
      INTEGER*4 ibond(nm)      ! records bond information
c
c for implementing a pointer list
      INTEGER*4 iptr(nm,0:1)
      INTEGER*4 iunused(nm)    ! unused pointers
      INTEGER*4 mused          ! total number of used pointers
      INTEGER*4 mfirst(ns)     ! pointer for least time for each spin
c
c for accumulating m statistics
      INTEGER*4 m_accum(0:nm)
      REAL*4    qm_accum(0:nm)
c
c for recording start locations
      INTEGER*4 nstart
      REAL*4    tstart
c
      INTEGER*4 iloop
      INTEGER*4 mused_act ! records actual storage (compare to nm)
      INTEGER*4 nseg_act  ! records actual storage (compare to nsegmax)
c
c for storing decay constant (varies from waypoint to waypoint)
      REAL*4    xlambda
      INTEGER*4 nwinding  ! winding number
c
c suffixes:
c sum = accumulator
c 2sm = accumulator of square
c bar = average
c dev = unbiased std dev
c err = error of the mean = dev/sqrt(n)
c pct = 100*err/bar
      REAL*4 xmag         ! magnetization
      REAL*4 xmagsum      ! magnetization
      REAL*4 xmag2sm      ! magnetization
      REAL*4 xmagbar      ! magnetization
      REAL*4 xmagdev      ! magnetization
      REAL*4 xmagerr      ! magnetization
      REAL*4 xmagpct      ! magnetization
      REAL*4 sus          ! susceptibility
      REAL*4 sussum       ! susceptibility
      REAL*4 sus2sm       ! susceptibility
      REAL*4 susbar       ! susceptibility
      REAL*4 susdev       ! susceptibility
      REAL*4 suserr       ! susceptibility
      REAL*4 suspct       ! susceptibility
      REAL*4 stagsus      ! staggered susceptibility
      REAL*4 stagsussum   ! staggered susceptibility
      REAL*4 stagsus2sm   ! staggered susceptibility
      REAL*4 stagsusbar   ! staggered susceptibility
      REAL*4 stagsusdev   ! staggered susceptibility
      REAL*4 stagsuserr   ! staggered susceptibility
      REAL*4 stagsuspct   ! staggered susceptibility
      REAL*4 ene          ! energy
      REAL*4 enesum       ! energy
```

```
      REAL*4 ene2sm      ! energy
      REAL*4 enebar      ! energy
      REAL*4 enedev      ! energy
      REAL*4 eneerr      ! energy
      REAL*4 enepct      ! energy
      REAL*4 cs          ! clustersize
      REAL*4 cssum       ! clustersize
      REAL*4 cs2sm       ! clustersize
      REAL*4 csbar       ! clustersize
      REAL*4 csdev       ! clustersize
      REAL*4 cserr       ! clustersize
      REAL*4 cspct       ! clustersize
      REAL*4 sl          ! segment length
      REAL*4 slsum       ! segment length
      REAL*4 sl2sm       ! segment length
      REAL*4 slbar       ! segment length
      REAL*4 sldev       ! segment length
      REAL*4 slerr       ! segment length
      REAL*4 slpct       ! segment length
c ### split up ene
      REAL*4 ene_1         ! energy
      REAL*4 ene_1sum      ! energy
      REAL*4 ene_12sm      ! energy
      REAL*4 ene_1bar      ! energy
      REAL*4 ene_1dev      ! energy
      REAL*4 ene_1err      ! energy
      REAL*4 ene_1pct      ! energy
      REAL*4 ene_2         ! energy
      REAL*4 ene_2sum      ! energy
      REAL*4 ene_22sm      ! energy
      REAL*4 ene_2bar      ! energy
      REAL*4 ene_2dev      ! energy
      REAL*4 ene_2err      ! energy
      REAL*4 ene_2pct      ! energy
      REAL*4 ene_3         ! energy
      REAL*4 ene_3sum      ! energy
      REAL*4 ene_32sm      ! energy
      REAL*4 ene_3bar      ! energy
      REAL*4 ene_3dev      ! energy
      REAL*4 ene_3err      ! energy
      REAL*4 ene_3pct      ! energy
c ### end "split up ene" section
c
      REAL*4 trackaccum   ! accumulates a tracking variable
c
c for storing current segment info (varies from segment to segment)
      INTEGER*4 iscurr, isprev, isnext, istatecurr, mcurr
      REAL*4    tcurr, tpost, thoriz
c
      INTEGER*4 neighbor(ns,nn)       ! stores neighbor structure
      INTEGER*4 mlnn(nn)              ! stores waypoint trailing ptrs
      INTEGER*4 istatenn(nn)          ! stores waypoint state
      INTEGER*4 mlnnsave(nnsave)      ! saves pointers from seg to seg
      INTEGER*4 istatennsave(nnsave)  ! saves states from seg to seg
      INTEGER*4 insave(nn)            ! points into mlnnsave, etc.
c
c the following stores .TRUE. if neighbor is masked by path segment
      LOGICAL   mask(nn)              ! segment masking info
```

```
      INTEGER*4 nmask .                ! counts number of masked nbrs
c
      INTEGER*4 new       ! =0 if horizon is old transition
                          ! =1 if horizon is new transition (curr path)
      INTEGER*4 ncannew  ! totals up the number of new's that get hit
c
      INTEGER*4 nactive, nway(nn), nnpick
      REAL*4    tway
c
      COMMON /lattns/   isref,mtrans,mfirst,neighbor
      COMMON /lattnm/   ibond,iptr,iunused
      COMMON /zlattnm1/ m_accum ! odd-size array
      COMMON /lattnn/   mlnnsave,istatennsave,insave,
     &                  mlnn,istatenn,nway,mask
      COMMON /lattint/  nstart,nside,mused,
     &                  iscurr,isprev,isnext,istatecurr,mcurr,
     &                  nactive,nnpick,iloop,nwinding,
     &                  mused_act,nseg_act,nmask,new,ncannew
      COMMON /lattrnm/  trans
      COMMON /zlattr1/  qm_accum ! odd-size array
      COMMON /lattreal/ tstart,xlambda,tcurr,tpost,thoriz,tway,
     &   xmag,   sus,   stagsus,   ene,   cs,   sl,
     &   xmagsum,sussum,stagsussum,enesum,cssum,slsum,
     &   xmag2sm,sus2sm,stagsus2sm,ene2sm,cs2sm,sl2sm,
     &   xmagbar,susbar,stagsusbar,enebar,csbar,slbar,
     &   xmagdev,susdev,stagsusdev,enedev,csdev,sldev,
     &   xmagerr,suserr,stagsuserr,eneerr,cserr,slerr,
     &   xmagpct,suspct,stagsuspct,enepct,cspct,slpct,
     &                  trackaccum
c ### split up ene
      COMMON /splitene/
     &   ene_1,ene_1sum,ene_12sm,ene_1bar,ene_1dev,ene_1err,ene_1pct,
     &   ene_2,ene_2sum,ene_22sm,ene_2bar,ene_2dev,ene_2err,ene_2pct,
     &   ene_3,ene_3sum,ene_32sm,ene_3bar,ene_3dev,ene_3err,ene_3pct
c
c for finding correlation length
      REAL*4 cx1dsum(nx)
      REAL*4 c1dsum(0:nx-1)
      REAL*4 c1dsm2(0:nx-1)
      COMMON /corr1d/ cx1dsum,c1dsum,c1dsm2
      REAL*4 xi
      REAL*4 xierr
      REAL*4 aco
      REAL*4 acoerr
      REAL*4 xichisq
      REAL*4 xinu
      REAL*4 xiqval
      COMMON /xidata/ xi,xierr,aco,acoerr,xichisq,xinu,xiqval
c
c character data for titles
      CHARACTER*9 today
      CHARACTER*8 now
      COMMON /chdata/ today,now
c
c structure for storing path segments
      STRUCTURE /pathseg/
         INTEGER*4 spin   ! records the current spin
         INTEGER*4 state  ! records the initial spin state
```

```
         REAL*4    t(0:1) ! tail (t(0)) and head (t(1)) of segment
         INTEGER*4 segptr ! used to track which segments hit which sites
         INTEGER*4 icross ! if segment crosses t=0: +1=pos, -1=neg dir
         REAL*4    length ! stores length of segment
      END STRUCTURE
c
      INTEGER*4 nseg, isegforce  ! indices for path()
      INTEGER*4 isegptr(ns) ! initial pointers for path().segptr
c
      RECORD /pathseg/ path(nsegmax)
      COMMON /pathrec/ path
      COMMON /pathns/  isegptr
      COMMON /pathint/ nseg,isegforce
c
c for deselecting exceptional cases
      LOGICAL startfault
      LOGICAL lastway
      COMMON /logs/ startfault, lastway
c
c for counting start faults and waypoint faults
      INTEGER*4 num_sf
      COMMON /nsf/ num_sf
c
c allowable number of start faults
      INTEGER*4 nsflim
      PARAMETER (nsflim=100)
```

The file `lattmpas.for`: .

```
c$DEBUG:'d'
c to port to Windows:
c   o use inline metacommand $DEBUG:'d' in place of FORT/D_LINES
c   o change character constants in header file
c
c known platform-specific idiosyncracies:
c   o Hex output descriptor should be Z8 for DOS, Z8.8 for VAX
c   o Suppress-CR descriptor is \ for DOS, ,$ for VAX
c   o DATA statements cannot precede any type declarations under DOS
c
      PROGRAM lattmpas
c
c program to implement the continuous time cluster algorithm for a
c multi-spin system -- PRODUCTION VERSION
c
c *** this version uses unimproved energy estimator ***
c *** this version modified for asymmetric lattices (nx*ny) ***
c
c Features:
c
c   o cyclic double linked list storage
c   o Luscher-Marsaglia-Zaman random number generator
c   o extensible "neighbor structure" for spin lattice
c   o portability enhancements
c   o includes improved estimators for ssceptibility
c     and staggered susceptibility
c
c Notes:
c
c Our spin storage convention is 1 = spin up, 0 = spin down.
c
c Spins are indexed one-dimensionally as is=1..ns but the actual
c topology of the lattice is implemented in subroutine nbr_defn,
c which fills the array neighbor(ns,nn), thus defining how many
c nearest neighbors each spin has and which spins are neighbors to
c which others. The nearest neighbors are indexed in=1..nn, where
c nn is the number of nearest neighbors for each spin. The maximum
c average number of transitions per site is nmqs and the maximum
c total number of transitions in the whole lattice is nm=ns*nmqs.
c
c In the continuous time cluster algorithm, spin states are implicitly
c constant with (euclidean) time, except at specific transition times
c when they change to the opposite direction. Thus each spin has the
c following storage items associated with it:
c
c INTEGER*4 isref(ns)   ! records the spin state at time zero, against
c                       ! which all subsequent states are determined.
c                       ! note it is possible to have a transition at
c                       ! t=0... by convention isref() will record the
c                       ! state PRIOR to the transition (i.e. it will
c                       ! equal the state at t=1 for periodic time).
c                       ! Changed type BYTE to INT*4 for portability.
c INTEGER*4 mtrans(ns)  ! records the number of transitions
c REAL*4    trans(nm)   ! records all the transition times
c INTEGER*4 ibond(nm)   ! records bonding info for each transition
c
```

```
c In addition, we implement the linked list with
c INTEGER*4 iptr(nm,id)   ! pointer to next time or previous time
c                         ! id = 1: forward, id = 0: backward
c INTEGER*4 iunused(nm)   ! array of unused slots
c INTEGER*4 mfirst(ns)    ! index of least time
c
c If this code is eventually adapted for dynamic memory allocation,
c then it should be possible to do away with the array iunused and
c replace it with individually-allocated slots within a structured
c storage unit.
c
c Note this is essentially a generalization of the 2-spin double-
c linked-list data structure with the following changes:
c
c   o Variables mtrans and mfirst gain spin indices and
c     become arrays.
c   o A new array, ibond, records bonding info for each transition.
c     This array is parallel to trans and records the matching spin
c     for each transition of a given spin site.
c   o The insertion pointer mlast has been relieved of duty.
c
c Note in the generalization to many spins, separate cycles of
c transitions for each spin must be maintained, whereas for the
c two-spin system, this information was redundant.
c
c The path itself is stored in a data structure that allows
c segment-by-segment storage of spin index, state, tail and tip
c times, and segment pointer. See the header file.
c
c Assuming a static memory allocation model, a key limitation is the
c maximum storage allocated for remembering transition times (array
c "trans"). We expect that the most probable number of transitions for
c each spin site, parameter "nmqs", will be approximately x/2, where
c x=beta*xj/2. This frequency is half what was expected for the
c two-spin case because we expect on average that only half of the
c neighbors of each site will be available for bonding at any time
c slice. Naively we expect the standard deviation of the transition
c frequency to be SQRT(x/2). Hence if beta*xj = 100, the peak will be
c around nmqs=25 and the tail of the distribution will behave like a
c Gaussian with a std dev of 5. A rule of thumb is to take the nmqs
c equal to beta*xj/2 for large beta*xj. The maximum storage for
c the pointer arrays, etc., is nm=ns*nmqs.
c
c p |               ..
c   |             .    .
c   |           .        .
c   |         .            .
c   |       .                .
c   |     .                    .
c   |.                          ....
c   +----------------^-------------------------
c                    / \ x/2 +/- SQRT(x/2)    m->
c
c The basic idea behind the linked list is that we keep track of
c the order of transition times with a recursive pointer array.
c The sequence of pointers for each spin site closes on itself
c just as the times themselves are cyclic.
c
```

```
c Mental picture:
c  (from the 2-spin code; suppress the spin index here...)
c
c Suppose for a given spin site we have 6 transitions, ordered from
c least to greatest as follows:
c  trans(17)  trans(49)  trans(20)  trans(13)  trans(81)  trans(10)
c
c The pointers corresponding to this configuration would be
c  iptr(17,1) = 49      iptr(17,0) = 10
c  iptr(49,1) = 20      iptr(49,0) = 17
c  iptr(20,1) = 13      iptr(20,0) = 49
c  iptr(13,1) = 81      iptr(13,0) = 20
c  iptr(81,1) = 10      iptr(81,0) = 13
c  iptr(10,1) = 17      iptr(10,0) = 81
c
c Slots in iptr and trans not used by any spin are always set to zero.
c (This is not strictly necessary but will immediately bring to light
c certain coding errors.)
c
c In addition, we keep track of the first times with the index mfirst.
c In this case, mfirst=17.
c
c Generally, the following relations will hold:
c
c     iptr(iptr(mx,0),1) == mx      iptr(iptr(mx,1),0) == mx
c
c     mfirst == iptr(mlast,1)       mlast == iptr(mfirst,0)
c
c where    trans(mfirst) will be the least time
c          trans(mlast) will be the greatest time
c
c     trans(iptr(mx,1))    will follow   trans(mx)  cyclically
c     trans(iptr(mx,0))    will precede  trans(mx)  cyclically
c
c Note we have replaced
c
c     iptr(m) with iptr(m,1) and invptr(m) with iptr(m,0)
c
c and the pointer mlast is not really necessary.
c
c This makes the code slightly less readable but facilitates walking
c the directed segments in the generalization to the multi-spin code.
c
c The principal advantage the double linked list scheme offers is
c that it is easy and quick to insert or delete times in the cycle,
c once the position has been determined. We no longer need to
c shuttle whole banks of times up and down as times are inserted
c and deleted.
c
c The principal disadvantage is that determining a time position can
c no longer be done by bisection, so the computer time it takes to
c find where a particular transition time should go in the cycle
c is proportional to mtrans, not log(mtrans).
c
c A tree storage method might speed this up, but it must be noted
c that tree-balancing would probably be required to maintain
c efficient search patterns.
c
```

```
c Also, we need to keep track of unused slots. We use the array
c iunused() as a stack. The number of used slots is mused, which
c is the sum of mtrans(is) over all the spins. We keep the first
c mused spaces in iunused() empty. Starting with iunused(mused+1), we
c keep a pool of unused slots - in the example above, the pool
c will contain all the numbers from 1 to nm EXCEPT 17,49,20,13,81,10.
c The pool is initialized to iunused(ix) = ix, but it can wind up
c in any order, since slots are added back to the pool randomly.
c (However, the point is that the actual indices don't really matter.)
c
c The header file lattmpas.hdr contains parameter declarations for pseudo-

c global variables as well as type and common block declarations.
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 SECNDS     ! function type declaration
      REAL*4 random_lmz ! function type declaration
c
c some local variables
      INTEGER*4 ix
      INTEGER*4 jx,jx1,jx2,jx3,jx4 ! for bar graph output
      REAL*4 at_begin, at_end, speed
      REAL*4 track
      INCLUDE 'cpufrag1.for' !TIME: Type declarations for timer
      CHARACTER*1 ans
c ###
      LOGICAL*4 dolongene
      COMMON /dole/ dolongene
c
      CHARACTER*10 latttype(0:2)
      DATA latttype / '1-D Chain', '2-D Nx*Ny', '2-D Hex' /
c
      INCLUDE 'cpufrag2.for' !TIME: initializes timer
      at_begin = SECNDS(0.0)
c###
      dolongene = .TRUE.
c
c open output files
      IF (append_mode) THEN
      OPEN (UNIT=2,FILE='lattmpas.dat',STATUS='UNKNOWN',ACCESS='APPEND')
      IF (ext_out) OPEN (UNIT=3,FILE='lattmpas.pth',STATUS='UNKNOWN',
     &                   ACCESS='APPEND')
      ELSE
      OPEN (UNIT=2,FILE='lattmpas.dat',STATUS='NEW')
      IF (ext_out) OPEN (UNIT=3,FILE='lattmpas.pth',STATUS='NEW')
      ENDIF
c
      CALL DATE(today)
      CALL TIME(now)
      WRITE(2,'(1h ,a,T60,a,1x,a/)') 'Program lattmpas',today,now
      WRITE(2,'(1h ,a,I9)')       'Number of iterations   = ',nloop
      WRITE(2,'(1h ,a,I9)')       'Excluded initial cases = ',nexclude
      WRITE(2,'(1h ,a,L1)')       'Extended output = ',ext_out
      WRITE(2,'(/1h ,a,i2,3x,a)') 'Lattice type    =    ',nntype,
     &                            '('//latttype(nntype)//')'
      WRITE(2,'(1h ,a,i5)')       'Nx              = ',nx
      WRITE(2,'(1h ,a,i5)')       'Ny              = ',ny
      WRITE(2,'(1h ,a,i5)')       'Number of spins = ',ns
```

```
      WRITE(2,'(1h ,a,f8.2/)')    'Beta = ',beta
c ###
      IF (dolongene) THEN
      WRITE(2,'(1h ,a/)') 'Energy estimator is: UNIMPROVED'
      ELSE
      WRITE(2,'(1h ,a/)') 'Energy estimator is: IMPROVED'
      ENDIF
c
c call initialization routine
      CALL main_init
      WRITE(2,'(1h ,a,i12/)') 'Seed for Random Numbers =',iseedstart
      IF (ext_out) CALL wrtstep
c
c each turn through outer loop represents one updating of the lattice
      DO iloop=1,nloop
c
c tickler
         IF(MOD(iloop,itick).EQ.0) THEN
c
            IF (iloop.LE.nexclude) THEN
            track = 0.
            ELSE
            track = trackaccum/FLOAT(iloop - nexclude)/(FLOAT(ns)/4.)!###

            ENDIF
c
         WRITE(*,'(/1h ,a,i9,4x,a,i6,4x,a,i6,4x,a,f8.5)')
     &   'iloop = ',iloop,'mused = ',mused,'nseg = ',nseg,
     &   'track = ',track
         jx1 = (80*mused)/nm
         jx2 = (80*mused_act)/nm
         jx3 = (80*nseg)/nsegmax
         jx4 = (80*nseg_act)/nsegmax
c
            IF (jx1.GE.1) THEN
            WRITE(*,'(1h ,80a)') ('*',jx=1,jx1)
            WRITE(*,'(1h ,80a)') ('*',jx=1,jx2)
            ELSE IF (jx2.GE.1) THEN
            WRITE(*,'(/1h ,80a)') ('*',jx=1,jx2)
            ELSE
            WRITE(*,'(//)')
            ENDIF
c
            IF (jx3.GE.1) THEN
            WRITE(*,'(1h ,80a)') ('+',jx=1,jx3)
            WRITE(*,'(1h ,80a)') ('+',jx=1,jx4)
            ELSE IF (jx4.GE.1) THEN
            WRITE(*,'(/1h ,80a)') ('+',jx=1,jx4)
            ELSE
            WRITE(*,'(//)')
            ENDIF
c
         ENDIF
c
      CALL loop_init ! clear path data structure
c
c select a starting point... and map to a point in spin-time space
      nstart = 1 + IFIX(ns*random_lmz())
```

```
      tstart = random_lmz()   ! single-call method thins d.o.f. too much
c
c iteration info
      IF (ext_out) WRITE(2,'(/1h ,a,i9,4x,a,i5,4x,a,f7.4)')
     &'iloop = ',iloop,'nstart = ',nstart,'tstart = ',tstart
c
      CALL genpath ! generate path and update transition table
c
         IF (startfault) THEN
         WRITE(2,'(/1h ,a,i9,a/1h ,a)')
     &   'Warning: skipped iloop = ',iloop,' because it tried to start',
     &   'at an existing transition'
         num_sf = num_sf + 1
         startfault = .FALSE.  ! clear flag
c
            IF (num_sf.GT.nsflim) THEN ! fatal condition
            WRITE(*,'(/1h ,a/)') 'Aborted program '
     &      //'because of excessive number of start faults'
            WRITE(2,'(/1h ,a/)') 'Aborted program '
     &      //'because of excessive number of start faults'
            STOP 'rats'
            ENDIF
c
         ELSE
         CALL measure               ! measure observables
         IF (ext_out) CALL pathdump ! document path segment
         IF (ext_out) CALL wrtstep  ! do output for each step
c
         ENDIF
c
         IF (step_by_step) THEN ! seek insight into operation of program
         CALL paintpath(6)
         WRITE(*,'(/1h ,a''//nocr//'')')
     &   'Proceed to next configuration ? <y>:'
         READ(*,'(a)') ans
c
            IF (ans.EQ.'n' .OR. ans.EQ.'N') THEN
            CALL datadump('Premature Exit Requested')
            STOP 'Premature Exit'
            ENDIF
c
         ENDIF
c
      ENDDO ! start over
c
      CALL writer ! write final output
c
c print estimate of speed, calls to random_lmz()
      at_end = SECNDS(0.0)
      speed = FLOAT(nloop)/(at_end - at_begin)
      WRITE(2,'(/1h ,a,f12.3)') 'Average iterations per second = ',speed
      WRITE(*,'(/1h ,a,f12.3)') 'Average iterations per second = ',speed
c
      INCLUDE 'cpufrag3.for' !TIME: finalizes timer
c
      WRITE(2,'(/1h ,79a/)') ('=',ix=1,79) ! spacer
c
      STOP 'all done'
```

```
      END
c=======================================================================
      SUBROUTINE genpath
c
c generate closed path segment-by-segment
c
c This routine is structured around the calls to subroutines survey
c and nextway.
c
c "survey" looks at the neighborhood at the current time tcurr
c and initializes the pointers and times required to traverse the
c current segment.
c
c "nextway" sets up the next waypoint in the current segment. In so
c doing it updates the pointers for the neighboring spins that
c participate in the waypoint. It also tests for masking by previous
c segments of the path.
c
c Note that the routine "locate" is called for the current spin
c only for the initial segment (nseg=0). Subsequent segments are
c automatically bracketed, and spin states are known, because we keep
c track of all the pointers local to the current spin and time. For
c the initial segment, "locate" is called for the nn nearest neighbors.
c For subsequent current segments, "locate" is called only for "new"
c neighboring spin sites.
c
c Notes on improved estimator for energy:
c
c The energy estimator included in this version is for the
c antiferromagnetic Heisenberg model (J > 0) only, and for 1D chain
c and 2D square lattices. The basic elements of the improved estimator
c are terms for the type of transition (free or forced, new or
c cancelling) and a term for the path segment length times the number
c of masks (which essentially counts revisits to "optional decay"
c plaquettes).
c
c Allocations are as follows:
c
c    For every new free transition,                    add -1/(4*beta)
c    For every cancellation of a new free transition,  add +1/(4*beta)
c    For every cancellation of an existing transition, add -1/(4*beta)
c    For every unit of segment length traversed next
c        to a previous segment of the same path,       add -(J/4) for
c                                                  each masked neighbor
c
c We simpify this calculation somewhat by counting -1/(4*beta) for
c each transition (nseg-1 total) and compensating by adding 1/(2*beta)
c for each cancellation of a new free transition. This last condition
c is determined at the time the horizon is computed; a running total
c of the number of cancelled new transitions is stored in the variable
c "ncannew", which must be reset to zero for each path configuration.
c
c The computation of the segment-length-dependent part of the estimator
c much be carried out waypoint-by-waypoint, since it depends on the
c number of masked channels.
c
c Note these rules only work for antiferromagnetic couplings for
c systems with "site parity", i.e. lattices for which the path
```

```
c building direction switches whenever a jump to adjacent site is
c made, and overall each site can be preassigned a definite parity.
c Other systems must employ more complicated rules.
c
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 random_lmz ! function type declaration
      REAL*4 deltat     ! function type declaration
      LOGICAL isbetween ! function type declaration
c
      LOGICAL closed
      LOGICAL done
      INTEGER*4 mdir
      INTEGER*4 in
      REAL*4 tau
      REAL*4 r
      REAL*4 tx
      REAL*4 signcurr
c
d     WRITE(*,'(1h ,a)') 'Entering GENPATH'
c
c path startup conditions
      closed = .FALSE.
      iscurr = nstart
      tcurr = tstart
c
c note the search direction for the starting segment is set to 1
c (forward); after we locate the trailing bracket pointer, we may need
c to fix it.
      mdir = 1
c
      CALL locate(iscurr,tcurr,mdir,mcurr,istatecurr)
c
      IF (mcurr.GT.0) THEN
c
         IF (trans(mcurr).EQ.tstart) THEN ! start fault
         startfault = .TRUE.
d        WRITE(*,'(1h ,a)') '>Leaving GENPATH'
         RETURN
         ENDIF
c
c revise trailing pointer if spin down
      IF (istatecurr.EQ.0) mcurr = iptr(mcurr,1)
c
      ENDIF
c
c Middle Loop: Segment-by-segment updating until path closes
      DO WHILE (.NOT.closed)
      done = .FALSE.
      signcurr = FLOAT(2*istatecurr - 1) ! maps 1=>1., 0=>-1.
      tpost = tcurr
c
c do initial survey of neighborhood: finds horizon, trailing bracket
c pointers for neighbors....
      CALL survey
c
c can startfault if tstart hits a neighbor's transition
         IF (nseg.EQ.0) THEN
```

```
            DO in=1,nn
               IF (mlnn(in).GT.0) THEN
                  IF (trans(mlnn(in)).EQ.tstart) THEN
                  startfault = .TRUE.
d                 WRITE(*,'(1h ,a)') '>Leaving GENPATH'
                  RETURN
                  ENDIF
               ENDIF
            ENDDO
         ENDIF
c
c Inner Loop: Step towards the horizon
         DO WHILE (.NOT.done)
c
         CALL nextway ! sets up next waypoint
c
c decide whether we need to generate a tau
            IF (nactive.GT.0) THEN
            xlambda = FLOAT(nactive)*beta*ABS(xj)/2.
            tx = tpost
            tau = 0.
c
c this "dowhile" traps truncation errors that arise when tau < 2**(-24)
c note we replaced calls to gettau and tadd with inline calculations...
               DO WHILE (tx.EQ.tpost .AND. tau.NE.1.0)
               r = random_lmz()
c
                  IF (r.LE.EXP(-xlambda)) THEN
                  tau = 1.0
                  ELSE
                  tau = LOG(1./r)/xlambda
                  ENDIF
c
               tx = tpost + signcurr*tau
               tx = AMOD(tx,1.0)
               IF (tx.LT.0.0) tx = tx + 1.0

               ENDDO
c
c HEART OF THE ROUTINE....
c
               IF (tau.LT.1. .AND.
     &              isbetween(tpost,
     &                        tx,
     &                        tway,
     &                        istatecurr)) THEN ! decay towards neighbor
               IF (xj.GT.0.0 .AND. nntype.LE.1) ! energy estimator
     &         ene = ene - (xj/4.)*nmask*deltat(tpost,tx)
c     &                     - 1./(4.*beta) !###free transition
               tpost = tx
c
                  IF (nactive.EQ.1) THEN ! maybe save call to random_lmz
                  nnpick = nway(1)
                  ELSE
                  nnpick = nway(1 + IFIX(nactive*random_lmz()))
                  ENDIF
c
               isnext = neighbor(iscurr,nnpick)
```

```
                done = .TRUE.
c
                ELSE IF (lastway) THEN ! reached horizon
c
                   IF (xj.GT.0.0 .AND. nntype.LE.1) THEN ! energy est.
                   ene = ene - (xj/4.)*nmask*deltat(tpost,thoriz)
c     &                   + (-1./4. + new*(1./2.))/beta ! cnx old?new###
                   ncannew = ncannew + new
                   ENDIF
c
                tpost = thoriz
                done = .TRUE.
c
                ELSE ! reached waypoint, go around again
                IF (xj.GT.0.0 .AND. nntype.LE.1) ! energy estimator
     &          ene = ene - (xj/4.)*nmask*deltat(tpost,tway)
                tpost = tway
c
                ENDIF
c
             ELSE
c
                IF (lastway) THEN ! reached horizon
c
                   IF (xj.GT.0.0 .AND. nntype.LE.1) THEN ! energy est.
                   ene = ene - (xj/4.)*nmask*deltat(tpost,thoriz)
c     &                   + (-1./4. + new*(1./2.))/beta ! cnx old?new###
                   ncannew = ncannew + new
                   ENDIF
c
                tpost = thoriz
                done = .TRUE.
c
                ELSE ! reached waypoint, go around again
                IF (xj.GT.0.0 .AND. nntype.LE.1) ! energy estimator
     &          ene = ene - (xj/4.)*nmask*deltat(tpost,tway)
                tpost = tway
c
                ENDIF
c
             ENDIF
c
          ENDDO
c
c Register the path segment.
c Note: isnext, the next spin, is defaulted to the forced transition
c value by "survey" and is changed above if a decay occurs.
      CALL reg_path
      IF (isnext.EQ.0) closed = .TRUE.
c
c end Middle Loop
      ENDDO
      IF (new.EQ.1) WRITE(*,'(1h ,a)') 'TROUBLE!'
c      ene = ene + 1./(4.*beta) !###
c
d     WRITE(*,'(1h ,a)') '>Leaving GENPATH'
c
      RETURN
```

```
      END
c=======================================================================
      SUBROUTINE survey
c
c routine to survey the neighbors of the current spin, called at
c beginning of visit to a spin site.
c
c The purpose of this routine is to construct a picture of the "local"
c neighborhood of the current spin and time. Note we would like to avoid
c repeated searching of the neighboring spins for bracketing times.
c Such searches are costly in computer time due to the linked-list
c data structure.
c
c The basic steps for each new segment are:
c
c    1) Find next transition on current spin
c    2) Find any intervening path segments on current spin
c    3) Least interval defines the horizon
c    4) Transfer pointer information for already located neighbors
c    5) Perform a "locate" on new neighbors to find trailing pointers
c
c Some observations:
c
c Regardless of the states of the neighboring spin sites, the
c current path segment can extend no further than the next
c forced transition. We call the time of the next forced transition
c the "horizon". There are two ways a transition can be forced:
c (1) The current spin can change state. This transition time is
c already stored in array trans() -- we just have to find the
c bracketing transition times and look in the right direction.
c (2) The current path may have visited the current spin on a
c previous segment. The times are stored in the structure path().
c
c A segment can also be forced to end if it runs into the
c start time and is on the start spin site. This is the path
c closure condition. This condition is flagged by a 0 value stored
c in the pointer "isnext".
c
c The direction in which the current path segment evolves is
c dictated by the current spin state "istatecurr".
c
c As defined above, the nearest forced transition is the "horizon".
c
c This routine also sets up default value for next spin. This
c index "isnext" is set to be the forced transition value,
c or set to zero if horizon is the starting point.
c
c We also need to map out the transition structure of all the
c neighboring spins. Once the current time is bracketed for each
c neighbor, we merely step through successive transition times until
c the horizon is exceeded.
c
c Each transition of a neighboring spin, or endpoint of a visit by
c the current path to a neighboring spin, is called a "waypoint".
c The number of decay channels and thus the decay constant xlambda
c thus changes only at the waypoints.
c
c To save computing time, we pass pointer information from the
```

```
c preceding segment to this routine via the pseudo-globals
c mlnnsave(), istatennsave(), and insave(). These arrays store the
c local spin states and pointers to the trailing bracketing time for
c both the current spin and known neighbors.
c
c Note that the routine "locate" is called for the current spin
c only for the initial segment (nseg=0). Subsequent segments are
c automatically bracketed, and spin states are known, because we keep
c track of all the pointers local to the current spin and time. For
c the initial segment, "locate" is called for the nn nearest neighbors.
c For subsequent current segments, "locate" is called only for "new"
c neighboring spin sites, that is, for the (nn - nnsave) sites that
c were not local to the previous site.
c
c To picture this graphically, consider a hexagonal lattice:
c Suppose 'x' is the current spin, a,b..f are its nearest neighbors,
c and the algorithm produces a transition from x to a at the end
c of the visit to x. Then b, x, and f are now nearest neighbors of a,
c so there is no need to do a "locate" for those sites. For this
c case, nn=6 and nnsave=3, so nn-nnsave=3 new sites must be surveyed.
c
c                                a
c                                ^
c                        b       |       f
c                                |
c                                x
c
c                        c               e
c
c                                d
c
c To transfer this information from segment to segment, we
c fill a small array mlnnsave(nnsave) with the trailing bracket pointers
c for the shared sites, and an array istatennsave() with the states.
c This bookkeeping is done in routine reg_path along with other cleanup
c for the current segment.
c
      INCLUDE 'lattmpas.hdr'
c
      LOGICAL isbetween ! function type declaration
c
      INTEGER*4 in, iseg
c
d     WRITE(*,'(1h ,T10,a)') 'Entering SURVEY'
c
c define previous spin
      IF (nseg.GT.0) THEN
      isprev = path(nseg).spin
      ELSE
      isprev = 0
      ENDIF
c
c put the horizon at the next current-spin transition, if any
      IF (mcurr.NE.0) THEN
      thoriz = trans(iptr(mcurr,istatecurr))
      isnext = ibond(iptr(mcurr,istatecurr)) ! note 0 is stored here if
                                             ! horizon is starting point
c
```

```
      ELSE
      thoriz = tcurr
      isnext = isprev ! bond to previous segment
c
      ENDIF
c
c search the path to find if there is a closer intervening segment....
c
c This is a little subtle, but if a segment intervenes on the current
c spin before the next recorded transition, the tip t(1) is always the
c closer end of the segment. This is essentially because the reason we
c must search the path is solely due to cancellation of a transition
c when a prior segment hits an existing transition....
c
c Note this only needs to be done if the lattice is ferromagnetic or
c does not have "site parity" -- can't happen if all the segments on
c a given site are in the same direction.
      IF (xj.LT.0.0 .OR. nntype.GT.1) THEN
      iseg = isegptr(iscurr) ! points to first segment on iscurr
      isegforce = 0 ! this variable not really needed...
         DO WHILE (iseg.GT.0)
            IF (tcurr.NE.path(iseg).t(1) .AND.
     &          isbetween(tcurr,
     &                    path(iseg).t(1),
     &                    thoriz,
     &                    istatecurr)) THEN
            thoriz = path(iseg).t(1)
            isnext = path(iseg+1).spin  ! must necessarily exist
            isegforce = iseg
            ENDIF
         iseg = path(iseg).segptr ! increment segment pointer
         ENDDO
      ENDIF
c
c determine if the horizon is a new transition, i.e. one on the
c current path -- this is only needed for energy estimator, used in
c antiferromagnetic + (1D chain or 2D square) case
      IF (xj.GT.0.0 .AND. nntype.LE.1) THEN
      new = 0
c      IF (thoriz.EQ.tcurr) WRITE(*,'(1h ,a)') 'BOOM!' ! ###
      IF (thoriz.EQ.tcurr .AND. isnext.NE.0) new = 1 ! special case
      iseg = isegptr(iscurr) ! points to first segment on iscurr
         DO WHILE (iseg.GT.0 .AND. new.EQ.0)
         IF (path(iseg).t(0).EQ.thoriz .AND. isnext.NE.0) new = 1
         iseg = path(iseg).segptr ! increment segment pointer
         ENDDO
      ENDIF
c
c Now search the neighbors to find bracketing times and waypoints.
c
c For second and subsequent segments, we will have at least one
c bracketing time from the previous segment. This information is
c stored in the arrays mlnnsave(), istatennsave(), and insave().
      nnpick = 0
      DO in=1,nn
c
         IF (insave(in).EQ.0) THEN
         CALL locate(neighbor(iscurr,in),tcurr,istatecurr,
```

```
     &               mlnn(in),istatenn(in))
          ELSE
          mlnn(in) = mlnnsave(insave(in))
          istatenn(in) = istatennsave(insave(in))
          ENDIF
c
c initialize the active waypoint arrays - to be filled by "nextway"
       nway(in) = 0
c
c default value of nnpick
       IF (isnext.EQ.neighbor(iscurr,in)) nnpick = in
c
       ENDDO
c
d      WRITE(*,'(1h ,T10,a)') '>Leaving SURVEY'
c
       RETURN
       END
c=======================================================================
       SUBROUTINE nextway
c
c assays the next waypoint and increments pointers, etc.
c
c This routine picks the next waypoint in the current segment. In so
c doing it updates the pointers for the neighboring spins that
c participate in the waypoint. The following pseudo-globals
c are updated here:
c
c    tway      = time of next waypoint
c    nactive   = number of active neighbors
c    nway()    = stores indices of active neighbors
c    mlnn()    = stores trailing bracket pointers to active neighbors
c    istatenn()= stores states of neighbors.
c    lastway   = flag that indicates the next waypoint is the horizon
c
c The steps for each waypoint:
c
c    1) Update pointers to trailing times
c    2) Search the path for nearest segments on each of the neighbors
c    3) Determine the next significant time (the "waytime")
c    4) Determine the number and indices of "open" decay channels (i.e.
c       neighbors with state opposite to the current state and no path
c       segment)
c
       INCLUDE 'lattmpas.hdr'
c
       LOGICAL isbetween ! function type declaration
c
       INTEGER*4 in,iseg
       INTEGER*4 ip
c
       INTEGER*4 ixor(0:1,0:1)
       DATA ixor / 0, 1, 1, 0 /
c
d      WRITE(*,'(1h ,T10,a)') 'Entering NEXTWAY'
c
       lastway = .FALSE.
       nmask = 0
```

```
c
c update the neighbors that participated in the last waypoint.
      DO in=1,nn
c
c refresh pointers as required (should test out null for first
c waypoint)
         IF (mlnn(in).GT.0) THEN
            IF (.NOT.isbetween(trans(mlnn(in)),
     &                         tpost,
     &                         trans(iptr(mlnn(in),istatecurr)),
     &                         istatecurr)) THEN ! tpost has moved
c                                                 ! past interval
            mlnn(in) = iptr(mlnn(in),istatecurr)
            istatenn(in) = 1 - istatenn(in)
            ENDIF
         ENDIF
c
c search the path for intervening path segments
c
      iseg = isegptr(neighbor(iscurr,in))
c
c also determine if segment masks this wayinterval
      mask(in) = .FALSE.
c
         DO WHILE (iseg.GT.0 .AND. .NOT.mask(in))
c
c this innocuous-looking line of code implements an "exclusive or"
c (xor) of istatecurr and path().state, needed to access the
c appropriate end of the segment to compare to trans(ml)...
c i.e. if istatecurr and path().state are the same, t(0) is the
c appropriate lower end; else if istatecurr and path().state are
c opposite, t(1) is the appropriate lower end.
         ip = ixor(istatecurr,path(iseg).state)
c
            IF (isbetween(path(iseg).t(ip),
     &                    tpost,
     &                    path(iseg).t(1-ip),
     &                    istatecurr)) THEN
            mask(in) = .TRUE.
            nmask = nmask + 1
            ENDIF
c
         iseg = path(iseg).segptr ! increment segment pointer
         ENDDO
c
      ENDDO
c
c refresh stage is completed <<<<<<<<<<<<<
c
c now search for active channels and the next waypoint
      tway = thoriz
      nactive = 0
      DO in=1,nn
c
c For anti-ferromagnetic case [ J > 0 ]:
c a neighbor is "active" (available for decay) if it is in the
c opposite spin state from iscurr and is not hosting an existing
c path segment.
```

```
c For ferromagnetic case [ J < 0 ]:
c a neighbor is "active" (available for decay) if it is in the
c same spin state from iscurr and is not hosting an existing
c path segment.
         IF (
     &   (xj.GT.0. .AND. istatecurr.NE.istatenn(in) .AND. .NOT.mask(in))
     &   .OR.
     &   (xj.LT.0. .AND. istatecurr.EQ.istatenn(in) .AND. .NOT.mask(in))
     &   ) THEN
         nactive = nactive + 1
         nway(nactive) = in
         ENDIF
c
c test tway against next transition
         IF (mlnn(in).GT.0) THEN
         IF (isbetween(tpost,
     &                 trans(iptr(mlnn(in),istatecurr)),
     &                 tway,
     &                 istatecurr))
     &                 tway = trans(iptr(mlnn(in),istatecurr))
         ENDIF
c
c test tway against segments
c
c Note that for the neighbor spins, we have the complementary search
c criterion: since existing transitions are picked up in the lines
c above, we need only examine the tail t(0).
      iseg = isegptr(neighbor(iscurr,in))
c
         DO WHILE (iseg.GT.0)
         IF (tpost.NE.path(iseg).t(0) .AND.
     &       isbetween(tpost,path(iseg).t(0),tway,istatecurr))
     &       tway = path(iseg).t(0)
         iseg = path(iseg).segptr ! increment segment pointer
         ENDDO
c
      ENDDO
c
      IF (tway.EQ.thoriz) lastway = .TRUE.
c
d     WRITE(*,'(1h ,T10,a)') '>Leaving NEXTWAY'
c
      RETURN
      END
c=======================================================================
      SUBROUTINE reg_path
c
c routine to register the completed path segment.
c
c Stores spin, times, segment pointer into path() structure. Inserts
c times into transition table.
c
c Note we choose to store segment times in the transition table
c "on the fly", i.e. during the path build, rather than waiting
c until the path closes. This obviates either a) researching the
c transition tables for bracketing pointers, or b) storing and
c "repairing" the bracketing pointers as times are inserted.
      INCLUDE 'lattmpas.hdr'
```

```
c
      LOGICAL isbetween ! function type declaration
c
d     WRITE(*,'(1h ,T10,a)') 'Entering REG_PATH'
c
      IF (nseg.EQ.nsegmax) THEN
      WRITE(2,'(/1h ,a/1h ,a//)')
     &'Fatal Error: Subroutine REG_PATH reached overflow.',
     &'Recommend increasing value of PARAMETER nsegmax.'
      WRITE(*,'(/1h ,a/1h ,a//)')
     &'Fatal Error: Subroutine REG_PATH reached overflow.',
     &'Recommend increasing value of PARAMETER nsegmax.'
      CALL datadump('Overflow of path(nsegmax)')
      STOP 'rats'
      ENDIF
c
      nseg = nseg + 1
c
      path(nseg).spin  = iscurr
      path(nseg).state = istatecurr
      path(nseg).t(0)  = tcurr  ! "tail" of segment
      path(nseg).t(1)  = tpost  ! "tip"  of segment
c
c add pointer to linked list for path
      path(nseg).segptr = isegptr(iscurr)
      isegptr(iscurr) = nseg ! always points to latest segment on site
c
c insert times into transition table
      CALL tinsert(path(nseg).t(0),isprev) ! insert tail first
      CALL tinsert(path(nseg).t(1),isnext) ! insert tip second
c
c record zero crossings - need to handle special case of t=0!
      IF ((isbetween(path(nseg).t(0),
     &               0.,
     &               path(nseg).t(1),
     &               istatecurr) .AND.
     &    .NOT.(istatecurr.EQ.1 .AND. path(nseg).t(0).EQ.0.))
     &    .OR. (istatecurr.EQ.1 .AND. path(nseg).t(1).EQ.0.)) THEN
      path(nseg).icross = 2*istatecurr - 1
      isref(iscurr) = 1 - isref(iscurr) ! change ref state if req'd
      ELSE
      path(nseg).icross = 0
      ENDIF
c
c compute length of segment
      IF (path(nseg).state.EQ.1) THEN
      path(nseg).length =   path(nseg).t(1)
     &                    - path(nseg).t(0)
     &                    + path(nseg).icross
      ELSE
      path(nseg).length = - path(nseg).t(1)
     &                    + path(nseg).t(0)
     &                    - path(nseg).icross
      ENDIF

c
c store pointers for neighborhood
      CALL nbr_save
```

```
c
c move to new spin site
      iscurr = isnext
      tcurr = tpost
c
      IF (nnpick.EQ.0) THEN
      mcurr = 0
      ELSE IF (mlnn(nnpick).EQ.0) THEN
      mcurr = 0
      ELSE IF (xj.GT.0.) THEN ! anti-ferromagnetic case
      mcurr = iptr(mlnn(nnpick),istatecurr)
      ELSE ! ferromagnetic jump can be decay or forced transition
         IF (tcurr.EQ.trans(iptr(mlnn(nnpick),istatecurr))) THEN
         mcurr = iptr(mlnn(nnpick),istatecurr)
         ELSE
         mcurr = mlnn(nnpick)
         ENDIF
      ENDIF
c
c the current state changes only for the anti-ferromagnetic case
      IF (xj.GT.0.) istatecurr = 1 - istatecurr
c
c
d     WRITE(*,'(1h ,T10,a)') '>Leaving REG_PATH'
c
      RETURN
      END
c=======================================================================
      SUBROUTINE nbr_save
c
c routine to store pointers for neighborhood
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 in, inx, isave
c
d     WRITE(*,'(1h ,T20,a)') 'Entering NBR_SAVE'
c
      IF (nntype.EQ.0) THEN
c
c map nnpick = 1 to insave(1|2) = 0|1
c     nnpick = 2 to insave(1|2) = 1|0
      insave(1) = 0 ! clear slot
      insave(2) = 0 ! clear slot
      insave(MOD(nnpick,2)+1) = 1
c
         IF (mcurr.EQ.0) THEN
         mlnnsave(1) = 0
         istatennsave(1) = isref(iscurr)
         ELSE IF (xj.GT.0.) THEN ! anti-ferromagnetic case
            IF (trans(mcurr).EQ.tpost) THEN ! new transition created
            mlnnsave(1) = mcurr
            ELSE                            ! existing annihilated
            mlnnsave(1) = iptr(mcurr,istatecurr)
            ENDIF
         istatennsave(1) = 1 - istatecurr
         ELSE ! ferromagnetic case
         mlnnsave(1) = mcurr
            IF (trans(mcurr).EQ.tpost) THEN ! new transition created
```

```
            istatennsave(1) = istatecurr
            ELSE                              ! existing annihilated
            istatennsave(1) = 1 - istatecurr
            ENDIF
         ENDIF
c
c
      ELSE IF (nntype.EQ.1) THEN
c
c map nnpick = 1 to insave(1|2|3|4) = 0|0|1|0
c     nnpick = 2 to insave(1|2|3|4) = 0|0|0|1
c     nnpick = 3 to insave(1|2|3|4) = 1|0|0|0
c     nnpick = 4 to insave(1|2|3|4) = 0|1|0|0
         DO in=1,nn
         insave(in) = 0 ! clear slot
         ENDDO
c
      insave(MOD(nnpick+1,4)+1) = 1
c
         IF (mcurr.EQ.0) THEN
         mlnnsave(1) = 0
         istatennsave(1) = isref(iscurr)
         ELSE IF (xj.GT.0.) THEN ! anti-ferromagnetic case
            IF (trans(mcurr).EQ.tpost) THEN ! new transition created
            mlnnsave(1) = mcurr
            ELSE                              ! existing annihilated
            mlnnsave(1) = iptr(mcurr,istatecurr)
            ENDIF
         istatennsave(1) = 1 - istatecurr
         ELSE ! ferromagnetic case
         mlnnsave(1) = mcurr
            IF (trans(mcurr).EQ.tpost) THEN ! new transition created
            istatennsave(1) = istatecurr
            ELSE                              ! existing annihilated
            istatennsave(1) = 1 - istatecurr
            ENDIF
         ENDIF
c
c
      ELSE IF (nntype.EQ.2) THEN
c
c map nnpick = 1 to insave(1|2|3|4|5|6) = 0|0|1|2|3|0
c     nnpick = 2 to insave(1|2|3|4|5|6) = 0|0|0|1|2|3
c     nnpick = 3 to insave(1|2|3|4|5|6) = 3|0|0|0|1|2
c     nnpick = 4 to insave(1|2|3|4|5|6) = 2|3|0|0|0|1
c     nnpick = 5 to insave(1|2|3|4|5|6) = 1|2|3|0|0|0
c     nnpick = 6 to insave(1|2|3|4|5|6) = 0|1|2|3|0|0
         DO in=1,nn
         insave(in) = 0 ! clear slot
         ENDDO
c
         DO isave=1,nnsave
         insave(MOD(nnpick+isave,6)+1) = isave
         ENDDO
c
c the indices are abstracted here to circumvent the activist
c intervention of the compiler...
         DO in=1,nnsave
```

```
c
c for nnpick=1, left-hand slot is number 2 position around current spin
            IF (in.EQ.1) THEN
            inx = MOD(nnpick,6)+1
c
               IF (mlnn(inx).EQ.0) THEN
               mlnnsave(in) = 0
               istatennsave(in) = isref(neighbor(iscurr,inx))
               ELSE IF (trans(mlnn(inx)).EQ.tpost) THEN
               mlnnsave(in) = mlnn(inx)
               istatennsave(in) = istatenn(inx)
               ELSE
               mlnnsave(in) = iptr(mlnn(inx),istatecurr)
               istatennsave(in) = istatenn(inx)
               ENDIF
c
c middle slot is always the current spin
            ELSE IF (in.EQ.2) THEN
c
               IF (mcurr.EQ.0) THEN
               mlnnsave(in) = 0
               istatennsave(in) = isref(iscurr)
               ELSE IF (xj.GT.0.) THEN ! anti-ferromagnetic case
                  IF (trans(mcurr).EQ.tpost) THEN ! new transition
                  mlnnsave(1) = mcurr
                  ELSE                              ! existing annihilated
                  mlnnsave(1) = iptr(mcurr,istatecurr)
                  ENDIF
               istatennsave(1) = 1 - istatecurr
               ELSE ! ferromagnetic case
               mlnnsave(1) = mcurr
                  IF (trans(mcurr).EQ.tpost) THEN ! new transition
                  istatennsave(1) = istatecurr
                  ELSE                              ! existing annihilated
                  istatennsave(1) = 1 - istatecurr
                  ENDIF
               ENDIF
c
c for nnpick=1, right-hand slot is number 6 position around current spin
            ELSE IF (in.EQ.3) THEN
            inx = MOD(nnpick+4,6)+1
c
               IF (mlnn(inx).EQ.0) THEN
               mlnnsave(in) = 0
               istatennsave(in) = isref(neighbor(iscurr,inx))
               ELSE IF (trans(mlnn(inx)).EQ.tpost) THEN
               mlnnsave(in) = mlnn(inx)
               istatennsave(in) = istatenn(inx)
               ELSE
               mlnnsave(in) = iptr(mlnn(inx),istatecurr)
               istatennsave(in) = istatenn(inx)
               ENDIF
c
            ENDIF
c
         ENDDO
c
      ENDIF
```

```
c
d     WRITE(*,'(1h ,T20,a)') '>Leaving NBR_SAVE'
c
      RETURN
      END
c=======================================================================
      SUBROUTINE locate(is,tx,mdir,ml,istate)
c
c routine that searches the table of transition times for bracketing
c times...
c
c The purpose of this routine is basically to find the "ml" and
c "mu" that are the "lower" and "upper" (or better, "trailing" and
c "leading") indices, respectively, of the times that bracket "tx". Only
c the trailing bracket pointer ml is returned, since the leading pointer
c is available from the pointer table iptr(). The routine also returns
c the spin state at tx as the variable istate.
c
c The search direction is specified with the flag mdir = 0 or 1:
c
c 1 : searches "forwards"  <==> trans(ml) precedes tx precedes trans(mu)
c 0 : searches "backwards" <==> trans(ml) follows  tx follows  trans(mu)
c
c This is similar to the code for the 2-spin case, except that the
c target spin and the time are passed explicitly, and the index ml
c is returned explicitly. This allows us to loop over the neighbors
c of the current spin to define the neighborhood.
c
c Also, the mtrans() = 0 condition is handled gracefully by returning
c a zero value for ml.
c
c This routine has been rehacked for linked list pointers.
c Unfortunately, we have given up our lovely bisection technique
c and replaced it with an ugly linear search...
c
c Note we adhere to an allocation convention that assumes a closed
c interval on the lower bound and an open interval on the upper...
c i.e. we basically would like ( trans(ml) .LE. tx .LT. trans(mu) )
c except that .LE. and .LT must take into account both the wrap-around
c in time and the duality of the search direction....
c
c The check for start faults has been moved out of this module.
      INCLUDE 'lattmpas.hdr'
c
      LOGICAL isbetween ! function type declaration
c
      INTEGER*4 is, mdir, ml, istate
      REAL*4 tx
c
d     WRITE(*,'(1h ,T20,a)') 'Entering LOCATE'
c
c wrap-around
      istate = isref(is)
c
      IF (mtrans(is).EQ.0) THEN
      ml = 0
c
      ELSE
```

```
c
         IF (mdir.EQ.1) THEN
         ml = mfirst(is)
         ELSE
         ml = iptr(mfirst(is),0) ! backwards search starts with mlast
         ENDIF
c
      istate = 1 - istate
c
c increment interval until we satisfy the bracketing condition
c note isbetween() is true if:
c
c           mdir=1 and trans(ml).LE.tx.AND.tx.LT.trans(mu)
c        or mdir=0 and trans(ml).GE.tx.AND.tx.GT.trans(mu)
c
c where mu=iptr(ml,mdir) and relationals are "wrap-around"
c
         DO WHILE (.NOT.isbetween(trans(ml),
     &                            tx,
     &                            trans(iptr(ml,mdir)),
     &                            mdir))
         ml = iptr(ml,mdir)
         istate = 1 - istate
         ENDDO
c
      ENDIF
c
d     WRITE(*,'(1h ,T20,a)') '>Leaving LOCATE'
c
      RETURN
      END
c=======================================================================
      SUBROUTINE tinsert(tx,ibx)
c
c routine to put a new time into the linked list for iscurr
c
c this module updates trans,ibond,iptr,mfirst,mtrans,iunused,mused
c
c also tests for deletion condition; calls tdelete, which changes
c the value of pointer mcurr
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 tx
      INTEGER*4 ibx
c
      INTEGER*4 mx
c
d     WRITE(*,'(1h ,T30,a)') 'Entering TINSERT'
c
c check for deletion conditions
      IF (mcurr.GT.0) THEN
c
         IF (tx.EQ.trans(mcurr)) THEN
         CALL tdelete
d        WRITE(*,'(1h ,T30,a)') '>Leaving TINSERT'
         RETURN
         ELSE IF (tx.EQ.trans(iptr(mcurr,istatecurr))) THEN
         mcurr = iptr(mcurr,istatecurr)
```

```
         CALL tdelete
d        WRITE(*,'(1h ,T30,a)') '>Leaving TINSERT'
         RETURN
         ENDIF
c
      ENDIF
c
c check for overflow
      IF (mused.LT.nm) THEN ! space available
c
c get unused index
      mx = iunused(mused+1)
      iunused(mused+1) = 0
c
         IF (mtrans(iscurr).EQ.0) THEN ! mcurr must also be zero; ignore
         mfirst(iscurr) = mx
         iptr(mx,1)     = mx
         iptr(mx,0)     = mx
c
         ELSE
c
c check to see if mfirst pointer is affected
         IF (tx.LT.trans(mfirst(iscurr))) mfirst(iscurr) = mx

            IF (mtrans(iscurr).EQ.1) THEN
c
c special case: mx and mcurr point to each other
            iptr(mcurr,1) = mx
            iptr(mcurr,0) = mx
            iptr(mx,1) = mcurr
            iptr(mx,0) = mcurr
c
            ELSE
c
c normal insertion, transfer pointers:
c
c order of slots is:
c                                insert
c                                vvvvvv
c   before         mcurr           mx      iptr(mcurr,istatecurr)
c   after  iptr(mx,1-istatecurr)   mx      iptr(mx,istatecurr)
c                    ^                            ^
c                    |                            |
c                    \----------------------------/
c
            iptr(mx,istatecurr)    = iptr(mcurr,istatecurr)
            iptr(mcurr,istatecurr) = mx
            iptr(mx,1-istatecurr)  = mcurr
            iptr(iptr(mx,istatecurr),1-istatecurr) = mx
c
            ENDIF
c
         ENDIF
c
      trans(mx) = tx
      ibond(mx) = ibx
      mtrans(iscurr) = mtrans(iscurr) + 1
      mused = mused + 1
```

```
      mcurr = mx
c
c write cryptic warning message
      ELSE
      WRITE(2,'(/1h ,a/1h ,a//)')
     &'Fatal Error: Subroutine TINSERT reached overflow.',
     &'Recommend increasing value of PARAMETER nm.'
      WRITE(*,'(/1h ,a/1h ,a//)')
     &'Fatal Error: Subroutine TINSERT reached overflow.',
     &'Recommend increasing value of PARAMETER nm.'
      STOP 'rats'
c
      ENDIF
c
d     WRITE(*,'(1h ,T30,a)') '>Leaving TINSERT'
c
      RETURN
      END
c=======================================================================
      SUBROUTINE tdelete
c
c routine to remove the current element mcurr from the linked list
c for iscurr
c
c this module updates mcurr,trans,ibond,iptr,mfirst,mtrans,iunused,mused
c
c assumes that mtrans(iscurr).GT.0 (otherwise how could you try to
c delete mcurr?)
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 msave
c
d     WRITE(*,'(1h ,T40,a)') 'Entering TDELETE'
c
c handle cleanup of mfirst pointer...
c only needs to be changed if mcurr=mfirst
      IF (mcurr.EQ.mfirst(iscurr)) THEN
c
         IF (mtrans(iscurr).GE.2) THEN
         mfirst(iscurr) = iptr(mfirst(iscurr),1) ! skip to next slot
         ELSE IF (mtrans(iscurr).EQ.1) THEN
         mfirst(iscurr) = 0                      ! no slots left
         ENDIF
c
      ENDIF
c
c transfer pointers:
c                                            delete
c                                            vvvvvv
c
c      order of slots is:     iptr(mcurr,0)    mcurr      iptr(mcurr,1)
c                                    ^                          ^
c                                    |                          |
c            (skip over mcurr)        \--------------------------/
c
      msave = iptr(mcurr,1-istatecurr)       ! store new trailing ptr
      iptr(iptr(mcurr,0),1) = iptr(mcurr,1)  ! transfer forward pointer
      iptr(iptr(mcurr,1),0) = iptr(mcurr,0)  ! transfer backward pointer
```

```
      iptr(mcurr,1) = 0                      ! clear memory slot
      iptr(mcurr,0) = 0                      ! clear memory slot
      trans(mcurr) = 0.                      ! clear memory slot
      ibond(mcurr) = 0                       ! clear memory slot
      iunused(mused) = mcurr                 ! put mcurr back into pool
      mtrans(iscurr) = mtrans(iscurr) - 1    ! decrement counter
      mused = mused - 1                      ! decrement counter
c
c replace pointer with preceding pointer, if it's still around.
      IF (mtrans(iscurr).GT.0) THEN
      mcurr = msave
      ELSE
      mcurr = 0
      ENDIF
c
d     WRITE(*,'(1h ,T40,a)') '>Leaving TDELETE'
c
      RETURN
      END
c=======================================================================
      LOGICAL FUNCTION isbetween(a,b,c,mdir)
c
c low-level routine to determine whether a,b,c are in cyclic order
c or reverse order.
c
c Input: times a, b, c and search direction (rotation sense) mdir
c
c Returns .TRUE. if
c            mdir=1 and  a ".LE." b ".LT." c in cyclic sense
c         or mdir=0 and  a ".GE." b ".GT." c in cyclic sense
c
c Note boundary condition: always returns .TRUE. if b=a.
c
      IMPLICIT NONE
c
      REAL*4 a, b, c
      INTEGER*4 mdir
c
      isbetween = .TRUE. ! convention: a=c always returns .TRUE.
c
      IF (a.LT.c) THEN         ! "regular order"
      isbetween = (mdir.EQ.1 .AND. (a.LE.b .AND. b.LT.c)) .OR.
     &            (mdir.EQ.0 .AND. (a.GE.b .OR. b.GT.c))
c
      ELSE IF (a.GT.c) THEN    ! "wrap-around order"
      isbetween = (mdir.EQ.1 .AND. (a.LE.b .OR. b.LT.c)) .OR.
     &            (mdir.EQ.0 .AND. (a.GE.b .AND. b.GT.c))
c
      ENDIF
c
      RETURN
      END
c=======================================================================
      REAL*4 FUNCTION deltat(t0,t1)
c
c routine to compute the (modular) difference of two (periodic)
c time values -- used by energy improved estimator
c
```

```
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 t0,t1
c
      IF (istatecurr.EQ.1) THEN
      deltat = t1 - t0
      ELSE
      deltat = t0 - t1
      ENDIF
c
      IF (deltat.LE.0.0) deltat = deltat + 1.0
c
      RETURN
      END
c=======================================================================
      SUBROUTINE main_init
c
c initializes arrays and flags
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 SECNDS     ! function type declaration
c
      INTEGER*4 is, im, id
c
c initialize counter for start faults
      num_sf = 0
      startfault = .FALSE.  ! clear flag
      iloop = 0
      nseg = 0
c
      xmag         = 0.0
      xmagsum      = 0.0
      xmag2sm      = 0.0
      xmagbar      = 0.0
      xmagdev      = 0.0
      xmagerr      = 0.0
      xmagpct      = 0.0
      sus          = 0.0
      sussum       = 0.0
      sus2sm       = 0.0
      susbar       = 0.0
      susdev       = 0.0
      suserr       = 0.0
      suspct       = 0.0
      stagsus      = 0.0
      stagsussum   = 0.0
      stagsus2sm   = 0.0
      stagsusbar   = 0.0
      stagsusdev   = 0.0
      stagsuserr   = 0.0
      stagsuspct   = 0.0
      ene          = 0.0
      enesum       = 0.0
      ene2sm       = 0.0
      enebar       = 0.0
      enedev       = 0.0
      eneerr       = 0.0
      enepct       = 0.0
```

```
      cs            = 0.0
      cssum         = 0.0
      cs2sm         = 0.0
      csbar         = 0.0
      csdev         = 0.0
      cserr         = 0.0
      cspct         = 0.0
      sl            = 0.0
      slsum         = 0.0
      sl2sm         = 0.0
      slbar         = 0.0
      sldev         = 0.0
      slerr         = 0.0
      slpct         = 0.0
c ### split up ene
      ene_1         = 0.0
      ene_2         = 0.0
      ene_3         = 0.0
      ene_1sum      = 0.0
      ene_2sum      = 0.0
      ene_3sum      = 0.0
      ene_12sm      = 0.0
      ene_22sm      = 0.0
      ene_32sm      = 0.0
      ene_1bar      = 0.0
      ene_2bar      = 0.0
      ene_3bar      = 0.0
      ene_1dev      = 0.0
      ene_2dev      = 0.0
      ene_3dev      = 0.0
      ene_1err      = 0.0
      ene_2err      = 0.0
      ene_3err      = 0.0
      ene_1pct      = 0.0
      ene_2pct      = 0.0
      ene_3pct      = 0.0
c
      trackaccum = 0.0
c
      DO is=1,ns
      isref(is) = 1   ! initialize to all spin up
      mtrans(is) = 0  ! no transitions
      mfirst(is) = 0
      ENDDO
c
      mused = 0
      mused_act = 0
      nseg_act  = 0
c
c initialize transition times and pointers
      DO im=1,nm
      trans(im) = 0.0
      ibond(im) = 0
      iunused(im) = im
         DO id=0,1
         iptr(im,id) = 0
         ENDDO
      ENDDO
```

```
c
      IF (do_table) THEN
         DO im=0,nm
         m_accum(im) = 0
         qm_accum(im) = 0.0
         ENDDO
      ENDIF
c
c set up seed for random number generator...
c SECNDS is a VAX system call that returns a REAL*4 containing
c the time since midnight accurate to .01 seconds.
      IF (inputseed.EQ.0) THEN
      iseed = IFIX(100*SECNDS(0.0))
      IF (MOD(iseed,2).EQ.0) iseed = iseed + 1  ! make iseed odd
      ELSE
      iseed = inputseed
      ENDIF
      iseedstart = iseed
      CALL init_random(iseed)
c
c define nearest neighbors
      CALL nbr_defn
c
c optional start configurations
      IF (rand_start) THEN
      CALL sprinkle
      ELSE IF (stag_start) THEN
      CALL stagger
      ENDIF
c
      RETURN
      END
c=======================================================================
      SUBROUTINE loop_init
c
c initializes the path data structure, etc.
c NOTE this is an "inner loop" routine so we must be careful to conserve
c cycles here....
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 is, iseg, in
c
      IF (nseg.EQ.0) nseg = nsegmax ! init entire array on first pass
      DO iseg=1,nseg ! usually clear only the slots that have been used
      path(iseg).spin   = 0
      path(iseg).state  = 0
      path(iseg).t(0)   = 0.0
      path(iseg).t(1)   = 0.0
      path(iseg).segptr = 0
      path(iseg).icross = 0
      path(iseg).length = 0.0
      ENDDO
c
      nseg = 0
      ncannew = 0
      ene = 0.0
      ene_1 = 0.0
      ene_2 = 0.0
```

```
      ene_3 = 0.0
c
      DO is=1,ns
      isegptr(is) = 0 ! couldn't think of a more efficient way to clear
      ENDDO
c
c initialize neighbor storage
      DO in=1,nn
      mlnn(in) = 0
      istatenn(in) = 0
      insave(in) = 0
      mask(in) = .FALSE.
      nway(in) = 0
      ENDDO
c
      nactive = 0
      nnpick = 0
      tway = 0.0
c
      DO in=1,nnsave
      mlnnsave(in) = 0
      istatennsave(in) = 0
      ENDDO
c
      RETURN
      END
c=======================================================================
      SUBROUTINE nbr_defn
c
c defines nearest-neighbor structure of lattice
c
c *** modified for asymmetric lattices ***
c
c nntype=0 means 1D chain
c nntype=1 means square 2D lattice
c nntype=2 means hexagonal 2D lattice
c
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 is, isd, in, ind
      INTEGER*4 ix, jx, ixd, jxd
      INTEGER*4 indisq(4), indjsq(4)
      INTEGER*4 indihex(6), indjhex(6)
      DATA indisq /  1,  0, -1,  0 /
      DATA indjsq /  0,  1,  0, -1 /
      DATA indihex /  1,  1,  0, -1, -1,  0 /
      DATA indjhex /  0,  1,  1,  0, -1, -1 /
c
      IF (nntype.EQ.0) THEN ! 1-D chain
c
         DO in=1,nn
         ind = 2*in - 3 ! maps in=1 to -1, in=2 to +1
c
            DO is=1,ns
            isd = is + ind
c
               IF (isd.LT.1)  isd = ns  ! wrap backward
               IF (isd.GT.ns) isd = 1   ! wrap forward
```

```
c
            neighbor(is,in) = isd
            ENDDO
c
         ENDDO
c
       ELSE IF (nntype.EQ.1) THEN ! 2-D square
c
         IF (MOD(nx,2).NE.0 .OR. MOD(ny,2).NE.0) THEN ! non-fatal error
         WRITE(*,'(/1h ,a/)') 'WARNING: lattice dimensions '
     &   //'should be even integers!'
         WRITE(2,'(/1h ,a/)') 'WARNING: lattice dimensions '
     &   //'should be even integers!'
         ENDIF
c
         DO in=1,nn
c
            DO ix=1,nx
c
               DO jx=1,ny
c
               ixd = ix + indisq(in)
               IF (ixd.LT.1)     ixd = nx     ! wrap backward
               IF (ixd.GT.nx)    ixd = 1      ! wrap forward
c
               jxd = jx + indjsq(in)
               IF (jxd.LT.1)     jxd = ny     ! wrap backward
               IF (jxd.GT.ny)    jxd = 1      ! wrap forward
c
               is = ix + nx*(jx - 1)
               isd = ixd + nx*(jxd - 1)
               neighbor(is,in) = isd
               ENDDO
c
            ENDDO
c
         ENDDO
c
       ELSE IF (nntype.EQ.2) THEN ! 2-D hexagonal
       nside = IFIX(SQRT(FLOAT(ns)/3.)+0.5) ! round to nearest integer
c
         IF (3*nside*nside.NE.ns) THEN ! fatal eror
         WRITE(*,'(/1h ,a/)')
     &   'ERROR: number of lattice sites for hex lattice '
     &   //'should be 3*L**2 for some L'
         STOP 'rats'
         ENDIF
c
         DO in=1,nn
c
            DO ix=1,nside
c
               DO jx=1,3*nside
c
               ixd = ix + indihex(in)
               jxd = jx + indjhex(in)
c
c note that "i-wrap" on hex lattice involves a "j-shift"
```

```
               IF (ixd.LT.1) THEN           ! wrap backward
               ixd = ixd + nside
               jxd = jxd - nside
               ELSE IF (ixd.GT.nside) THEN  ! wrap forward
               ixd = ixd - nside
               jxd = jxd + nside
               ENDIF
c
               IF (jxd.LT.1)       jxd = jxd + 3*nside  ! wrap backward
               IF (jxd.GT.3*nside) jxd = jxd - 3*nside  ! wrap forward
c
               is  = ix  + nside*(jx  - 1)
               isd = ixd + nside*(jxd - 1)
               neighbor(is,in) = isd
               ENDDO
c
            ENDDO
c
         ENDDO
c
      ELSE ! fatal error
      WRITE(*,'(/1h ,a/)')
     &'ERROR: Nearest neighbor type declaration unknown'
      STOP 'rats'
      ENDIF
c
      RETURN
      END
c=======================================================================
      SUBROUTINE sprinkle
c
c routine to do a random initialization
c
c Note that unlike the two-spin code, which inserted transitions
c at random to make a start configuration, this module merely
c sets the initial reference spins to random values
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 random_lmz ! function type declaration
c
      INTEGER*4 is,ndown
      REAL*4 asym
c
      ndown = 0
      DO is=1,ns
         IF (random_lmz().LT.0.5) THEN
         isref(is) = 0
         ndown = ndown + 1
         ENDIF
      ENDDO
c
c compute asymmetry=(nup-ndown)/(nup+ndown), just for kicks
      asym = FLOAT(ns - 2*ndown)/FLOAT(ns)
c
      WRITE(*,'(1h ,a,f10.5)') 'Random Start: Asymmetry = ',asym
      WRITE(2,'(1h ,a,f10.5)') 'Random Start: Asymmetry = ',asym
c
      RETURN
```

```
      END
c=======================================================================
      SUBROUTINE stagger
c
c routine to do a staggered start initialization
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 is, ix, jx
c
      IF (nntype.EQ.0) THEN
c
         DO is=2,ns,2
         isref(is) = 0
         ENDDO
c
      WRITE(*,'(1h ,a)') 'Staggered Start'
      WRITE(2,'(1h ,a)') 'Staggered Start'
c
      ELSE IF (nntype.EQ.1) THEN
c
         DO ix=1,nx
            DO jx=1,ny
            isref(ix+nx*(jx-1)) = 1 - MOD(MOD(ix,2) + MOD(jx,2),2)
            ENDDO
         ENDDO
c
      WRITE(*,'(1h ,a)') 'Staggered Start'
      WRITE(2,'(1h ,a)') 'Staggered Start'
c
      ELSE
      WRITE(*,'(/1h ,a)') 'WARNING: cannot do staggered start for '
     &//'this type of lattice'
      WRITE(2,'(/1h ,a)') 'WARNING: cannot do staggered start for '
     &//'this type of lattice'
c
      ENDIF
c
      RETURN
      END
c=======================================================================
      SUBROUTINE init_random(iseed)
c
c routine to initialize Luscher-Marsaglia-Zaman random number generator
      IMPLICIT NONE
c
      REAL*4 RAN ! function type declaration (implicit on VAX)
c
      INTEGER*4 r
      PARAMETER (r=24)
      INTEGER*4 s
      PARAMETER (s=10)
c
      REAL*4 x(0:r-1), c
      INTEGER*4 lcn, lcs, lcr
      COMMON /stuff/ x,c,lcn,lcs,lcr
c
      INTEGER*4 iseed
      INTEGER*4 ix
```

```
c
      DO ix=0,r-1
      x(ix) = RAN(iseed)  ! use VAX RAN function to fill initial array
      ENDDO
c
      c = 0
      lcr = 0
      lcs = r - s
      lcn = 0
c
      RETURN
      END
c=======================================================================
      REAL*4 FUNCTION random_lmz()
c
c routine that generates Luscher-Marsaglia-Zaman random numbers
      IMPLICIT NONE
c
      INTEGER*4 r
      PARAMETER (r=24)
      INTEGER*4 s
      PARAMETER (s=10)
      INTEGER*4 p
      PARAMETER (p=223)
      REAL*4 fb
      PARAMETER (fb=1.0)
      REAL*4 fc
      PARAMETER (fc=2.**(-24))
c
      REAL*4 x(0:r-1), c
      INTEGER*4 lcn, lcs, lcr
      COMMON /stuff/ x,c,lcn,lcs,lcr
c
      INTEGER*4 lcp
      REAL*4 d
c
      IF (lcn.EQ.0) THEN
      lcp = p - r
      lcn = r
      ELSE
      lcp = 0
      lcn = lcn - 1
      ENDIF
c
      DO WHILE(lcp.GE.0)
      d = x(lcs) - x(lcr) - c
         IF (d.LT.0.) THEN
         c = fc
         d = d + fb
         ELSE
         c = 0.
         ENDIF
      x(lcr) = d
      lcr = lcr + 1
      IF (lcr.EQ.r) lcr = 0 ! replaces cycle-eating MOD function
      lcs = lcs + 1
      IF (lcs.EQ.r) lcs = 0 ! replaces cycle-eating MOD function
      lcp = lcp - 1
```

```
      ENDDO
c
      random_lmz = d
c
      RETURN
      END
c=======================================================================
      SUBROUTINE measure
c
c records relevant observables
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 clustersize ! function type declaration
c
      INTEGER*4 iseg, ix, jx
      REAL*4 sum, stagsign
c ###
      LOGICAL*4 dolongene
      COMMON /dole/ dolongene
c
c get clustersize and average segment length
      cs = clustersize()
      sl = cs/FLOAT(nseg) ! average segment length
c
c count zero crossings to compute winding number...
c path().icross stores +1 for positive crossing
c                  and -1 for negative crossing
c                  and  0 if no crossing
      nwinding = 0
c
         DO iseg=1,nseg
         nwinding = nwinding + path(iseg).icross
         ENDDO
c
      xmag = FLOAT(nwinding)
      sus = (3.*beta/4.)*xmag*xmag/cs
c
c compute staggered susceptibility
      IF (nntype.EQ.0 .AND. MOD(ns,2).EQ.0) THEN
c
c improved estimator
      sum = 0.
c
         DO iseg=1,nseg
         stagsign = FLOAT(2*MOD(path(iseg).spin,2) - 1)
            IF (path(iseg).state.EQ.1) THEN
            sum = sum + stagsign*path(iseg).length
            ELSE
            sum = sum - stagsign*path(iseg).length
            ENDIF
         ENDDO
c
      stagsus = (3.*beta/4.)*sum*sum/cs
c
      ELSE IF (nntype.EQ.1) THEN
c
c improved estimator
      sum = 0.
```

```
c
         DO iseg=1,nseg
         ix = 1 + MOD(path(iseg).spin - 1,nx)
         jx = 1 + (path(iseg).spin - ix)/nx
         stagsign = FLOAT(1 - 2*MOD(MOD(ix,2) + MOD(jx,2),2))
            IF (path(iseg).state.EQ.1) THEN
            sum = sum + stagsign*path(iseg).length
            ELSE
            sum = sum - stagsign*path(iseg).length
            ENDIF
         ENDDO
c
      stagsus = (3.*beta/4.)*sum*sum/cs
c
      ELSE ! note we don't do stag sus for hex lattice
      stagsus = 0.
c
      ENDIF
c
c improved estimator for internal energy -- see note in genpath
c ### split up ene
      ene_1 = ene/cs
      ene_2 = -FLOAT(nseg-1)/(4.*beta*cs)
      ene_3 = FLOAT(ncannew)/(2.*beta*cs)
      ene = ene_1 + ene_2 + ene_3
c      ene = (ene - FLOAT(nseg-1)/(4.*beta) + FLOAT(ncannew)/(2.*beta))
c     &      /cs
      IF (dolongene) CALL longene ! unimproved est (overwrites ene)
c
c only do accumulations after "thermalization" period
      IF (iloop.GT.nexclude) THEN
c
c accumulate statistics on the number of transitions
      IF (do_table) m_accum(mused/2) = m_accum(mused/2) + 1
c
      xmagsum = xmagsum + xmag ! should average to zero
      sussum  = sussum + sus
      stagsussum = stagsussum + stagsus
      enesum = enesum + ene
      cssum = cssum + cs
      slsum = slsum + sl
c
      xmag2sm = xmag2sm + xmag*xmag
      sus2sm  = sus2sm + sus*sus
      stagsus2sm = stagsus2sm + stagsus*stagsus
      ene2sm = ene2sm + ene*ene
      cs2sm = cs2sm + cs*cs
      sl2sm = sl2sm + sl*sl
c### split of ene
      ene_1sum = ene_1sum + ene_1
      ene_2sum = ene_2sum + ene_2
      ene_3sum = ene_3sum + ene_3
      ene_12sm = ene_12sm + ene_1*ene_1
      ene_22sm = ene_22sm + ene_2*ene_2
      ene_32sm = ene_32sm + ene_3*ene_3
c
c store data for correlation lengths
      IF (xj.GT.0.) CALL corraccum1d
```

```
c
      ENDIF
c
c this variable is for tracking purposes - (reassign at will....)
      trackaccum = enesum
c
c accumulate statistics for data storage requirements
      IF (mused.GT.mused_act) mused_act = mused
      IF (nseg.GT.nseg_act)   nseg_act  = nseg
c
      RETURN
      END
c=====================================================================
      SUBROUTINE longene
c
c routine to measure energy the long way (unimproved estimator)
c
c only need to do staggered sites (o's in diagram)
c
c    |  |  |  |  |  |  |  |
c  --o--x--o--x--o--x--o--x--
c    |  |  |  |  |  |  |  |
c  --x--o--x--o--x--o--x--o--
c    |  |  |  |  |  |  |  |
c  --o--x--o--x--o--x--o--x--
c    |  |  |  |  |  |  |  |
c  --x--o--x--o--x--o--x--o--
c    |  |  |  |  |  |  |  |
c  --o--x--o--x--o--x--o--x--
c    |  |  |  |  |  |  |  |
c  --x--o--x--o--x--o--x--o--
c    |  |  |  |  |  |  |  |
c  --o--x--o--x--o--x--o--x--
c    |  |  |  |  |  |  |  |
c  --x--o--x--o--x--o--x--o--
c    |  |  |  |  |  |  |  |
c
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 deltat ! function type declaration
c
      INTEGER*4 ix,jx,in,nix,njx,kx
      INTEGER*4 ishome,isnbr,istatehome,istatenbr,mhome,mnbr
      REAL*4 w_len(0:1,0:1),w_bond
      REAL*4 t0,t1,txhome,txnbr
c ### split up ene
      REAL*4 enexxx(0:1,0:1)
c
      IF (nntype.EQ.0) THEN
      nix = 1
      njx = ns
      ELSE IF (nntype.EQ.1) THEN
      nix = nx
      njx = ny
      ELSE
      RETURN
      ENDIF
c
```

```
      istatecurr = 1 ! needed for function deltat
c
      w_bond = -1./(8.*beta)
      w_len(0,0) = xj/8.
      w_len(0,1) = -w_len(0,0)
      w_len(1,0) = w_len(0,1)
      w_len(1,1) = w_len(0,0)
c
      ene = 0.
      enexxx(0,0) = 0.0
      enexxx(0,1) = 0.0
      enexxx(1,0) = 0.0
      enexxx(1,1) = 0.0
c
      DO ix=1,nix
c
         DO jx=MOD(ix-1,2)+1,njx,2 ! only do staggered sites
         ishome = jx + (ix - 1)*njx
c
            DO in=1,nn
            isnbr = neighbor(ishome,in)
c
            istatehome = isref(ishome)
            istatenbr  = isref(isnbr)
c
               IF (mtrans(ishome).GT.0 .AND. mtrans(isnbr).GT.0) THEN
               mhome = mfirst(ishome)
               mnbr  = mfirst(isnbr)
               t0 = 0.
               t1 = AMIN1(trans(mhome),trans(mnbr))
c
                  DO WHILE (t1.LT.1.0)
                  ene = ene + w_len(istatehome,istatenbr)*deltat(t0,t1)
                  enexxx(istatehome,istatenbr) = !### split up ene
     &            enexxx(istatehome,istatenbr) +
     &            w_len(istatehome,istatenbr)*deltat(t0,t1)
                  t0 = t1
c
                     IF (trans(mhome).EQ.t1) THEN
                     mhome = iptr(mhome,1)
                     istatehome = 1 - istatehome
                     ENDIF
                     IF (trans(mnbr ).EQ.t1) THEN
                     mnbr  = iptr(mnbr ,1)
                     istatenbr  = 1 - istatenbr
                     ENDIF
c
                  txhome = trans(mhome)
                  IF (t1.GT.trans(mhome)) txhome = 1.
                  txnbr  = trans(mnbr)
                  IF (t1.GT.trans(mnbr )) txnbr  = 1.
                  t1 = AMIN1(txhome,txnbr)
c
                  ENDDO
c
c last segment (t1=1.)
               ene = ene + w_len(istatehome,istatenbr)*deltat(t0,t1)
                  enexxx(istatehome,istatenbr) = !### split up ene
```

```
     &            enexxx(istatehome,istatenbr) +
     &            w_len(istatehome,istatenbr)*deltat(t0,t1)
c
               ELSE IF (mtrans(ishome).GT.0) THEN ! nbr has no trans
               mhome = mfirst(ishome)
               t0 = trans(iptr(mhome,0)) ! start deltat before t=0
               t1 = trans(mhome)
c
                  DO kx=1,mtrans(ishome)
                  ene = ene + w_len(istatehome,istatenbr)*deltat(t0,t1)
                  enexxx(istatehome,istatenbr) = !### split up ene
     &            enexxx(istatehome,istatenbr) +
     &            w_len(istatehome,istatenbr)*deltat(t0,t1)
                  mhome = iptr(mhome,1)
                  t0 = t1
                  t1 = trans(mhome)
                  istatehome = 1 - istatehome
                  ENDDO
c
               ELSE IF (mtrans(isnbr ).GT.0) THEN ! home has no trans
               mnbr = mfirst(isnbr)
               t0 = trans(iptr(mnbr,0)) ! start deltat before t=0
               t1 = trans(mnbr)
c
                  DO kx=1,mtrans(isnbr)
                  ene = ene + w_len(istatenbr,istatehome)*deltat(t0,t1)
                  enexxx(istatehome,istatenbr) = !### split up ene
     &            enexxx(istatehome,istatenbr) +
     &            w_len(istatenbr,istatehome)*deltat(t0,t1)
                  mnbr = iptr(mnbr,1)
                  t0 = t1
                  t1 = trans(mnbr)
                  istatenbr = 1 - istatenbr
                  ENDDO
c
               ELSE ! both are constant so deltat = 1.0
               ene = ene + w_len(istatehome,istatenbr)
                  enexxx(istatehome,istatenbr) = !### split up ene
     &            enexxx(istatehome,istatenbr) +
     &            w_len(istatehome,istatenbr)
c
               ENDIF
c
            ENDDO
c
         ENDDO
c
      ENDDO
c
c add in contribution of transitions (number of transitions = mused/2)
      ene = ene + w_bond*FLOAT(mused/2)
c ### split up ene
      ene_1 = enexxx(0,0) + enexxx(1,1) ! "like neighbors"
      ene_2 = enexxx(0,1) + enexxx(1,0) ! "unlike neighbors"
      ene_3 = w_bond*FLOAT(mused/2)     ! "transitions"
c
      RETURN
      END
```

```
c=======================================================================
      REAL*4 FUNCTION clustersize()
c
c computes path length
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 iseg
c
      clustersize = 0.
c
      DO iseg=1,nseg
      clustersize = clustersize + path(iseg).length
      ENDDO
c
      RETURN
      END
c=======================================================================
      SUBROUTINE corraccum1d
c
c stores correlation length data in accumulation array
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 ix,ixd,j1,j2,iseg
      INTEGER*4 nnon,nonnullx(nx)
      REAL*4 val(0:nx-1)
c
c NOTE THAT stagger*spin state is already accounted for in the
c stored segment lengths, which are all positive.
c
c get sum of staggered spin for each site
c very similar to staggered susceptibility calculation
c
c initialize array (this should really be in loop_init,
c but let's encapsulate)
      DO ix=1,nx
      cx1dsum(ix) = 0.
      ENDDO
c
      IF (nntype.EQ.0 .AND. MOD(ns,2).EQ.0) THEN
c
         DO iseg=1,nseg
         ix = path(iseg).spin
         cx1dsum(ix) = cx1dsum(ix) + path(iseg).length
         ENDDO
c
      ELSE IF (nntype.EQ.1) THEN
c
         DO iseg=1,nseg
         ix = 1 + MOD(path(iseg).spin - 1,nx)
         cx1dsum(ix) = cx1dsum(ix) + path(iseg).length
         ENDDO
c
      ENDIF
c
c find the non-zero elements
      nnon = 0
      DO ix=1,nx
         IF (cx1dsum(ix).NE.0) THEN
```

```
         nnon = nnon + 1
         nonnullx(nnon) = ix
         ENDIF
      val(ix-1) = 0. ! initialize local accumulator
      ENDDO
c
c compute contribution to accumulator
c this code doubles counts each pair but required no extra
c unbiasing or symmetrization.
      DO j1=1,nnon
         DO j2=1,nnon  ! was DO j2=j1,nnon...dbl cntng symmetrizes this
         ixd = nonnullx(j2) - nonnullx(j1)
         IF (ixd.LT.0) ixd = ixd + nx ! "forward difference"
c
c The 1/cs is required because we compensate for cluster size in the
c single-cluster algorithm.
         val(ixd) = val(ixd) + cx1dsum(nonnullx(j1))*
     &                         cx1dsum(nonnullx(j2))/cs
         ENDDO
      ENDDO
c
      DO ixd=0,nx-1
      c1dsum(ixd) = c1dsum(ixd) + val(ixd)
      c1dsm2(ixd) = c1dsm2(ixd) + val(ixd)*val(ixd)
      ENDDO
c
      RETURN
      END
c=======================================================================
      SUBROUTINE docorr1d
c
c finish up correlation length data
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 ixd
      REAL*4 stagck,xn
c
c normalize
      WRITE(*,'(/1h ,a)') 'Proceeding to compute 1d correlation array'
c
c normalize and compute errors
      WRITE(2,'(//1h ,a)') '1D Correlations'
      WRITE(2,'(/1h ,t3,a,2(6x,a,2x))') 'dx',' x(dx) ',' err '
      stagck = 0.
c
      DO ixd=0,nx-1
      xn = FLOAT(nloop-nexclude-num_sf)
      c1dsum(ixd) = c1dsum(ixd)/xn
      c1dsm2(ixd) = c1dsm2(ixd)/xn
      c1dsm2(ixd) = SQRT(c1dsm2(ixd)-c1dsum(ixd)*c1dsum(ixd))
     &                  /SQRT(xn-1.)
c
      WRITE(2,'(1h ,i4,2x,f12.8,2x,a,f12.8)')
     &ixd,c1dsum(ixd),'+/-',c1dsm2(ixd)
c
      stagck = stagck + c1dsum(ixd)
      ENDDO
c
```

```
      stagck = (3.*beta/4.)*stagck
      WRITE(2,'(//1h ,a,f12.7)')
     &'Cross check this sum with Staggered Susceptibility:',stagck
      WRITE(2,'(1h ,a,f12.7)')
     &'              from above: Staggered Susceptibility:',stagsusbar
      WRITE(2,'(1h ,a,f12.8)') 'Ratio = ',stagck/stagsusbar
c
      CALL do_xi
c
      RETURN
      END
c=======================================================================
      SUBROUTINE do_xi
c
c adapted from the program 'get_xi.for'
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 nxmax
      PARAMETER (nxmax=640) ! set to largest number of input points
      INTEGER*4 mmax
      PARAMETER (MMAX=20) ! set to largest number of fit parameters
c
      REAL*4 GAMMQ ! function type declaration
c
c shell for testing routine to do least-squares fit on non-linear
c function ~ in this case, A*cosh((x-x_0)/B)
c Cf. Press et al., Numerical Recipes, section 14.4
c
      INTEGER*4 ix,jx,ndata
      INTEGER*4 ma,nca,mfit
      REAL*4 alamda
      REAL*4 x(nxmax),y(nxmax),sig(nxmax)
      REAL*4 a(2) ! ma = 2
      INTEGER*4 lista(2) ! ma = 2
      REAL*4 covar(2,2),alpha(2,2) ! nca = 2
      REAL*4 chisq,chisqold,chisqqnu
      INTEGER*4 ncalls
      LOGICAL done
c
      WRITE(*,'(/1h ,a,f6.2,a,i5,a,i5)')
     &'CL Data reduction for beta = ',beta,', nx = ',nx,', and ny = ',ny
      WRITE(2,'(/1h ,a,f6.2,a,i5,a,i5)')
     &'CL Data reduction for beta = ',beta,', nx = ',nx,', and ny = ',ny
c
      ma = 2 ! total number of parameters
      mfit = 2 ! number of parameters to vary in the fit
      nca = 2 ! dimension of covariance matrix
      lista(1) = 1
      lista(2) = 2
c
c note we shift the definition of the index and number from 1,2...nx
      ndata = nx/2 + 1
      DO ix=1,ndata
      x(ix) = FLOAT(ndata - ix) ! note we do the zero shift here
      y(ix) = c1dsum(ix-1)
      sig(ix) = c1dsm2(ix-1)
      ENDDO
c
```

```
c initialize routine
      alamda = -1.0
c
         IF (y(ndata).NE.0.) THEN
         a(1) = y(ndata)
         a(2) = FLOAT(ndata-1)/ALOG(2.*y(1)/y(ndata))
         ELSE ! use 2 and 3 because of suspicion of bias in y(1)
         a(2) = 1./ALOG(y(2)/y(3))
         a(1) = 2.*y(2)/EXP(FLOAT(ndata-2)/a(2))
            DO WHILE (y(ndata).EQ.0.)
            ndata = ndata - 1
               IF (ndata.EQ.0) THEN
               WRITE(*,'(1h ,a)')
     &'WARNING! encountered too many y=0. points! Skipping this point!'
               RETURN
               ENDIF
            ENDDO
         ENDIF
c
      WRITE(*,'(/1h ,a,i4,a/)') 'Using first ',ndata,' points'
c
      CALL mrqmin(x,y,sig,ndata,a,ma,lista,mfit,
     &    covar,alpha,nca,chisq,alamda)
      ncalls = 1
      WRITE(*,'(1h ,i3,a,f7.2,1x,3(a,g14.6,1x))')
     &ncalls,'log(alamda) = ',ALOG10(alamda),
     &'A = ',a(1),
     &'xi = ',a(2),
     &'ChiSq = ',chisq
c
c iterate
      chisqold = 1000000.
      done = .FALSE.
      DO WHILE (.NOT.done)
      CALL mrqmin(x,y,sig,ndata,a,ma,lista,mfit,
     &    covar,alpha,nca,chisq,alamda)
      ncalls = ncalls + 1
      WRITE(*,'(1h ,i3,a,f7.2,1x,3(a,g14.6,1x))')
     &ncalls,'log(alamda) = ',ALOG10(alamda),
     &'A = ',a(1),
     &'xi = ',a(2),
     &'ChiSq = ',chisq
         IF (chisq.LT.chisqold .AND. (chisqold-chisq).LT.0.1) THEN
         done = .TRUE.
         ELSE IF (alamda.GT.1.) THEN
         done = .TRUE.
         WRITE(2,'(1h ,a/)')
     &   'Warning: Termination by ALAMDA > 1 Criterion. '//
     &   'Examine data carefully.'
         ENDIF
      chisqold = chisq
      ENDDO
c
c find covariance matrix
      alamda = 0.0
      CALL mrqmin(x,y,sig,ndata,a,ma,lista,mfit,
     &    covar,alpha,nca,chisq,alamda)
      ncalls = ncalls + 1
```

```
      WRITE(*,'(1h ,i3,22x,3(a,g14.6,1x)//)')
     &ncalls,
     &'A = ',a(1),
     &'xi = ',a(2),
     &'ChiSq = ',chisq
c
c assign output
      aco = a(1)
      acoerr = SQRT(covar(1,1))
      xi = a(2)
      xierr = SQRT(covar(2,2))
      xichisq = chisq
      xinu = FLOAT(ndata-2)
      xiqval = GAMMQ(xinu/2.,xichisq/2.)
      chisqqnu = xichisq/xinu
c
      WRITE(*,'(/1h ,a)') 'Bottom line:'
      WRITE(*,'(1h ,a,g14.6,2x,a,2x,g14.6)')
     &'A  = ',aco,'+/-',acoerr
      WRITE(*,'(1h ,a,g14.6,2x,a,2x,g14.6/)')
     &'Xi = ',xi,'+/-',xierr
c
c repeat for file:
      WRITE(2,'(/1h ,a)') 'Covariance matrix:'
      WRITE(2,'(1h ,2g18.7)') ((covar(ix,jx),ix=1,2),jx=1,2)
c
      WRITE(2,'(/1h ,a)') 'Curvature matrix:'
      WRITE(2,'(1h ,2g18.7)') ((alpha(ix,jx),ix=1,2),jx=1,2)
c
      WRITE(2,'(/1h ,a)') 'Bottom line:'
      WRITE(2,'(1h ,a,g14.6,2x,a,2x,g14.6)')
     &'A  = ',aco,'+/-',acoerr
      WRITE(2,'(1h ,a,g14.6,2x,a,2x,g14.6/)')
     &'Xi = ',xi,'+/-',xierr
c
      WRITE(2,'(/1h ,a,f12.7)')
     &'Chi squared = ',xichisq
      WRITE(2,'(1h ,a,f12.7,a,f12.7,a/)')
     &'ChiSq/Nu    = ',chisqqnu,' (significance = ',xiqval,')'
c
      WRITE(2,'(/1h ,a,i3/)') 'Total number of calls to MRQMIN:',ncalls
c
      RETURN
      END
c=======================================================================
      SUBROUTINE funcs(x,args,y,dyda,ma)
c
c fitting function: y = A*cosh(x/B)
c                dy/dA = cosh(x/B)
c                dy/dB = A*sinh(x/B)*(-x/B**2)
      REAL*4 args(ma), dyda(ma)
      REAL*4 a,b
c
      a = args(1)
      b = args(2)
      y = a*COSH(x/b)
      dyda(1) = COSH(x/b)
      dyda(2) = a*sinh(x/b)*(-x/b**2)
```

```
c
      RETURN
      END
c=======================================================================
      SUBROUTINE MRQMIN(X,Y,SIG,NDATA,A,MA,LISTA,MFIT,
     *    COVAR,ALPHA,NCA,CHISQ,ALAMDA)
c
c Levenberg-Marquardt method, attempting to reduce the value of
c chi-squared of a fit between a set of NDATA data points X(I), Y(I)
c with individual standard deviations SIG(I), and a nonlinear function
c dependent on MA coefficients A. The array LISTA numbers the parameters
c A such that the first MFIT elements correspond to values actually
c being adjusted; the remaining MA-MFIT are held fixed at their input
c values. The program returns current best-fit values for the MA fit
c parameters A, and chi-squared = CHISQ. The arrays COVAR(NCA,NCA),
c ALPHA(NCA,NCA) with physical dimension NCA (.GE. MFIT) are used
c as working space during most iterations. Supply a subroutine
c FUNCS(X,A,YFIT,DYDA,MA) that evaluates the fitting function YFIT,
c and its derivatives with respect to the fitting parameters A at X.
c On the first call provide an initial guess for the parameters A,
c and set ALAMDA < 0 for initialization (which then sets
c ALAMDA = 0.001). If a step succeeds CHISQ becomes smaller and ALAMDA
c decreases by a factor of 10. If a step fails ALAMDA grows by a
c factor of 10. You must call this routine repeatedly until
c convergence is achieved. Then, make one final call with ALAMDA = 0,
c so that COVAR(I,J) returns the covariance matrix, and ALPHA(I,J)
c the curvatures matrix.
      PARAMETER (MMAX=20) ! set to largest number of fit parameters
      DIMENSION X(NDATA),Y(NDATA),SIG(NDATA),A(MA),LISTA(MA),
     *  COVAR(NCA,NCA),ALPHA(NCA,NCA),ATRY(MMAX),BETA(MMAX),DA(MMAX)
      IF(ALAMDA.LT.0.)THEN ! initialization
        KK=MFIT+1
        DO 12 J=1,MA ! does LISTA contain a proper permutation of
                     ! the coefficients?
          IHIT=0
          DO 11 K=1,MFIT
            IF(LISTA(K).EQ.J)IHIT=IHIT+1
11        CONTINUE
          IF (IHIT.EQ.0) THEN
            LISTA(KK)=J
            KK=KK+1
          ELSE IF (IHIT.GT.1) THEN
            PAUSE 'Improper permutation in LISTA'
          ENDIF
12      CONTINUE
        IF (KK.NE.(MA+1)) PAUSE 'Improper permutation in LISTA'
        ALAMDA=0.001
        CALL MRQCOF(X,Y,SIG,NDATA,A,MA,LISTA,MFIT,ALPHA,BETA,NCA,CHISQ)
        OCHISQ=CHISQ
        DO 13 J=1,MA
          ATRY(J)=A(J)
13      CONTINUE
      ENDIF
      DO 15 J=1,MFIT ! Alter linearized fitting matrix, by augmenting
                     ! diagonal elements
        DO 14 K=1,MFIT
          COVAR(J,K)=ALPHA(J,K)
14      CONTINUE
```

```
        COVAR(J,J)=ALPHA(J,J)*(1.+ALAMDA)
        DA(J)=BETA(J)
15    CONTINUE
      CALL GAUSSJ(COVAR,MFIT,NCA,DA,1,1) ! Matrix solution
      IF(ALAMDA.EQ.0.)THEN ! once converged, evaluate covariance matrix
        CALL COVSRT(COVAR,NCA,MA,LISTA,MFIT)
        RETURN
      ENDIF
      DO 16 J=1,MFIT ! Did the trial succeed?
        ATRY(LISTA(J))=A(LISTA(J))+DA(J)
16    CONTINUE
      CALL MRQCOF(X,Y,SIG,NDATA,ATRY,MA,LISTA,MFIT,COVAR,DA,NCA,CHISQ)
      IF(CHISQ.LT.OCHISQ)THEN ! success, accept the new solution
        ALAMDA=0.1*ALAMDA
        OCHISQ=CHISQ
        DO 18 J=1,MFIT
          DO 17 K=1,MFIT
            ALPHA(J,K)=COVAR(J,K)
17        CONTINUE
          BETA(J)=DA(J)
          A(LISTA(J))=ATRY(LISTA(J))
18      CONTINUE
      ELSE ! failure, increase ALAMDA and return
        ALAMDA=10.*ALAMDA
        CHISQ=OCHISQ
      ENDIF
      RETURN
      END
c=======================================================================
      SUBROUTINE MRQCOF(X,Y,SIG,NDATA,A,MA,LISTA,MFIT,ALPHA,BETA,NALP,
     *CHISQ)
c
c Used by MRQMIN to evaluate the linearized fitting matrix ALPHA,
c and the vector BETA as in (14.4.8).
      PARAMETER (MMAX=20)
      DIMENSION X(NDATA),Y(NDATA),SIG(NDATA),ALPHA(NALP,NALP),BETA(MA),
     *    DYDA(MMAX),LISTA(MFIT),A(MA)
      DO 12 J=1,MFIT ! Initialize (symmetric) ALPHA, BETA
        DO 11 K=1,J
          ALPHA(J,K)=0.
11      CONTINUE
        BETA(J)=0.
12    CONTINUE
      CHISQ=0.
      DO 15 I=1,NDATA ! summation loop over all data
        CALL FUNCS(X(I),A,YMOD,DYDA,MA)
        SIG2I=1./(SIG(I)*SIG(I))
        DY=Y(I)-YMOD
        DO 14 J=1,MFIT
          WT=DYDA(LISTA(J))*SIG2I
          DO 13 K=1,J
            ALPHA(J,K)=ALPHA(J,K)+WT*DYDA(LISTA(K))
13        CONTINUE
          BETA(J)=BETA(J)+DY*WT
14      CONTINUE
        CHISQ=CHISQ+DY*DY*SIG2I ! and find Chi Squared
15    CONTINUE
      DO 17 J=2,MFIT ! Fill in the symmetric side
```

```
        DO 16 K=1,J-1
          ALPHA(K,J)=ALPHA(J,K)
16      CONTINUE
17    CONTINUE
      RETURN
      END
c=======================================================================
      SUBROUTINE COVSRT(COVAR,NCVM,MA,LISTA,MFIT)
c
c Given the covariance matrix COVAR of a fit for MFIT of MA total
c parameters, and their ordering LISTA(I), repack the covariance
c matrix to the true order of the parameters. Elements associated
c with fixed parameters will be zero. NCVM is the physical dimension
c of COVAR.
      DIMENSION COVAR(NCVM,NCVM),LISTA(MFIT)
      DO 12 J=1,MA-1 ! Zero all elements below diagonal
        DO 11 I=J+1,MA
          COVAR(I,J)=0.
11      CONTINUE
12    CONTINUE
      DO 14 I=1,MFIT-1 ! Repack off-diagonal elements of fit into
                       ! correct location below diagonal.
        DO 13 J=I+1,MFIT
          IF(LISTA(J).GT.LISTA(I)) THEN
            COVAR(LISTA(J),LISTA(I))=COVAR(I,J)
          ELSE
            COVAR(LISTA(I),LISTA(J))=COVAR(I,J)
          ENDIF
13      CONTINUE
14    CONTINUE
      SWAP=COVAR(1,1) ! Temporarily store original diagonal elements
                      ! in top row, and zero the diagonal.
      DO 15 J=1,MA
        COVAR(1,J)=COVAR(J,J)
        COVAR(J,J)=0.
15    CONTINUE
      COVAR(LISTA(1),LISTA(1))=SWAP
      DO 16 J=2,MFIT ! Now sort elements into proper order on diagonal.
        COVAR(LISTA(J),LISTA(J))=COVAR(1,J)
16    CONTINUE
      DO 18 J=2,MA ! Finally, fill in above diagonal by symmetry.
        DO 17 I=1,J-1
          COVAR(I,J)=COVAR(J,I)
17      CONTINUE
18    CONTINUE
      RETURN
      END
c=======================================================================
      SUBROUTINE GAUSSJ(A,N,NP,B,M,MP)
c
c Linear equation solution by Gauss-Jordan elimination, equation (2.1.1)
c in Press et al. See Section 2.1 for further details.
      PARAMETER (NMAX=50)
      DIMENSION A(NP,NP),B(NP,MP),IPIV(NMAX),INDXR(NMAX),INDXC(NMAX)
      DO 11 J=1,N
        IPIV(J)=0
11    CONTINUE
      DO 22 I=1,N
```

```
      BIG=0.
      DO 13 J=1,N
        IF(IPIV(J).NE.1)THEN
          DO 12 K=1,N
            IF (IPIV(K).EQ.0) THEN
              IF (ABS(A(J,K)).GE.BIG)THEN
                BIG=ABS(A(J,K))
                IROW=J
                ICOL=K
              ENDIF
            ELSE IF (IPIV(K).GT.1) THEN
              PAUSE 'Singular matrix'
            ENDIF
12        CONTINUE
        ENDIF
13    CONTINUE
      IPIV(ICOL)=IPIV(ICOL)+1
      IF (IROW.NE.ICOL) THEN
        DO 14 L=1,N
          DUM=A(IROW,L)
          A(IROW,L)=A(ICOL,L)
          A(ICOL,L)=DUM
14      CONTINUE
        DO 15 L=1,M
          DUM=B(IROW,L)
          B(IROW,L)=B(ICOL,L)
          B(ICOL,L)=DUM
15      CONTINUE
      ENDIF
      INDXR(I)=IROW
      INDXC(I)=ICOL
      IF (A(ICOL,ICOL).EQ.0.) PAUSE 'Singular matrix.'
      PIVINV=1./A(ICOL,ICOL)
      A(ICOL,ICOL)=1.
      DO 16 L=1,N
        A(ICOL,L)=A(ICOL,L)*PIVINV
16    CONTINUE
      DO 17 L=1,M
        B(ICOL,L)=B(ICOL,L)*PIVINV
17    CONTINUE
      DO 21 LL=1,N
        IF(LL.NE.ICOL)THEN
          DUM=A(LL,ICOL)
          A(LL,ICOL)=0.
          DO 18 L=1,N
            A(LL,L)=A(LL,L)-A(ICOL,L)*DUM
18        CONTINUE
          DO 19 L=1,M
            B(LL,L)=B(LL,L)-B(ICOL,L)*DUM
19        CONTINUE
        ENDIF
21    CONTINUE
22  CONTINUE
    DO 24 L=N,1,-1
      IF(INDXR(L).NE.INDXC(L))THEN
        DO 23 K=1,N
          DUM=A(K,INDXR(L))
          A(K,INDXR(L))=A(K,INDXC(L))
```

```
            A(K,INDXC(L))=DUM
23        CONTINUE
        ENDIF
24    CONTINUE
      RETURN
      END
c=======================================================================
      FUNCTION GAMMQ(A,X)
      IF(X.LT.0..OR.A.LE.0.)PAUSE
      IF(X.LT.A+1.)THEN
        CALL GSER(GAMSER,A,X,GLN)
        GAMMQ=1.-GAMSER
      ELSE
        CALL GCF(GAMMCF,A,X,GLN)
        GAMMQ=GAMMCF
      ENDIF
      RETURN
      END
c=======================================================================
      SUBROUTINE GCF(GAMMCF,A,X,GLN)
      PARAMETER (ITMAX=100,EPS=3.E-7)
      GLN=GAMMLN(A)
      GOLD=0.
      A0=1.
      A1=X
      B0=0.
      B1=1.
      FAC=1.
      DO 11 N=1,ITMAX
        AN=FLOAT(N)
        ANA=AN-A
        A0=(A1+A0*ANA)*FAC
        B0=(B1+B0*ANA)*FAC
        ANF=AN*FAC
        A1=X*A0+ANF*A1
        B1=X*B0+ANF*B1
        IF(A1.NE.0.)THEN
          FAC=1./A1
          G=B1*FAC
          IF(ABS((G-GOLD)/G).LT.EPS)GO TO 1
          GOLD=G
        ENDIF
11    CONTINUE
      PAUSE 'A too large, ITMAX too small'
1     GAMMCF=EXP(-X+A*ALOG(X)-GLN)*G
      RETURN
      END
c=======================================================================
      FUNCTION GAMMLN(XX)
      REAL*8 COF(6),STP,HALF,ONE,FPF,X,TMP,SER
      DATA COF,STP/76.18009173D0,-86.50532033D0,24.01409822D0,
     *    -1.231739516D0,.120858003D-2,-.536382D-5,2.50662827465D0/
      DATA HALF,ONE,FPF/0.5D0,1.0D0,5.5D0/
      X=XX-ONE
      TMP=X+FPF
      TMP=(X+HALF)*LOG(TMP)-TMP
      SER=ONE
      DO 11 J=1,6
```

```
        X=X+ONE
        SER=SER+COF(J)/X
11    CONTINUE
      GAMMLN=TMP+LOG(STP*SER)
      RETURN
      END
c=======================================================================
      SUBROUTINE GSER(GAMSER,A,X,GLN)
      PARAMETER (ITMAX=100,EPS=3.E-7)
      GLN=GAMMLN(A)
      IF(X.LE.0.)THEN
        IF(X.LT.0.)PAUSE
        GAMSER=0.
        RETURN
      ENDIF
      AP=A
      SUM=1./A
      DEL=SUM
      DO 11 N=1,ITMAX
        AP=AP+1.
        DEL=DEL*X/AP
        SUM=SUM+DEL
        IF(ABS(DEL).LT.ABS(SUM)*EPS)GO TO 1
11    CONTINUE
      PAUSE 'A too large, ITMAX too small'
1     GAMSER=SUM*EXP(-X+A*LOG(X)-GLN)
      RETURN
      END
c=======================================================================
      SUBROUTINE writer
c
c writes final output
      INCLUDE 'lattmpas.hdr'
c
      REAL*4 qm_exact   ! function type declaration
c
c array for storing error bounds
      REAL*4 dm_accum(0:nm)
      REAL*4 qm_0,qm_store(10)
c
c local variables
      INTEGER*4 ix, iout1, iout2
      CHARACTER*80 line
      REAL*4 x
      REAL*4 xn
      REAL*4 gs ! fudge factor for masking effect
      DATA gs / 0.666667 /
c
      xn = FLOAT(nloop-nexclude-num_sf)
c
      xmagbar = xmagsum/xn
      xmagdev = SQRT((xmag2sm - xn*xmagbar**2)/(xn - 1.))
      xmagerr = xmagdev/SQRT(xn)
c
      IF (xmagbar.NE.0.) THEN
      xmagpct = 100.*ABS(xmagerr/xmagbar)
      ELSE
      xmagpct = 0.
```

```
      ENDIF
c
      susbar = sussum/xn
      susdev = SQRT((sus2sm - xn*susbar**2)/(xn - 1.))
      suserr = susdev/SQRT(xn)
c
      IF (susbar.NE.0.) THEN
      suspct = 100.*ABS(suserr/susbar)
      ELSE
      suspct = 0.
      ENDIF
c
      stagsusbar = stagsussum/xn
      stagsusdev = SQRT((stagsus2sm - xn*stagsusbar**2)/(xn - 1.))
      stagsuserr = stagsusdev/SQRT(xn)
c
      IF (stagsusbar.NE.0.) THEN
      stagsuspct = 100.*ABS(stagsuserr/stagsusbar)
      ELSE
      stagsuspct = 0.
      ENDIF
c
      enebar = enesum/xn
      enedev = SQRT((ene2sm - xn*enebar**2)/(xn - 1.))
      eneerr = enedev/SQRT(xn)
c
      IF (enebar.NE.0.) THEN
      enepct = 100.*ABS(eneerr/enebar)
      ELSE
      enepct = 0.
      ENDIF
c ### "split up ene"
      ene_1bar = ene_1sum/xn
      ene_1dev = SQRT((ene_12sm - xn*ene_1bar**2)/(xn - 1.))
      ene_1err = ene_1dev/SQRT(xn)
c
      IF (ene_1bar.NE.0.) THEN
      ene_1pct = 100.*ABS(ene_1err/ene_1bar)
      ELSE
      ene_1pct = 0.
      ENDIF
c
      ene_2bar = ene_2sum/xn
      ene_2dev = SQRT((ene_22sm - xn*ene_2bar**2)/(xn - 1.))
      ene_2err = ene_2dev/SQRT(xn)
c
      IF (ene_2bar.NE.0.) THEN
      ene_2pct = 100.*ABS(ene_2err/ene_2bar)
      ELSE
      ene_2pct = 0.
      ENDIF
c
      ene_3bar = ene_3sum/xn
      ene_3dev = SQRT((ene_32sm - xn*ene_3bar**2)/(xn - 1.))
      ene_3err = ene_3dev/SQRT(xn)
c
      IF (ene_3bar.NE.0.) THEN
      ene_3pct = 100.*ABS(ene_3err/ene_3bar)
```

```
      ELSE
      ene_3pct = 0.
      ENDIF
c ### end "split up ene" section
      csbar = cssum/xn
      csdev = SQRT((cs2sm - xn*csbar**2)/(xn - 1.))
      cserr = csdev/SQRT(xn)
c
      IF (csbar.NE.0.) THEN
      cspct = 100.*ABS(cserr/csbar)
      ELSE
      cspct = 0.
      ENDIF
c
      slbar = slsum/xn
      sldev = SQRT((sl2sm - xn*slbar**2)/(xn - 1.))
      slerr = sldev/SQRT(xn)
c
      IF (slbar.NE.0.) THEN
      slpct = 100.*ABS(slerr/slbar)
      ELSE
      slpct = 0.
      ENDIF
c
c summarize running variables and observables
      WRITE(2,'(/1h ,a,f7.3,5x,a,f7.3)') 'Beta = ',beta,'J = ',xj
      WRITE(2,'(/1h ,a/)') 'Averages +/- errors:'
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,15x,a,f10.5)')
     & 'Magnetization   :',xmagbar   ,'+/-',xmagerr   ,
     &            'sdev=',xmagdev
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,2x,a,f5.2,a,5x,a,f10.5)')
     & 'Susceptibility  :',susbar    ,'+/-',suserr    ,'(',suspct,'%)',
     &            'sdev=',susdev
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,2x,a,f5.2,a,5x,a,f10.5)')
     & 'Stag Susc.      :',stagsusbar,'+/-'
     &                           ,stagsuserr,'(',stagsuspct,'%)',
     &            'sdev=',stagsusdev
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,2x,a,f5.2,a,5x,a,f10.5)')
     & 'Internal Energy :',enebar    ,'+/-',eneerr    ,'(',enepct,'%)',
     &            'sdev=',enedev
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,2x,a,f5.2,a,5x,a,f10.5)')
     & 'Energy (1)      :',ene_1bar  ,'+/-',ene_1err,'(',ene_1pct,'%)',
     &            'sdev=',ene_1dev
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,2x,a,f5.2,a,5x,a,f10.5)')
     & 'Energy (2)      :',ene_2bar  ,'+/-',ene_2err,'(',ene_2pct,'%)',
     &            'sdev=',ene_2dev
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,2x,a,f5.2,a,5x,a,f10.5)')
     & 'Energy (3)      :',ene_3bar  ,'+/-',ene_3err,'(',ene_3pct,'%)',
     &            'sdev=',ene_3dev
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,2x,a,f5.2,a,5x,a,f10.5)')
     & 'Segment length  :',slbar     ,'+/-',slerr     ,'(',slpct,'%)',
     &            'sdev=',sldev
      WRITE(2,'(1h ,a,f12.5,2x,a,f10.5,2x,a,f5.2,a,5x,a,f10.5)')
     & 'Cluster size    :',csbar     ,'+/-',cserr     ,'(',cspct,'%)',
     &            'sdev=',csdev
      WRITE(2,'(1h ,a,f5.2,a)')
     & '(csbar/ns = ',100.*csbar/FLOAT(ns),'%)'
c
```

```
c for correlation lengths
      IF (xj.GT.0.) CALL docorr1d
c
      OPEN (UNIT=7,FILE='lattmpas.sum',STATUS='UNKNOWN',
     &      ACCESS='APPEND')
      WRITE(7,'(1h ,f6.2,1x,f5.0,1x,f5.0,1x,'//
     &        'f12.5,1x,f9.5,2x,'//
     &        'f12.5,1x,f9.5,2x,'//
     &        'f12.5,1x,f9.5,2x,'//
     &        'f12.5,1x,f9.5,2x,'//
     &        'i7,1x,i7)')
     &beta,FLOAT(nx),FLOAT(ny),
     &susbar,suserr,
     &stagsusbar,stagsuserr,
     &enebar,eneerr,
     &xi,xierr,
     &mused_act,nseg_act
      CLOSE (UNIT=7)
c
c binning table >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
c
      IF (do_table) THEN
      x = gs*ns*beta*ABS(xj)/2.   ! conjecture for mean xlambda (= x/2)
c
c compute fractions
         DO ix=0,nm
         qm_accum(ix) = FLOAT(m_accum(ix))/FLOAT(nloop-nexclude)
         dm_accum(ix) = SQRT(FLOAT(m_accum(ix)))/FLOAT(nloop-nexclude)
         ENDDO
c
c precompute exact answers
      qm_0 = qm_exact(0,x)
c
         DO ix=2,20,2
         qm_store(ix/2) = qm_exact(ix,x)
         ENDDO
c
c show the statistics on small m
      WRITE(2,'(1h ,11(i6,1x))') (ix,ix=0,20,2)
      WRITE(2,'(1h ,11(i6,1x))') (m_accum(ix),ix=0,20,2)
      WRITE(2,'(1h ,11(f6.4,1x))') (qm_accum(ix),ix=0,20,2)
      WRITE(2,'(1h ,11(f6.4,1x))') qm_0,(qm_store(ix),ix=1,10)
c
c make flags (don't even bother flagging m=0 case)
      line = ' '
c
         DO ix=2,20,2
            IF (m_accum(ix).GT.4) THEN
            IF (qm_store(ix/2).LT.(qm_accum(ix)-dm_accum(ix)))
     &      line(7*ix/2+2:7*ix/2+7) = '^^^^^^'  ! data are high
            IF (qm_store(ix/2).GT.(qm_accum(ix)+dm_accum(ix)))
     &      line(7*ix/2+2:7*ix/2+7) = 'vvvvvv'  ! data are low
            ENDIF
         ENDDO
c
      WRITE(2,'(a)') line
c
c additional output sections
```

```
      iout1 = 22
c
         DO WHILE (iout1.LE.nm .AND. (m_accum(iout1).GT.0
     &                                .OR.iout1.LT.IFIX(x/2.)))
         iout2 = iout1 + 18
         IF (iout2.GT.nm) iout2 = nm
c
c precompute exact answers
            DO ix=iout1,iout2,2
            qm_store((ix-iout1)/2 + 1) = qm_exact(ix,x)
            ENDDO
c
         WRITE(2,'(//)')
         WRITE(2,'(1h ,7x,10(i6,1x))') (ix,ix=iout1,iout2,2)
         WRITE(2,'(1h ,7x,10(i6,1x))') (m_accum(ix),ix=iout1,iout2,2)
         WRITE(2,'(1h ,7x,10(f6.4,1x))') (qm_accum(ix),ix=iout1,iout2,2)
         WRITE(2,'(1h ,7x,10(f6.4,1x))')
     &                          (qm_store(ix),ix=1,(iout2-iout1)/2+1)
c
c make flags
         line = ' '
c
            DO ix=1,(iout2-iout1)/2+1
               IF (m_accum(iout1+2*(ix-1)).GT.4) THEN
               IF (qm_store(ix).LT.
     &             (qm_accum(iout1+2*(ix-1))-dm_accum(iout1+2*(ix-1))))
     &         line(7*ix+2:7*ix+7) = '^^^^^^'  ! data are high
               IF (qm_store(ix).GT.
     &             (qm_accum(iout1+2*(ix-1))+dm_accum(iout1+2*(ix-1))))
     &         line(7*ix+2:7*ix+7) = 'vvvvvv'  ! data are low
               ENDIF
            ENDDO
c
         WRITE(2,'(a)') line
         iout1 = iout1 + 20
         ENDDO
c
      ENDIF
c
c end binning table <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
c
c write storage statistics
      WRITE(2,'(/1h ,a)')
     &'This data run used the following actual storage:'
      WRITE(2,'(/1h ,12x,a,i6,a,i6,a,f4.1,a)')
     &'mused:',mused_act,'        vs. nm      = ',nm,
     &' (',100.*FLOAT(mused_act)/FLOAT(nm),'%)'
      WRITE(2,'(/1h ,12x,a,i6,a,i6,a,f4.1,a)')
     &'nseg :',nseg_act, '        vs. nsegmax = ',nsegmax,
     &' (',100.*FLOAT(nseg_act)/FLOAT(nsegmax),'%)'
c
      RETURN
      END
c=======================================================================
      REAL*4 FUNCTION qm_exact(m,x)
c
c returns a semi-Poisson distribution for xlambda = x/2
c (this is not really the 'exact' binning distribution but it should be
```

```
c pretty close for large x)
      INTEGER*4 m
      REAL*4 x
      INTEGER*4 mx
c
c protect against floating point overflows...
c Note 175 is approximately 2*ACOSH(2**126)...
c this can be increased if we were to go to double precision....
      IF (x.GT.175.) THEN
      qm_exact = 0.
      RETURN
      ENDIF
c
c implement semi-Poisson distribution
      IF (m.EQ.0) THEN
      qm_exact = 1./COSH(x/2.)
c
      ELSE IF (m.GT.0) THEN
c
c unfortunately, we get a floating point overflow for m>33, so
c we need to be a little tricky about the rest of this computation.
c the "correct" expression for qm_exact for m>0 is
c     qm_exact = (1./COSH(x/2.))*(x/2.)**m)/factorial(m)
c but we need to group factors of (x/2) and terms in m! together
c to prevent an overflow
      qm_exact = 1./COSH(x/2.)
c
         DO mx=1,m
         qm_exact = qm_exact*(x/2.)/FLOAT(mx)
         ENDDO
c
      ENDIF
c
      RETURN
      END
c=======================================================================
      SUBROUTINE pathdump
c
c records relevant observables
      INCLUDE 'lattmpas.hdr'
c
      CHARACTER*1 CHAR   ! function type declaration
      REAL*4 clustersize ! function type declaration
c
      INTEGER*4 iseg
      CHARACTER*1 formfeed
c
      formfeed = CHAR(12)
      IF (iloop.GT.1) WRITE(3,'(a)') formfeed
c
      WRITE(3,'(1h ,a,i5,T60,a,1x,a/)')
     &'Path Dump for LATTM configuration ',iloop,today,now
c
      CALL paintpath(3)
c
      WRITE(3,'(/1h ,a,i5)')  'Number of segments = ',nseg
      cs = clustersize()
      sl = cs/FLOAT(nseg) ! average segment length
```

```
      WRITE(3,'(1h ,a,f12.5)') 'Total Path Length  = ',cs
      WRITE(3,'(1h ,a,f12.5)') 'Avg Segment Length = ',sl
c
      WRITE(3,'(/1h ,T10,a,T20,a,T29,a,T39,a,T49,a,T59,a,T66,a,T74,a)')
     &'iseg','SPIN','STATE','TAIL','TIP','segptr','cross','length'
c
      DO iseg=1,nseg
c
      WRITE(3,'(1h ,T8,i5,T18,i5,T30,i2,'
     &//'T36,f8.5,T46,f8.5,T58,i5,T67,i2,T73,f7.4)')
     &iseg,
     &path(iseg).spin,
     &path(iseg).state,
     &path(iseg).t(0),
     &path(iseg).t(1),
     &path(iseg).segptr,
     &path(iseg).icross,
     &path(iseg).length
c
      ENDDO
c
      WRITE(3,'(/1h ,a,i5/)') 'Winding Number = ',nwinding
c
      RETURN
      END
c=======================================================================
      SUBROUTINE paintpath(iunit)
c
c routine to make picture of path
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 iunit
c
      INTEGER*4 is,iseg
      INTEGER*4 ix,jx,ix1,ix2,ixx,jxx,ixd,my
      INTEGER*4 nscr
      PARAMETER(nscr=64)
      CHARACTER*1 screen(nscr,3*ns)
c
c check for lattice type
      IF (nntype.EQ.1) THEN
      CALL paintpath1(iunit)
      RETURN
      ELSE IF (nntype.EQ.2) THEN
      CALL paintpath2(iunit)
      RETURN
      ENDIF
c
c clear screen
      DO ix=1,nscr
         DO jx=1,3*ns
         screen(ix,jx) = ' '
         ENDDO
      ENDDO
c
c plot starting point
      ix = IFIX(nscr*path(1).t(0)) + 1
      jx = 3*path(1).spin - 1
```

```
      screen(ix,jx) = 'x'
      DO iseg=1,nseg
c
c do segment
      jx = 3*path(iseg).spin - 1
c
         IF (path(iseg).t(0).NE.path(iseg).t(1)) THEN
         ix1 = IFIX(nscr*path(iseg).t(1-path(iseg).state)) + 1
         ix2 = IFIX(nscr*path(iseg).t(  path(iseg).state)) + 1
         ixd = ix2 - ix1 + 1
         IF (ixd.LT.1) ixd = ixd + nscr
c
         ELSE
         ix1 = 1
         ixd = nscr
c
         ENDIF
c
         DO ixx=1,ixd
         ix = ix1 + ixx - 1
         IF (ix.GT.nscr) ix = ix - nscr
c
            IF (screen(ix,jx).EQ.' ') THEN
               IF (path(iseg).state.EQ.1) THEN
               screen(ix,jx) = '>' ! forward segment
               ELSE
               screen(ix,jx) = '<' ! backward segment
               ENDIF
            ELSE IF (screen(ix,jx).NE.'x') THEN
            screen(ix,jx) = '+'  ! symbol for overlap
            ENDIF
c
         ENDDO
c
c do bond
         IF (iseg.LT.nseg) THEN
         ix = IFIX(nscr*path(iseg).t(1)) + 1
c
            DO jxx=1,2
c
               IF (path(iseg+1).spin .EQ.
     &             neighbor(path(iseg).spin,2)) THEN
               jx = jx + 1
               IF (jx.GT.3*ns) jx = jx - 3*ns
c
                  IF (screen(ix,jx).EQ.' ') THEN
                  screen(ix,jx) = 'v'
                  ELSE
                  screen(ix,jx) = '+'
                  ENDIF
c
               ELSE
               jx = jx - 1
               IF (jx.LT.1) jx = jx + 3*ns
c
                  IF (screen(ix,jx).EQ.' ') THEN
                  screen(ix,jx) = '^'
                  ELSE
```

```
                  screen(ix,jx) = '+'
                  ENDIF
c
               ENDIF
c
            ENDDO
c
         ENDIF
c
      ENDDO
c
c overlay transitions
      DO is=1,ns
      jx = 3*is - 1
c
         IF (mtrans(is).GT.0) THEN
         my = mfirst(is)
            DO ixx=1,mtrans(is)
            ix = IFIX(nscr*trans(my)) + 1
               IF (screen(ix,jx).NE.'x') THEN
               screen(ix,jx) = '|'
               ELSE
               screen(ix,jx) = '*'
               ENDIF
            my = iptr(my,1)
            ENDDO
         ENDIF
c
      ENDDO
c
      DO jx=1,3*ns
         IF (MOD(jx,3).EQ.2) THEN
         is = (jx + 1)/3
         WRITE(iunit,'(1h ,i5,1x,i3.3,i2,2x,120a)')
     &   is,mtrans(is),isref(is),(screen(ix,jx),ix=1,nscr)
         ELSE
         WRITE(iunit,'(1h ,13x,120a)') (screen(ix,jx),ix=1,nscr)
         ENDIF
      ENDDO
c
      RETURN
      END
c=======================================================================
      SUBROUTINE paintpath1(iunit)
c
c routine to make picture of path for 2-D square lattice
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 iunit
c
      WRITE(iunit,'(/1h ,a/)') 'Sorry, PAINTPATH not yet implemented '
     &//'for this lattice type'
c
      RETURN
      END
c=======================================================================
      SUBROUTINE paintpath2(iunit)
c
```

```
c routine to make picture of path for 2-D hex lattice
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 iunit
c
      WRITE(iunit,'(/1h ,a/)') 'Sorry, PAINTPATH not yet implemented '
     &//'for this lattice type'
c
      RETURN
      END
c=======================================================================
      SUBROUTINE wrtstep
c
c handles output for each step
c [called only if step-by-step output is desired]
      INCLUDE 'lattmpas.hdr'
c
c depict lines as 64 character hexes (256 slices)
      INTEGER*4 nhex
      PARAMETER(nhex=8)
      INTEGER*4 nc
      PARAMETER(nc=64)
      INTEGER*4 nt
      PARAMETER(nt=256)
c
      INTEGER*4 hex(ns,nhex)
      CHARACTER*(nc) chgbar
      CHARACTER*1 chgflag(ns)
      CHARACTER*(nc) chhex(ns)
      CHARACTER*(nc) chhexold(ns)
      COMMON /wrtstuff/ chhexold ! save between calls
c
c some local variables
      INTEGER*4 is,itdo,ix,jx,nxx,mx,icaret
      INTEGER*4 istate
      REAL*4 tdo
      LOGICAL mdone
c
d     WRITE(*,'(1h ,a)') 'Entering WRTSTEP'
c
      WRITE(2,'(1h ,a,i5)') 'Winding Number = ',nwinding
c
c generate discretized output
      DO is=1,ns
      istate = isref(is)   ! initialize the state
      mx = mfirst(is)
      mdone = .FALSE.
c
         DO ix=1,nhex
         hex(is,ix) = 0
c
            DO jx=0,31
            itdo = 32*(ix-1)+jx+1
            tdo = FLOAT(itdo)/256.
c
c count the number of transitions in this time slice
               IF (mtrans(is).GT.0) THEN
c
```

```
                  DO WHILE(.NOT.mdone.AND.trans(mx).LE.tdo)
                  IF (mx.EQ.iptr(mfirst(is),0)) mdone = .TRUE.
                  istate = 1 - istate
                  mx = iptr(mx,1)
                  ENDDO
c
               ENDIF
c
            IF (istate.eq.1) hex(is,ix) = IBSET(hex(is,ix),31-jx)
c
            ENDDO
c
         ENDDO
c
      ENDDO
c
c create "change bar"
      chgbar = ' '
c
      DO is=1,ns
      WRITE(chhex(is),'(8'//zfmt//')') (hex(is,ix),ix=1,nhex)
      chgflag(is) = ' '
c
         IF (iloop.GT.0) THEN
c
c put flags next to changed elements
            DO nxx=1,nc
               IF (chhexold(is)(nxx:nxx).NE.chhex(is)(nxx:nxx)) THEN
               chgbar(nxx:nxx) = '='
               chgflag(is) = '<'
               ENDIF
            ENDDO
c
         ENDIF
c
      ENDDO
c
      IF (iloop.GT.0) THEN
      icaret = 1 + IFIX(tstart*nt)/4
      chgbar(icaret:icaret) = 'x'
      chgflag(nstart) = 'x'
      ENDIF
c
      WRITE(2,'(1h ,6x,a)') chgbar
c
      DO is=1,ns
      WRITE(2,'(1h ,i3.3,3x,a,2x,a)') mtrans(is),chhex(is),chgflag(is)
      chhexold(is) = chhex(is)  ! save the hex output
      ENDDO
c
      WRITE(2,'(1h ,6x,a)') chgbar
c
d     WRITE(*,'(1h ,a)') '>Leaving WRTSTEP'
c
      RETURN
      END
c=======================================================================
      SUBROUTINE integ
```

```
c
c performs integrity check on pointers...multi-spin version
      INCLUDE 'lattmpas.hdr'
c
      INTEGER*4 my, ix, is, msum
      LOGICAL okay
c
d     WRITE(*,'(1h ,T50,a,3i3,f7.4)') 'Entering INTEG '
d    &                            ,istatecurr,iscurr,mcurr,tcurr
c
      okay = .TRUE.
      msum = 0
c
      DO is=1,ns
c
         IF (mtrans(is).GT.0) THEN
         msum = msum + mtrans(is)
c
         my = mfirst(is)
c
            DO ix=1,mtrans(is)
            IF (iptr(iptr(my,0),1).NE.my) okay = .FALSE.
            IF (iptr(iptr(my,1),0).NE.my) okay = .FALSE.
c
c check time order
            IF (ix.LT.mtrans(is).AND.
     &          trans(iptr(my,1)).LT.trans(my)) okay = .FALSE.
            my = iptr(my,1)
            ENDDO
c
c should have returned to mfirst()
         IF (my.NE.mfirst(is)) okay = .FALSE.
c
c repeat for backwards direction
         my = mfirst(is)
c
            DO ix=1,mtrans(is)
            IF (iptr(iptr(my,0),1).NE.my) okay = .FALSE.
            IF (iptr(iptr(my,1),0).NE.my) okay = .FALSE.
            my = iptr(my,0)
            ENDDO
c
c should have returned to mfirst()
         IF (my.NE.mfirst(is)) okay = .FALSE.
c
         ELSE
         IF(mfirst(is).NE.0) okay = .FALSE.
c
         ENDIF
c
      ENDDO
c
      IF (msum.NE.mused) okay = .FALSE.
c
      IF (.NOT.okay) THEN ! fatal error
      WRITE(*,'(/1h ,a)') '***** FAILED INTEGRITY CHECK *****'
      CALL datadump('Failed pointer table integrity check')
      STOP 'rats'
```

```
      ENDIF
c
d     WRITE(*,'(1h ,T50,a,3i3,f7.4)') '>Leaving INTEG '
d    &                            ,istatecurr,iscurr,mcurr,tcurr
c
      RETURN
      END
c=======================================================================
      SUBROUTINE datadump(errmsg)
c
c dumps values of internal variables to a file
      INCLUDE 'lattmpas.hdr'
c
      CHARACTER*1 CHAR   ! function type declaration
c
      INTEGER*4 in,is,iline,icap,i1,i2,mx(ns),is1,is2
      INTEGER*4 ix
      CHARACTER*80 line
      LOGICAL notdone
      INTEGER*4 iseg
      CHARACTER*(*) errmsg
      CHARACTER*1 formfeed
c
      formfeed = CHAR(12)
c
c      OPEN (UNIT=4,FILE='lattmpas.dmp',STATUS='NEW',
c     &      CARRIAGECONTROL='LIST')  ! not recognized by MS FORTRAN 5.1
      OPEN (UNIT=4,FILE='lattmpas.dmp',STATUS='NEW')
c
      WRITE(4,'(1h ,a,T60,a,1x,a/)')
     &           'DUMP OF LATTM INTERNAL VARIABLES',today,now
      WRITE(4,'(1h ,a,i5/)') 'Configuration number iloop = ',iloop
      WRITE(4,'(1h ,a/)') errmsg
      WRITE(4,'(1h ,a,i5)') 'Lattice Type code (nntype)   = ',nntype
      WRITE(4,'(1h ,a,i5)') 'Number of Neighbors (nn)     = ',nn
      WRITE(4,'(1h ,a,i5)') 'Number of spin sites (ns)    = ',ns
      WRITE(4,'(1h ,a,i5)') 'Number of storage slots (nm) = ',nm
c
      WRITE(4,'(/1h ,a,L1)') 'Staggered Start : ',stag_start
      WRITE(4,'(1h ,a,L1/)') 'Random Start    : ',rand_start
c
      WRITE(4,'(1h ,a,f8.2)') 'Simulation Beta = ',beta
      WRITE(4,'(1h ,a,f8.3)') 'Simulation J    = ',xj
      WRITE(4,'(1h ,a,f8.3)') 'Latest Lambda   = ',xlambda
c
      WRITE(4,'(/1h ,a,i5)')
     &'Starting Spin for Latest Configuration (nstart) = ',nstart
      WRITE(4,'(1h ,a,f8.5/)')
     &'Starting Time for Latest Configuration (tstart) = ',tstart
c
      WRITE(4,'(1h ,a/)') 'Local pointers and time markers:'
      WRITE(4,'(1h ,a,i5)')   'iscurr     = ',iscurr
      WRITE(4,'(1h ,a,i5)')   'isprev     = ',isprev
      WRITE(4,'(1h ,a,i5)')   'isnext     = ',isnext
      WRITE(4,'(1h ,a,i5)')   'istatecurr = ',istatecurr
      WRITE(4,'(1h ,a,i5)')   'mcurr      = ',mcurr
      WRITE(4,'(1h ,a,f8.5)') 'tcurr      = ',tcurr
      WRITE(4,'(1h ,a,f8.5)') 'tpost      = ',tpost
```

```
      WRITE(4,'(1h ,a,f8.5)') 'thoriz     = ',thoriz
      WRITE(4,'(1h ,a,f8.5)') 'tway       = ',tway
c
      WRITE(4,'(/1h ,a,i5)') 'Number of used slots (mused) = ',mused
c
      iline = 0
      icap = 10
      DO WHILE (10*iline.LT.ns)
      IF (10*iline+10.GT.ns) icap = ns - (10*iline)
      is1 = 10*iline + 1
      is2 = 10*iline + icap
c
      WRITE(4,'(//1h ,a,i5,a,i5/)')
     &'Data for spin sites ',is1,' through ',is2
c
      WRITE(4,'(1h ,a,10(i5,2x))') 'is        ',(is,is=is1,is2)
      WRITE(4,'(1h ,a,10(i5,2x))') 'isref     ',(isref(is),is=is1,is2)
      WRITE(4,'(1h ,a,10(i5,2x))') 'mtrans    ',(mtrans(is),is=is1,is2)
      WRITE(4,'(1h ,a,10(i5,2x))') 'mfirst    ',(mfirst(is),is=is1,is2)
      WRITE(4,'(1h ,a,10(i5,2x))') 'isegptr   ',(isegptr(is),is=is1,is2)
c
         DO in=1,nn
         WRITE(4,'(1h ,a,i1,a,10(i5,2x))')
     &   'nbr(is,',in,') ',(neighbor(is,in),is=is1,is2)
         ENDDO
c
c dump out times in order
         DO is=is1,is2
         mx(is) = mfirst(is)
         ENDDO
c
      notdone = .TRUE.
c
         DO WHILE(notdone)
         line = ' (time)    '
c
            DO is=is1,is2
            i1 = 11 + 7*(is - 10*iline - 1)
            i2 = i1 + 6
c
               IF (mx(is).NE.0) THEN
               WRITE(line(i1:i2),'(f7.4)') trans(mx(is))
               mx(is) = iptr(mx(is),1)
               ENDIF
c
            IF (mx(is).EQ.mfirst(is)) mx(is) = 0
            ENDDO
c
         WRITE(4,'(a)') line ! always write out at least one line
c
         notdone = .FALSE.
c
            DO is=is1,is2
            IF (mx(is).NE.0) notdone = .TRUE.
            ENDDO
c
         ENDDO
c
```

```
c proceed to next set of spin sites
      WRITE(4,'(//)')
      iline = iline + 1
      ENDDO
c
      WRITE(4,'(1h ,a)') 'Latest neighbor pointers:'
      WRITE(4,'(1h ,a,6(i5,3x))')    'in       = ',(in,in=1,nn)
      WRITE(4,'(1h ,a,6(i5,3x))')    'mlnn     = ',(mlnn(in),in=1,nn)
      WRITE(4,'(1h ,a,6(4x,L1,3x))') 'mask     = ',(mask(in),in=1,nn)
      WRITE(4,'(1h ,a,6(i5,3x))')    'nway     = ',(nway(in),in=1,nn)
c
      WRITE(4,'(/1h ,a,i5)')  'nactive = ',nactive
      WRITE(4,'(1h ,a,i5)')   'nnpick  = ',nnpick
      WRITE(4,'(1h ,a,f8.4)') 'tway    = ',tway
      WRITE(4,'(/1h ,a,L1)')  'lastway : ',lastway
c
      WRITE(4,'(a)') formfeed
c
      WRITE(4,'(1h ,a)') 'Dump of large arrays:'
c
      WRITE(4,'(/1h ,a/)') 'Transition Times (trans):'
      iline = 0
      icap = 10
      DO WHILE (10*iline.LT.nm)
      IF (10*iline+10.GT.nm) icap = nm - (10*iline)
      WRITE(4,'(1h ,10(f7.4,1x))')
     &(trans(ix),ix=10*iline+1,10*iline+icap)
      iline = iline + 1
      ENDDO
c
      WRITE(4,'(/1h ,a/)') 'Bonding Info (ibond):'
      iline = 0
      icap = 10
      DO WHILE (10*iline.LT.nm)
      IF (10*iline+10.GT.nm) icap = nm - (10*iline)
      WRITE(4,'(1h ,10(i5,3x))')
     &(ibond(ix),ix=10*iline+1,10*iline+icap)
      iline = iline + 1
      ENDDO
c
      WRITE(4,'(/1h ,a/)') 'Forward Pointers (iptr(?,1)):'
      iline = 0
      icap = 10
      DO WHILE (10*iline.LT.nm)
      IF (10*iline+10.GT.nm) icap = nm - (10*iline)
      WRITE(4,'(1h ,10(i5,3x))')
     &(iptr(ix,1),ix=10*iline+1,10*iline+icap)
      iline = iline + 1
      ENDDO
c
      WRITE(4,'(/1h ,a/)') 'Backward Pointers (iptr(?,0)):'
      iline = 0
      icap = 10
      DO WHILE (10*iline.LT.nm)
      IF (10*iline+10.GT.nm) icap = nm - (10*iline)
      WRITE(4,'(1h ,10(i5,3x))')
     &(iptr(ix,0),ix=10*iline+1,10*iline+icap)
      iline = iline + 1
```

```
      ENDDO
c
      WRITE(4,'(/1h ,a/)') 'Unused Pointers (iunused):'
      iline = 0
      icap = 10
      DO WHILE (10*iline.LT.nm)
      IF (10*iline+10.GT.nm) icap = nm - (10*iline)
      WRITE(4,'(1h ,10(i5,3x))')
     &(iunused(ix),ix=10*iline+1,10*iline+icap)
      iline = iline + 1
      ENDDO
c
      WRITE(4,'(a)') formfeed
c
c dump of latest path
c
      CALL paintpath(4)
c
      WRITE(4,'(/1h ,a,i5)') 'Number of segments = ',nseg
      WRITE(4,'(/1h ,T10,a,T20,a,T29,a,T39,a,T49,a,T59,a,T66,a,T74,a)')
     &'iseg','SPIN','STATE','TAIL','TIP','segptr','cross','length'
c
c note since this routine may get called before subr "measure"
c we compute the winding number independently of that routine...
      nwinding = 0
      DO iseg=1,nseg
c
      nwinding = nwinding + path(iseg).icross
c
      WRITE(4,'(1h ,T8,i5,T18,i5,T30,i2,'
     &//'T36,f8.5,T46,f8.5,T58,i5,T67,i2,T73,f7.4)')
     &iseg,
     &path(iseg).spin,
     &path(iseg).state,
     &path(iseg).t(0),
     &path(iseg).t(1),
     &path(iseg).segptr,
     &path(iseg).icross,
     &path(iseg).length
c
      ENDDO
c
      WRITE(4,'(/1h ,a,i5/)') 'Winding Number = ',nwinding
c
      WRITE(4,'(//1h ,a)') 'Better luck next time!'
c
      CLOSE (UNIT=4)
c
      RETURN
      END
```

"Here I go when I don't know why
I spin so ceaselessly
'Til I lose my sense of gravity"
– Patti Smith

Appendix B. The 2-spin system

It is helpful in some of the other derivations to have handy the results for the system composed of 2 spin-1/2 particles. As we shall see, implicit in even this simple case is the importance of tracking the number and distribution of the transitions from one state to another.

The Heisenberg Hamiltonian is the sum over nearest neighbors

$$\mathrm{H} = J\sum_{\langle ij\rangle}\sigma_i\cdot\sigma_j$$

For the system consisting only of two spins, this Hamiltonian is then diagonal in the basis $|s,m_s\rangle$, with eigenvalues $+\frac{1}{4}J$ for the triplet state and $-\frac{3}{4}J$ for the singlet state. The ground state and its degeneracy are determined by the sign of J. The partition function is

$$Z \equiv \mathrm{Tr}\left(\exp(-\beta H)\right) = 3\exp(-\tfrac{1}{4}\beta J) + \exp(+\tfrac{3}{4}\beta J)$$

which is, of course, independent of basis.

Systems with many spins on a lattice are generally not so easily diagonalized; it is usually convenient to work in the "spin-up/spin-down" basis; i.e. the state of the system at any time is characterized by the specification of spin projections along some quantization axis for each of the constituent spins. For N lattice sites, this basis contains 2^N elements. For the two-spin system, the singlet-triplet states have the familiar form

$$|s,m_s\rangle = \begin{cases} |1,1\rangle = |\uparrow\uparrow\rangle \\ |1,0\rangle = \frac{1}{\sqrt{2}}\left(|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle\right) \\ |1,-1\rangle = |\downarrow\downarrow\rangle \\ |0,0\rangle = \frac{1}{\sqrt{2}}\left(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle\right) \end{cases}$$

$$\Rightarrow \begin{cases} |\uparrow\downarrow\rangle = \frac{1}{\sqrt{2}}\left(|1,0\rangle + |0,0\rangle\right) \\ |\downarrow\uparrow\rangle = \frac{1}{\sqrt{2}}\left(|1,0\rangle - |0,0\rangle\right) \end{cases}$$

Trotter-Suzuki Decomposition of the Two-spin system

We can effect a Trotter-Suzuki decomposition for the two-spin system. We first split up the partition function

$$Z = \sum_n \langle n|\exp(-\beta H)|n\rangle = \sum_n \langle n|\left(\exp(-\varepsilon\beta H)\right)^N|n\rangle$$

with $N\varepsilon = 1$. A complete set of states is inserted between each operator

$$Z = \sum_{\{n_k\}} \langle n_0|\exp(-\varepsilon\beta H)|n_1\rangle\langle n_1|\ldots|n_{N-1}\rangle\langle n_{N-1}|\exp(-\varepsilon\beta H)|n_0\rangle$$

Then a typical factor in this sum over products is

$$\langle n_k|\exp(-\varepsilon\beta H)|n_{k+1}\rangle = \exp(-\tfrac{1}{4}\varepsilon\beta J)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}\left(1+e^{\varepsilon\beta J}\right) & \frac{1}{2}\left(1-e^{\varepsilon\beta J}\right) & 0 \\ 0 & \frac{1}{2}\left(1-e^{\varepsilon\beta J}\right) & \frac{1}{2}\left(1+e^{\varepsilon\beta J}\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}_{k,k+1}$$

The N^{th} power of this matrix is straightforwardly computed to be

$$\prod_{k=0}^{N-1}\langle n_k|\exp(-\varepsilon\beta H)|n_{k+1}\rangle = \exp(-\tfrac{1}{4}N\varepsilon\beta J)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}\left(1+e^{N\varepsilon\beta J}\right) & \frac{1}{2}\left(1-e^{N\varepsilon\beta J}\right) & 0 \\ 0 & \frac{1}{2}\left(1-e^{N\varepsilon\beta J}\right) & \frac{1}{2}\left(1+e^{N\varepsilon\beta J}\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

or, since $N\varepsilon = 1$,

$$\prod_{k=0}^{N-1}\langle n_k|\exp(-\varepsilon\beta H)|n_{k+1}\rangle = \exp(-\tfrac{1}{4}\beta J)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2}\left(1+e^{\beta J}\right) & \frac{1}{2}\left(1-e^{\beta J}\right) & 0 \\ 0 & \frac{1}{2}\left(1-e^{\beta J}\right) & \frac{1}{2}\left(1+e^{\beta J}\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The partition function is the trace of this matrix, and we recover the previous result $Z = 3e^{-\frac{1}{4}\beta J} + e^{+\frac{3}{4}\beta J}$.

We can also express this same partition function in a multinomial (actually binomial) expansion. We first write the partition function explicitly

$$Z = Z_{\uparrow\uparrow} + Z_{\downarrow\downarrow} + Z_{\uparrow\downarrow} + Z_{\downarrow\uparrow}$$

meaning, e.g. $Z_{\uparrow\uparrow} = \langle\uparrow\uparrow|\exp(-\beta H)|\uparrow\uparrow\rangle$. We see from the transfer matrix above that the terms in Z that begin and end with $|\uparrow\uparrow\rangle$ are all zero unless every intervening state is also $|\uparrow\uparrow\rangle$. Thus one part of the partition function is

$$Z_{\uparrow\uparrow} = \prod_{k=0}^{N-1}\langle\uparrow\uparrow|\exp(-\varepsilon\beta H)|\uparrow\uparrow\rangle = \prod_{k=0}^{N-1}\exp(-\tfrac{1}{4}\varepsilon\beta J) = \exp(-\tfrac{1}{4}N\varepsilon\beta J) = \exp(-\tfrac{1}{4}\beta J)$$

Similarly, the terms that begin and end with $|\downarrow\downarrow\rangle$ are all zero unless every intervening state is also $|\downarrow\downarrow\rangle$. These terms contribute the same quantity

$$Z_{\downarrow\downarrow} = \prod_{k=0}^{N-1}\langle\downarrow\downarrow|\exp(-\varepsilon\beta H)|\downarrow\downarrow\rangle = \exp(-\tfrac{1}{4}\beta J)$$

The remaining terms are the result of summing over intermediate states that can be either $|\uparrow\downarrow\rangle$ or $|\downarrow\uparrow\rangle$. They are

$$Z_{\uparrow\downarrow} = \sum_{\{\pm\}}\prod_{k=0}^{N-1} e^{-\frac{1}{4}\varepsilon\beta J}\left(\frac{1\pm e^{\varepsilon\beta J}}{2}\right) = Z_{\downarrow\uparrow}$$

where the upper (plus) sign in the interior parentheses is for factors in which the bra and ket are the same state, the lower (minus) sign is for factors where the bra and ket are different states, and the sum is over all combinations of plus and minus signs. The sum can be regrouped into terms with exactly m transitions (minus sign terms), $0 \le m \text{ even} \le N$. (Since we return to the same state, the number of transitions must be even). There are exactly $\binom{N}{m}$ such terms. Thus

$$Z_{\uparrow\downarrow} = Z_{\downarrow\uparrow} = e^{-\frac{1}{4}\beta J}\sum_{\substack{m=0\\ m\text{ even}}}^{N}\binom{N}{m}\left(\frac{1+e^{\varepsilon\beta J}}{2}\right)^{N-m}\left(\frac{1-e^{\varepsilon\beta J}}{2}\right)^{m}$$

Recall that we know from the previous calculation that $Z_{\uparrow\downarrow} = Z_{\downarrow\uparrow} = e^{-\frac{1}{4}\beta J}\left(\frac{1+e^{\beta J}}{2}\right)$.

In fact, as shown below, it is straightforward to prove the more general equivalence

$$\sum_{\substack{m=0\\ m\text{ even}}}^{N}\binom{N}{m}\left(\frac{1+x}{2}\right)^{N-m}\left(\frac{1-x}{2}\right)^{m} = \left(\frac{1+x^{N}}{2}\right).$$

We thus reproduce the result $Z = Z_{\uparrow\uparrow} + Z_{\downarrow\downarrow} + Z_{\uparrow\downarrow} + Z_{\downarrow\uparrow} = 3e^{-\frac{1}{4}\beta J} + e^{+\frac{3}{4}\beta J}$.

For simplicity we can define

$$p \equiv \left(\frac{1-e^{\varepsilon\beta J}}{2}\right) \qquad q \equiv \left(\frac{1+e^{\varepsilon\beta J}}{2}\right).$$
$$p+q=1$$

Note for $0<\varepsilon<<1/\beta J$ we have $|p|<<1$ and $q\approx 1$. We note in passing that we expect the largest contribution to this sum to come from terms with many factors of q and relatively few factors of p; in other words, from paths that require few transitions from $|\uparrow\downarrow\rangle$ to $|\downarrow\uparrow\rangle$ or back.

Fleshing out the low-order binomial expansion

We can make explicit the goodness of the low-order approximation. We take $x \equiv \beta J$ and define the m^{th}-order terms

$$Z_m(m,x) \equiv \exp(-\tfrac{1}{4}x)\binom{N}{m} q^{N-m} p^m$$

(subscript m suggests the index m) and the partial sums of these terms

$$Z_p(m,x) \equiv \sum_{\substack{m'=0 \\ m' \text{ even}}}^{m} Z_m(m',x)$$

(subscript p suggests "partial"). These are simply the increasing order approximations to the partition function $Z_{\uparrow\downarrow} = Z_{\downarrow\uparrow} = Z_p(N,\beta J)$. Thus $Z_p(0,x) = e^{-\frac{1}{4}x} q^N$ is the approximation with no transitions, $Z_p(2,x)$ is the approximation with either zero or two transitions, and so on.

Consider as a concrete example the case with N=2048. In this case $\varepsilon \approx 1/20\,\%$. For small values of $x = \beta J$, the lowest order approximations give a good fit to the partition function. The following plot shows the analytic value of the partition function $Z_{\uparrow\downarrow} = \exp(-\tfrac{1}{4}x)(1+\exp(x))/2$ together with the three lowest-order approximations $Z_p(0,x)$, $Z_p(2,x)$, and $Z_p(4,x)$, over a range $-5 \le x \le 5$. The third order ($m=4$) fit looks reasonable over this range.

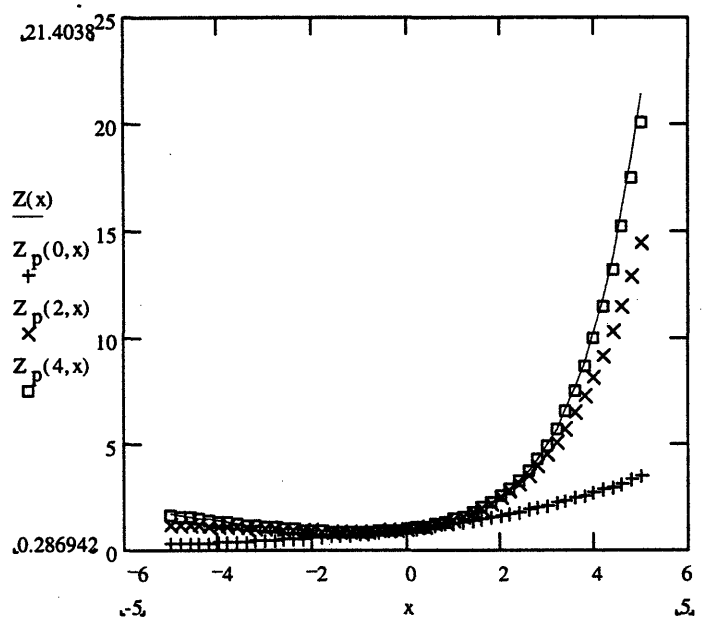


Figure B-1. Low-order approximations to the partition function

However, when we stray from the origin, the fit worsens exponentially. The following plot shows the same three lowest order approximations, expressed as ratios to the analytical expression, over the larger range $-50 \le x \le 50$.

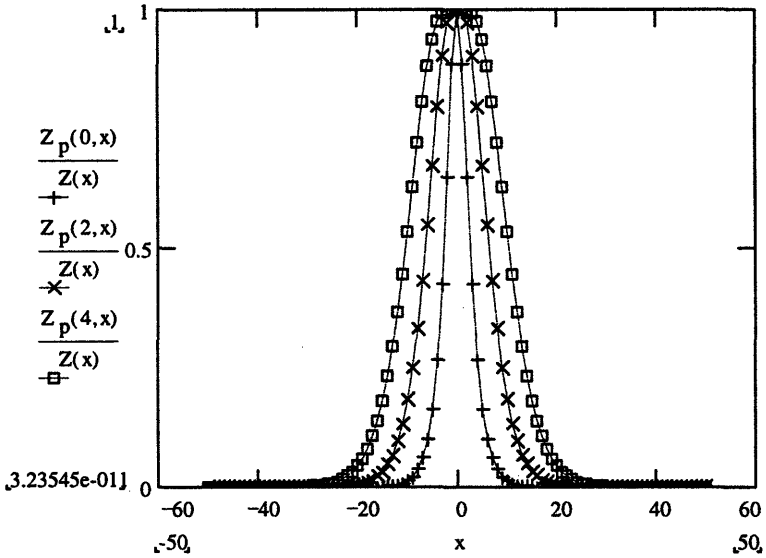


Figure B-2. Ratio of low-order approximants of Z to the partition function.

Clarification of the contribution of low-m terms

We have loosely treated the idea that the partition function is dominated by terms with few transitions. Here we clarify the notion by showing the calculation of low order terms for various parameter sweeps. In this appendix we label the number of bytes with N and the number of bits with $8N$.

We defined above the m^{th}-order terms

$$Z_m(m,x) \equiv \exp(-\tfrac{1}{4}x)\binom{8N}{m} q^{8N-m} p^m$$

and the partial sums of these terms

$$Z_p(m,x) \equiv \sum_{\substack{m'=0 \\ m' \text{ even}}}^{m} Z_m(m',x)$$

Let us also normalize the m^{th}-order terms by introducing

$$f_m(m,x) \equiv \frac{Z_m(m,x)}{Z(x)}$$

which is just the fraction of the partition function at any x given by the m^{th}-order term. The following plot shows the function f_m versus m for various x for the case $8N = 2048$.

By the definition of f_m, the sum over m of f_m for any fixed x is unity. We see that for $x = 0$, the partition function is entirely composed of the $m = 0$ term. As x increases, the partition function is spread out over a range of m values; the order of the term contributing the most to $Z(x)$ increases with x.

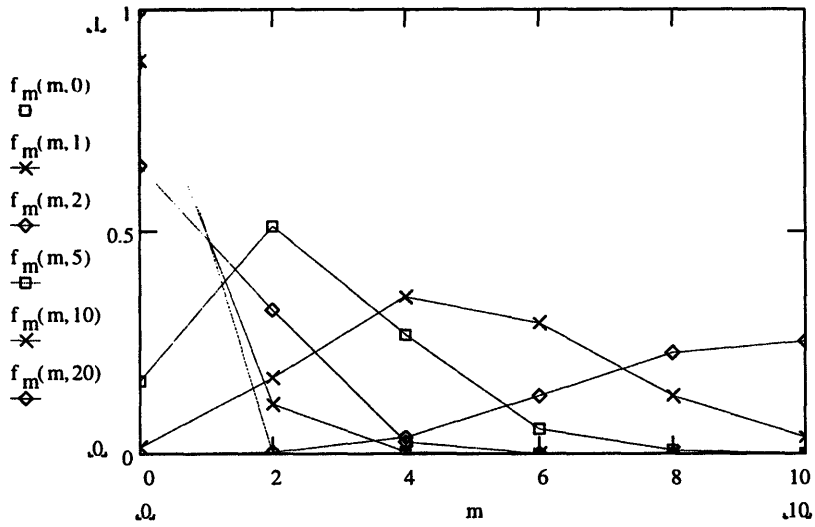


Figure B-3. Low-order terms in the partition function.

In fact, in the case where $x < 0$ (i.e. $J < 0$) so that $p > 0$, f_m qualifies as a discrete probability distribution in m – in fact a binomial distribution. Inserting the result for $Z(x)$, we find

$$f_m(m,x) = \frac{2}{1+e^x}\binom{8N}{m} q^{8N-m} p^m$$

where the normalization factor is present because we only take even m. The mean value of m is

$$\mu = (8N)(p) = 8N\left(\frac{1-\exp(x/8N)}{2}\right) \doteq 4N\left(\frac{-x}{8N}\right) = -\frac{x}{2}$$

for $|x| << 8N$. We conjecture that the analogous result for $x > 0$ is $\mu = +x/2$. The standard deviation for $x < 0$ is

$$\sigma = \sqrt{(8N)(p)(q)} = \sqrt{8N\left(\frac{1-\exp(x/8N)}{2}\right)\left(\frac{1+\exp(x/8N)}{2}\right)} = \sqrt{2N(1-\exp(x/4N))} \doteq \sqrt{-\frac{x}{2}}$$

These results match the trends shown in the plot above.

Another way of slicing the same data is to plot f_m versus x for various m. The next plot shows this variation. Again this plot is for $8N = 2048$.

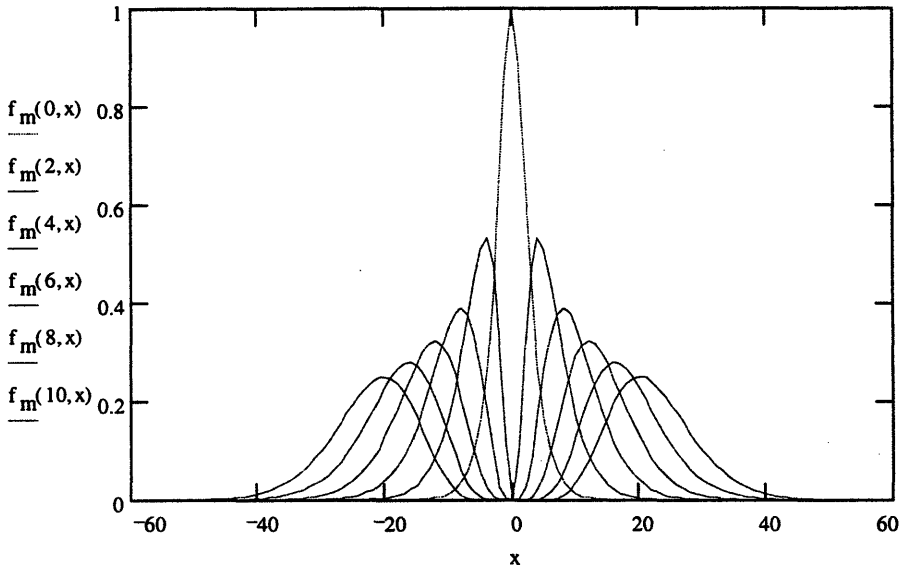


Figure B-4. Low order terms in the partition function versus x.

We see clearly the converse of the trend in the first plot: the value of x where the m^{th}-order term contributes the most increases with m.

Finally, we can show trends with increasing N. Here we plot f_m versus N and m for $x = 20$. We see that the fractions f_m are essentially insensitive to N for the region which we have been exploring.

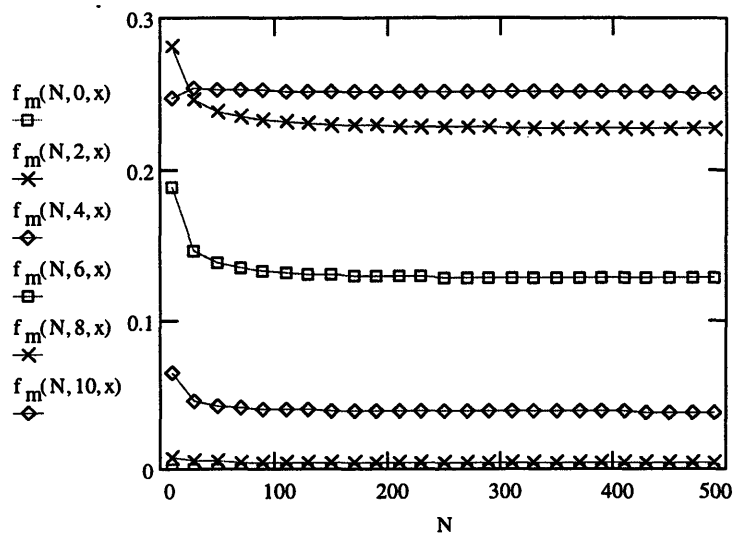


Figure B-5. Low order terms in the partition function versus N.

Aside: Proof of the relation

$$\sum_{\substack{m=0 \\ m \text{ even}}}^{N} \binom{N}{m} \left(\frac{1+x}{2}\right)^{N-m} \left(\frac{1-x}{2}\right)^{m} = \left(\frac{1+x^N}{2}\right)$$

Note first that since

$$\sum_{\substack{m=0 \\ (\text{all } m)}}^{N} \binom{N}{m} \left(\frac{1+x}{2}\right)^{N-m} \left(\frac{1-x}{2}\right)^{m} = 1$$

from the binomial theorem, the result to be proved implies also

$$\sum_{\substack{m=0 \\ m \text{ odd}}}^{N} \binom{N}{m} \left(\frac{1+x}{2}\right)^{N-m} \left(\frac{1-x}{2}\right)^{m} = \left(\frac{1-x^N}{2}\right)$$

The proof proceeds by induction. The cases N=1 and N=2 are trivially verified. If the theorem is true for N, then we can establish the result for N+1 by taking the even formula multiplied by $(1+x)/2$ and adding the odd result multiplied by $(1-x)/2$. We have

$$\left(\frac{1+x^N}{2}\right)\left(\frac{1+x}{2}\right)+\left(\frac{1-x^N}{2}\right)\left(\frac{1-x}{2}\right)=\left(\frac{1+x^{N+1}}{2}\right)$$

which proves the theorem.

The reason this works is that the summation is essentially a sum over products of N factors

$$\underbrace{\left(\frac{1\pm x}{2}\right)\left(\frac{1\pm x}{2}\right)\ \cdots\ \left(\frac{1\pm x}{2}\right)\left(\frac{1\pm x}{2}\right)}_{N\text{ factors}}$$

with an even number of minus signs chosen. The sum over products of N+1 factors must either terminate with a plus factor or a minus factor. If plus, then the preceding factors are required to have an even number of minus signs and hence sum to $\left(1+x^N\right)/2$. If minus, then the preceding factors are required to have an odd number of minus signs and hence sum to $\left(1-x^N\right)/2$.

Note this computation is identical to the matrix computation

$$\begin{pmatrix} \frac{1+x^N}{2} & \frac{1-x^N}{2} \\ \frac{1-x^N}{2} & \frac{1+x^N}{2} \end{pmatrix}\begin{pmatrix} \frac{1+x}{2} & \frac{1-x}{2} \\ \frac{1-x}{2} & \frac{1+x}{2} \end{pmatrix}=\begin{pmatrix} \frac{1+x^{N+1}}{2} & \frac{1-x^{N+1}}{2} \\ \frac{1-x^{N+1}}{2} & \frac{1+x^{N+1}}{2} \end{pmatrix}$$

which we implicitly used in calculating the partition function.

"All abstract sciences are nothing but the study of relations between signs."
– Denis Diderot

Appendix C. The sign estimator for the 3 spin system

The derivation of the sign estimator for the system composed of 3 spin ½ particles is presented here.

In order to illustrate the decomposition of the partition function into positive and negative contributions, we analyze explicitly the system composed of 3 spin ½ particles. From an eigenvalue analysis of the 3 spin system the partition function is

$$Z = 4\exp\left(-\tfrac{3}{4}\beta J\right) + 4\exp\left(\tfrac{3}{4}\beta J\right).$$

We can do a Trotter-Suzuki-like decomposition into N time slices, and let $\varepsilon = 1/N$, so that

$$Z = \mathrm{Tr}\left(\exp(-\beta H)\right) = \sum_{\{n_k\}} \prod_{\substack{k=0 \\ n_N = n_0}}^{N-1} \langle n_k | \exp(-\varepsilon\beta H) | n_{k+1} \rangle$$

The transfer matrix for this system is

$$M = \left(\begin{array}{c|ccc|ccc|c} x & & & & & & & \\ \hline & y & -z & -z & & & & \\ & -z & y & -z & & & & \\ & -z & -z & y & & & & \\ \hline & & & & y & -z & -z & \\ & & & & -z & y & -z & \\ & & & & -z & -z & y & \\ \hline & & & & & & & x \end{array}\right) \leftrightarrow \begin{pmatrix} 111 \\ 110 \\ 101 \\ 011 \\ 100 \\ 010 \\ 001 \\ 000 \end{pmatrix}$$

where as usual a 1 denotes a spin up site and a 0 denotes a spin down site. Note that the states $|100\rangle$ and $|011\rangle$ are out of order from the usual "binary-$|n\rangle$" basis, in order to show the block-diagonal form of the transfer matrix. The blocks themselves are associated with the conserved $m_s = \frac{3}{2}, \frac{1}{2}, -\frac{1}{2}, -\frac{3}{2}$. The quantities x, y, and z are all positive numbers for the antiferromagnetic ($J > 0$) coupling. We plan to absorb the minus signs in the off-diagonal elements into the estimator for sign. We have

$$x = \exp(-\tfrac{3}{4}\varepsilon\beta J)$$
$$y = \frac{1}{3}\left(\exp(-\tfrac{3}{4}\varepsilon\beta J) + 2\exp(\tfrac{3}{4}\varepsilon\beta J)\right)$$
$$z = \frac{1}{3}\left(\exp(\tfrac{3}{4}\varepsilon\beta J) - \exp(-\tfrac{3}{4}\varepsilon\beta J)\right)$$

The sign of each contribution to the partition function depends on the number of off-diagonal ("z") terms that appear in the product. To simplify our calculation, we can also segregate the terms according to the initial (t=0) states. We define

$$Z = \left(Z^+_{111} - Z^-_{111}\right) + \left(Z^+_{110} - Z^-_{110}\right) + \left(Z^+_{101} - Z^-_{101}\right) + \ldots + \left(Z^+_{000} - Z^-_{000}\right)$$

where the plus or minus superscript denotes the decomposition into even and odd sectors.

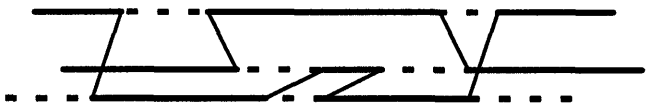
Note that transitions from the 111 and 000 states are not allowed; hence these sectors consist entirely of prisms mapping 111 to 111, etc., so that

$$Z^+_{111} = \left(\exp(-\tfrac{3}{4}\varepsilon\beta J)\right)^N = \exp(-\tfrac{3}{4}\beta J) = Z^+_{000}$$
$$Z^-_{111} = 0 = Z^-_{000}$$

Also note that symmetry requires

$$Z^+_{110} = Z^+_{101} = Z^+_{011} = Z^+_{100} = Z^+_{010} = Z^+_{001}$$
$$Z^-_{110} = Z^-_{101} = Z^-_{011} = Z^-_{100} = Z^-_{010} = Z^-_{001}$$

Hence we can concentrate on one particular initial configuration, say 110. In the Trotter-Suzuki picture, this configuration can undergo either the diagonal process "y", which corresponds to the continuation prism, or the off-diagonal process "z" which corresponds to a forced transition on one side of the prism. Using continuum-picture notation, a typical contribution to the partition function is



Here as usual a solid line indicates a succession of spin up sites; a dotted line, spin down. The solid lines crossing the prism faces are "bonds" indicating the location and orientation of "z" type prisms.

We would like to express the partial partition function $Z_{110} = Z^+_{110} - Z^-_{110}$ first as a sum

$$Z_{110} = \sum_{b=0}^{N} c_b y^{N-b} (-z)^b$$

(where we are to find the coefficients c_b) so that we may compute the + and - sectors as restrictions of this sum to even and odd b, respectively. The index b can be thought of as the number of bonds in a given configuration. The coefficient c_b is the number of distinct ways that b bonds can be arranged to give a configuration periodic in time. Then we will separate the sum into two sums, over even and odd b, which will be Z^{+}_{110} and $-Z^{-}_{110}$, respectively.

The coefficient c_b can be expressed as the product of two factors: a simple combinatoric factor indicating the number of ways of choosing b out of N prisms to be of the "z" type, and a geometry factor f_b that is the number of ways the b bonds can be arranged to give a configuration that is periodic in time. The first factor is just $\binom{N}{b}$.

The calculation of the geometry factor is slightly more involved. Let us examine some particular cases.

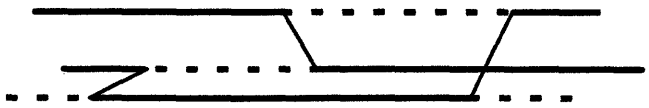
For $b=0$, obviously we define $f_b=1$. For $b=1$, we find $f_b=0$, since there are no configurations which have a single bond and are periodic in time.

For $b=2$, we find $f_b=2$, as follows:

and its mirror image

For $b=3$, we find $f_b=2$, as follows:

and its mirror image



We will proceed to the general case by recursion. First note that for each $b > 1$, there are precisely 2 distinct configurations that have the special property of *irreducibility*. An irreducible configuration is one which we cannot cut anywhere between the two end bonds and find a periodic subconfiguration. The first example bond configuration shown above (with 6 bonds) is not irreducible because we can cut the configuration after the third bond and find a periodic subconfiguration. An example of an irreducible configuration for $b = 8$ is

and another is its mirror image. A moment's reflection will convince us that these are the only two irreducible configurations for $b = 8$. The point is that in an irreducible configuration the spin down segment can only bounce back and forth between the two distal edges (the edges that start as spin up) -- as soon as it returns to the proximal edge (the edge starting spin down), the configuration is periodic. The same reasoning applies whether b is even or odd.

Now consider all the configurations with b bonds. Each configuration starts with some irreducible section of length p, for some p with $2 \le p \le b$. For each p the contribution to the total number of configurations is $2f_{b-p}$. Hence we can immediately write down a recursion relation for f_b as the sum over p:

$$f_b = 2f_{b-2} + 2f_{b-3} + 2f_{b-4} + \ldots + 2f_0$$

Conveniently, we can turn this recursion relation into a second order difference equation by examining f_{b-1}. It differs from f_b only in lacking the leading term $2f_{b-2}$; hence

$$f_b - f_{b-1} = 2f_{b-2} \Rightarrow f_{b+2} - f_{b+1} - 2f_b = 0$$

Instructed by the elementary theory of difference equations, we examine the related quadratic equation $x^2 - x - 2 = 0$ and find its roots $x = 2, -1$. Hence the general solution to this difference equation is

$$f_b = A \cdot 2^b + B \cdot (-1)^b$$

and the particular solution for the initial terms $f_0 = 1$, $f_1 = 0$ is

$$f_b = \tfrac{1}{3} \cdot 2^b + \tfrac{2}{3} \cdot (-1)^b$$

Thus we find

$$Z_{110} = \sum_{b=0}^{N} \left(\tfrac{1}{3} \cdot 2^b + \tfrac{2}{3} \cdot (-1)^b \right) \binom{N}{b} y^{N-b} (-z)^b$$

If the sum is restricted to even b, we get Z_{110}^{+}; with odd b, we get $-Z_{110}^{-}$.

To rearrange this sum, note the handy elementary relations

$$\sum_{\substack{m=0 \\ m \text{ even}}}^{N} \binom{N}{m} a^{N-m} b^{m} = \frac{(a+b)^{N} + (a-b)^{N}}{2}$$

$$\sum_{\substack{m=0 \\ m \text{ odd}}}^{N} \binom{N}{m} a^{N-m} b^{m} = \frac{(a+b)^{N} - (a-b)^{N}}{2}$$

These are easily proven by taking the sum and difference of the respective binomial expansions of $(a+b)^{N}$ and $(a-b)^{N}$.

Then we find

$$\begin{aligned} Z_{110}^{+} &= \sum_{\substack{b=0 \\ b \text{ even}}}^{N} \left(\tfrac{1}{3}\cdot 2^{b} + \tfrac{2}{3}\right) \binom{N}{b} y^{N-b} z^{b} \\ &= \tfrac{1}{3} \sum_{b \text{ even}} \binom{N}{b} y^{N-b} (2z)^{b} + \tfrac{2}{3} \sum_{b \text{ even}} \binom{N}{b} y^{N-b} z^{b} \\ &= \tfrac{1}{6}\left[(y+2z)^{N} + (y-2z)^{N}\right] + \tfrac{1}{3}\left[(y+z)^{N} + (y-z)^{N}\right] \end{aligned}$$

and similarly

$$\begin{aligned} Z_{110}^{-} &= -\sum_{\substack{b=0 \\ b \text{ odd}}}^{N} \left(-\tfrac{1}{3}\cdot 2^{b} + \tfrac{2}{3}\right) \binom{N}{b} y^{N-b} z^{b} \\ &= \tfrac{1}{3} \sum_{b \text{ odd}} \binom{N}{b} y^{N-b} (2z)^{b} - \tfrac{2}{3} \sum_{b \text{ odd}} \binom{N}{b} y^{N-b} z^{b} \\ &= \tfrac{1}{6}\left[(y+2z)^{N} - (y-2z)^{N}\right] - \tfrac{1}{3}\left[(y+z)^{N} - (y-z)^{N}\right] \end{aligned}$$

Now, because we have

$$y = \frac{1}{3}\left(\exp\left(-\tfrac{3}{4}\varepsilon\beta J\right) + 2\exp\left(\tfrac{3}{4}\varepsilon\beta J\right)\right)$$

$$z = \frac{1}{3}\left(\exp\left(\tfrac{3}{4}\varepsilon\beta J\right) - \exp\left(-\tfrac{3}{4}\varepsilon\beta J\right)\right)$$

we compute

$$y - 2z = \exp\left(-\tfrac{3}{4}\varepsilon\beta J\right) \qquad y + 2z = -\tfrac{1}{3}\exp\left(-\tfrac{3}{4}\varepsilon\beta J\right) + \tfrac{4}{3}\exp\left(\tfrac{3}{4}\varepsilon\beta J\right)$$

$$y + z = \exp\left(\tfrac{3}{4}\varepsilon\beta J\right) \qquad y - z = \tfrac{2}{3}\exp\left(-\tfrac{3}{4}\varepsilon\beta J\right) + \tfrac{1}{3}\exp\left(\tfrac{3}{4}\varepsilon\beta J\right)$$

The expressions $(y-2z)^N = \exp(-\frac{3}{4}\beta J)$ and $(y+z)^N = \exp(\frac{3}{4}\beta J)$ fall out easily. The complementary expressions can be simplified in the continuum limit with the lemma

$$\lim_{N\to\infty}\left[a\exp(x/N)+(1-a)\exp(-x/N)\right]^N = \exp\left((2a-1)x\right)$$

Hence in the limit $N \to \infty$, we find

$$Z^+_{110} = \tfrac{1}{6}\left[\exp(\tfrac{5}{4}\beta J)+\exp(-\tfrac{3}{4}\beta J)\right]+\tfrac{1}{3}\left[\exp(\tfrac{3}{4}\beta J)+\exp(-\tfrac{1}{4}\beta J)\right]$$
$$Z^-_{110} = \tfrac{1}{6}\left[\exp(\tfrac{5}{4}\beta J)-\exp(-\tfrac{3}{4}\beta J)\right]-\tfrac{1}{3}\left[\exp(\tfrac{3}{4}\beta J)-\exp(-\tfrac{1}{4}\beta J)\right]$$

Recall from the arguments at the beginning of this section $Z^{\pm} = 2Z^{\pm}_{111} + 6Z^{\pm}_{110}$; with $Z^-_{111} = 0$ this gives

$$\begin{aligned} Z^+ &= 2Z^+_{111} + 6Z^+_{110} \\ &= \exp(\tfrac{5}{4}\beta J)+2\exp(\tfrac{3}{4}\beta J)+2\exp(-\tfrac{1}{4}\beta J)+3\exp(-\tfrac{3}{4}\beta J) \\ Z^- &= 6Z^-_{110} \\ &= \exp(\tfrac{5}{4}\beta J)-2\exp(\tfrac{3}{4}\beta J)+2\exp(-\tfrac{1}{4}\beta J)-\exp(-\tfrac{3}{4}\beta J) \end{aligned}$$

These results are in agreement with the result from the eigenvalue analysis of the 3 spin system:

$$Z = Z^+ - Z^- = 4\exp(\tfrac{3}{4}\beta J)+4\exp(-\tfrac{3}{4}\beta J)$$

We can also find the expectation of the "Sign" operator in the system where we choose to take the sign as an observable:

$$\langle \text{Sign} \rangle = \frac{Z^+ - Z^-}{Z^+ + Z^-} = \frac{4\exp(\tfrac{3}{4}\beta J)+4\exp(-\tfrac{3}{4}\beta J)}{2\exp(\tfrac{5}{4}\beta J)+4\exp(-\tfrac{1}{4}\beta J)+2\exp(-\tfrac{3}{4}\beta J)}$$

This function starts at unity for $\beta = 0$ and decreases monotonically to zero, asymptotically going to $2\exp(-\beta J/2)$. This accords with our expectation that $\langle \text{Sign} \rangle \propto \exp(-\beta V(f-f'))$ where $(f-f')$ is a difference in free energy densities.

"In our wildest aberrations
we dream of an equilibrium
we have left behind
and which we naïvely expect to find
at the end of our errors."
–Albert Camus

Appendix D. Thermalization Study

An ever-present consideration in Monte Carlo simulations of lattice spin systems is the startup transient. Typically one perforce specifies an initial configuration with a high degree of order. One relies on the algorithm to randomize the configuration in a finite number of steps, so that measurements subsequently can be taken over a large number of effectively "thermalized" configurations.

The loop cluster algorithm is a very efficient Monte Carlo technique, with very small autocorrelation time. Previous studies had used a thermalization period of 10,000 configurations followed by 100,000 measurements [Wiese92]. Based in part on the study documented here, we reduced this transient to 1,000 configurations, which we still believe to be more than sufficient to randomize the spin configuration with the continuous-time cluster algorithm.

We exhaustively examined four temperature-volume conditions in a matrix with β equal to 10 and 20, and N_s equal to 100 and 256. We let the simulation run for 100,000 configurations and recorded the following data for each step:

- Mused (number of transitions)
- Nseg (number of segments in path)
- Ene (internal energy)
- Sus (uniform susceptibility)
- Stagsus (staggered susceptibility)
- Csize (cluster size)

These parameters capture the trends of interest for both the observables and the key measures of operational status.

Plots were prepared of the binning distribution of these parameters, as well as current value versus configuration number for various bin sizes. The resulting plots, over 200 in number, were analyzed for signs of transient character prior to the establishment of equilibrium. Only a small fraction of these are presented here, since the great majority provide redundant information.

Additional statistical tests were performed to determine whether binning of the data had a significant influence on the error estimates.

The startup transient was most noticeable in the plots of Mused and Ene. It should be noted that Mused is expected to be a reasonable, albeit crude, measure of thermalization. The observed behavior of the AFHM simulations is that the number of transitions is randomly distributed about a mean that is proportional to the lattice volume βN_s, with a constant of proportionality that is $O(1)$.

The frequency distribution of Mused resembles a skewed Gaussian, and is shown in Figure D-1 for the condition β=20 and N_s=100. Tellingly, the time series for Mused shows a clearly detectable but short startup transient before reaching its mean value. This trend is shown in Figure D-2 for the same volume and temperature conditions and the first 1000 configurations.

The internal energy density Ene also shows a startup transient of width comparable to that of Mused. Figure D-3 shows a typical time series for -Ene for the same conditions as above.

Most of the other parameters showed far less distinct signatures on startup. The longest startup transient, measured as the number of configurations required for Mused to equal βN_s, was about 50 configurations, well within the 1,000-configuration guideline that we subsequently established. Figure D-4 shows a parametric plot of the typical initial joint evolution of Mused and -Ene. The trajectory rapidly evolves towards the thermalized region.

The statistical analysis of the effect of binning the data showed very little effect. A typical variation of error estimate with bin size is shown in Figure D-5. Here the mean and error estimate for uniform susceptibility is plotted versus bin size for $\beta = 10$, $N_s = 256$. Other parameters showed even less variation with bin size. (Note the tiny but detectable variation of the mean at large bin sizes is due to left-over points because powers of 2 beyond 32 do not divide evenly into 100,000).

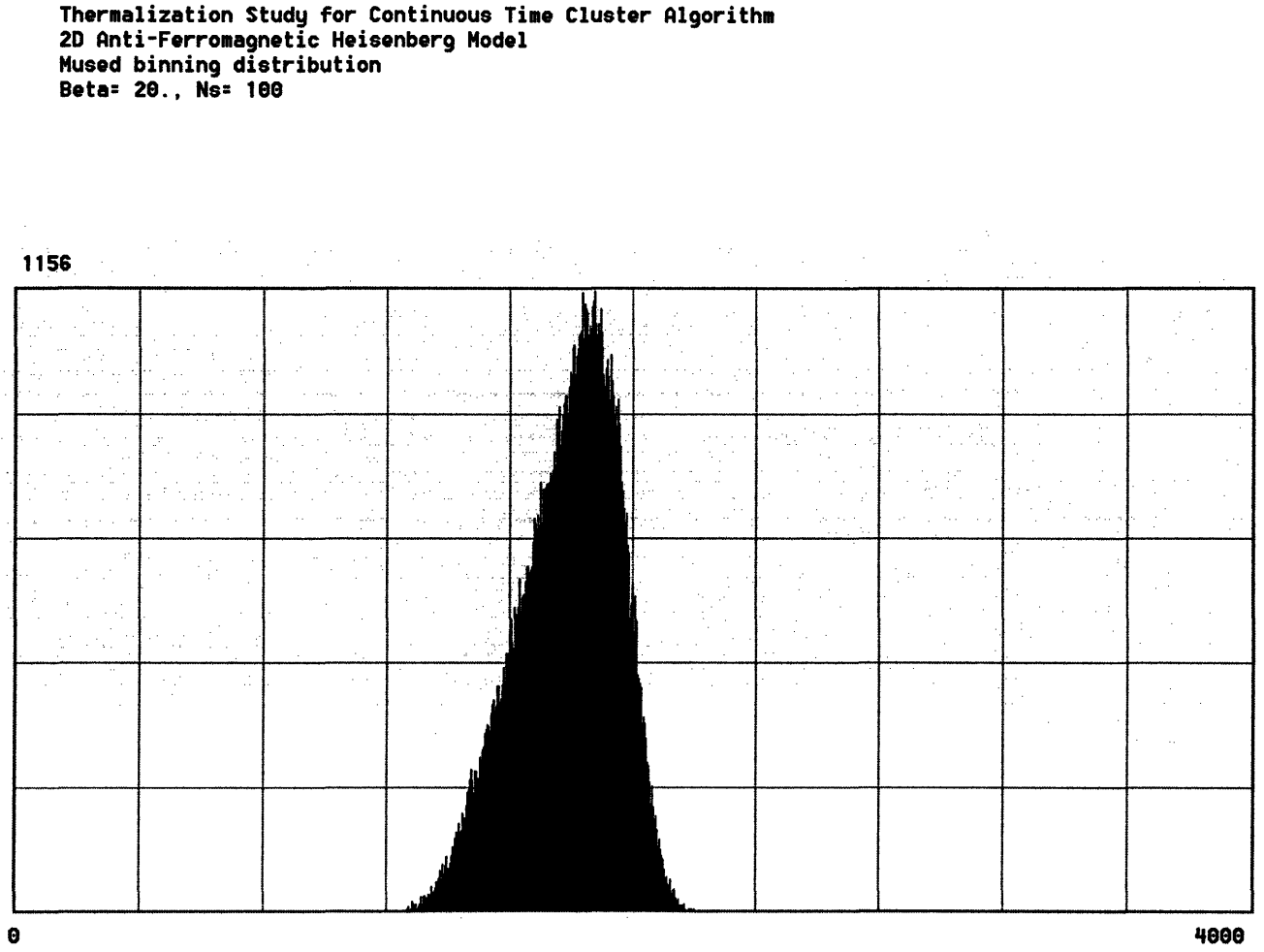


Figure D-1. Binning distribution for the measure Mused, which is the maximum number of transitions in the spin-time lattice, for the condition $\beta = 20$, $N_s = 100$.

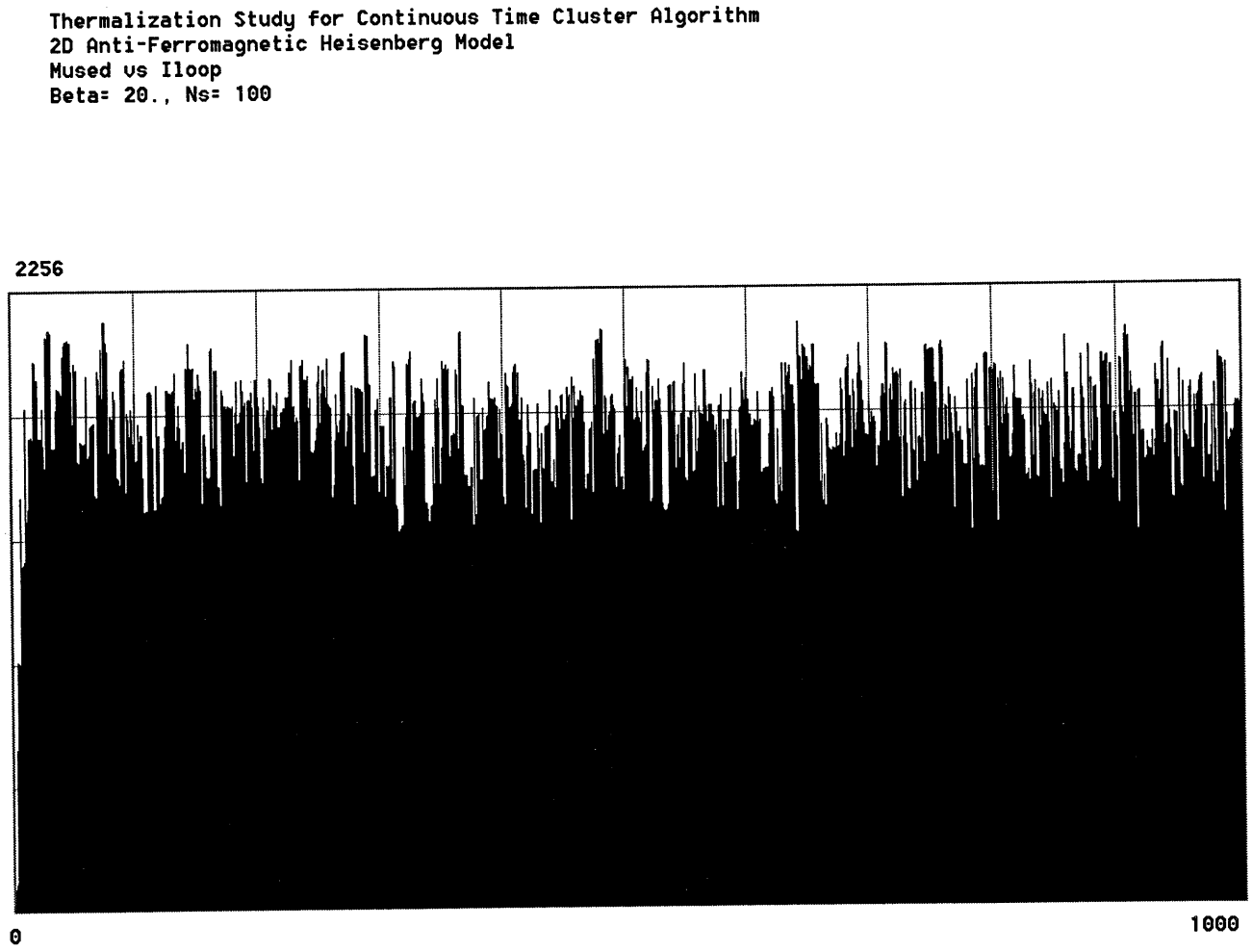


Figure D-2. Startup transient for the measure Mused, which is the maximum number of transitions in the spin-time lattice, for the condition $\beta = 20$, $N_s = 100$.

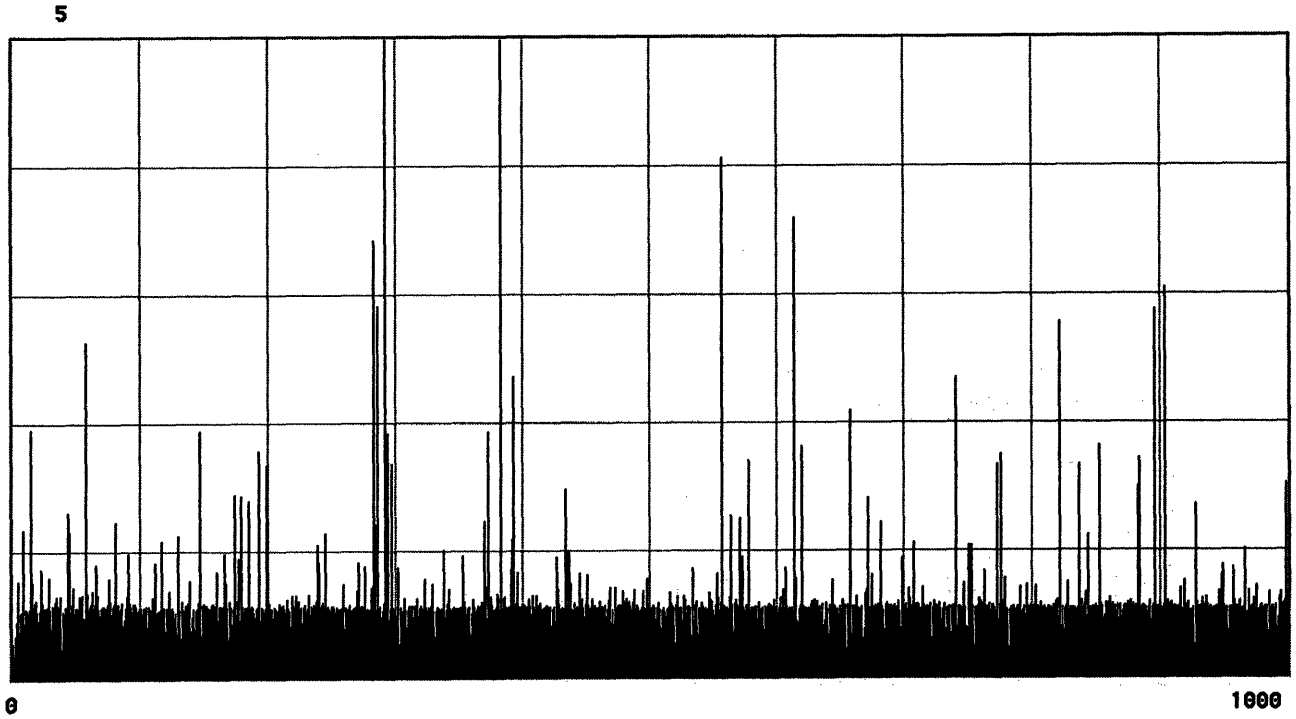


Figure D-3. Startup transient for the -Ene, the improved estimator for internal energy density, for the condition $\beta = 20$, $N_s = 100$. There is a brief but visible transient for the first 50 or so steps.

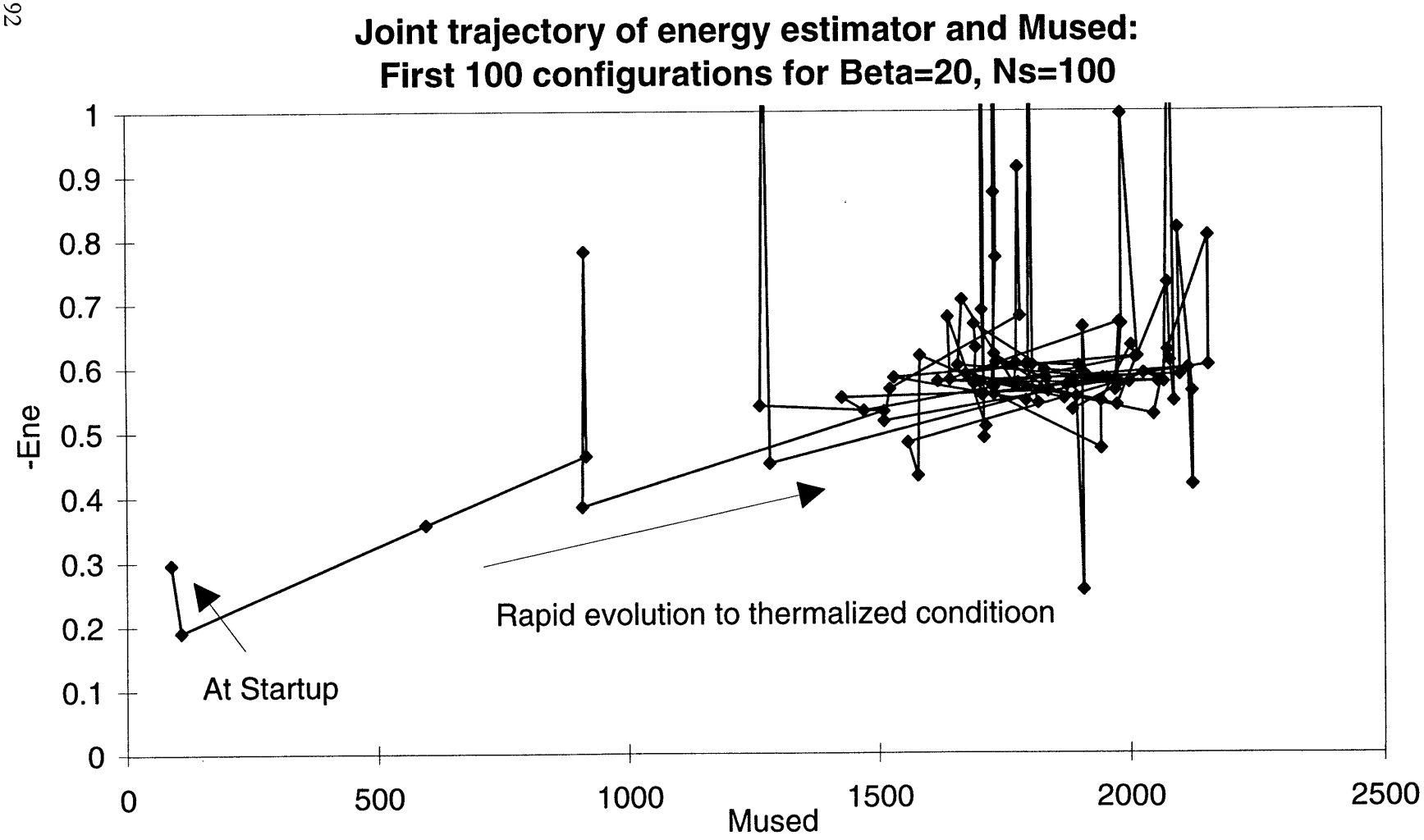


Figure D-4. Initial trajectory for -Ene versus Mused. Each point represents one configuration update. Evoltuion toward a thermalized condition appears to be quite rapid.

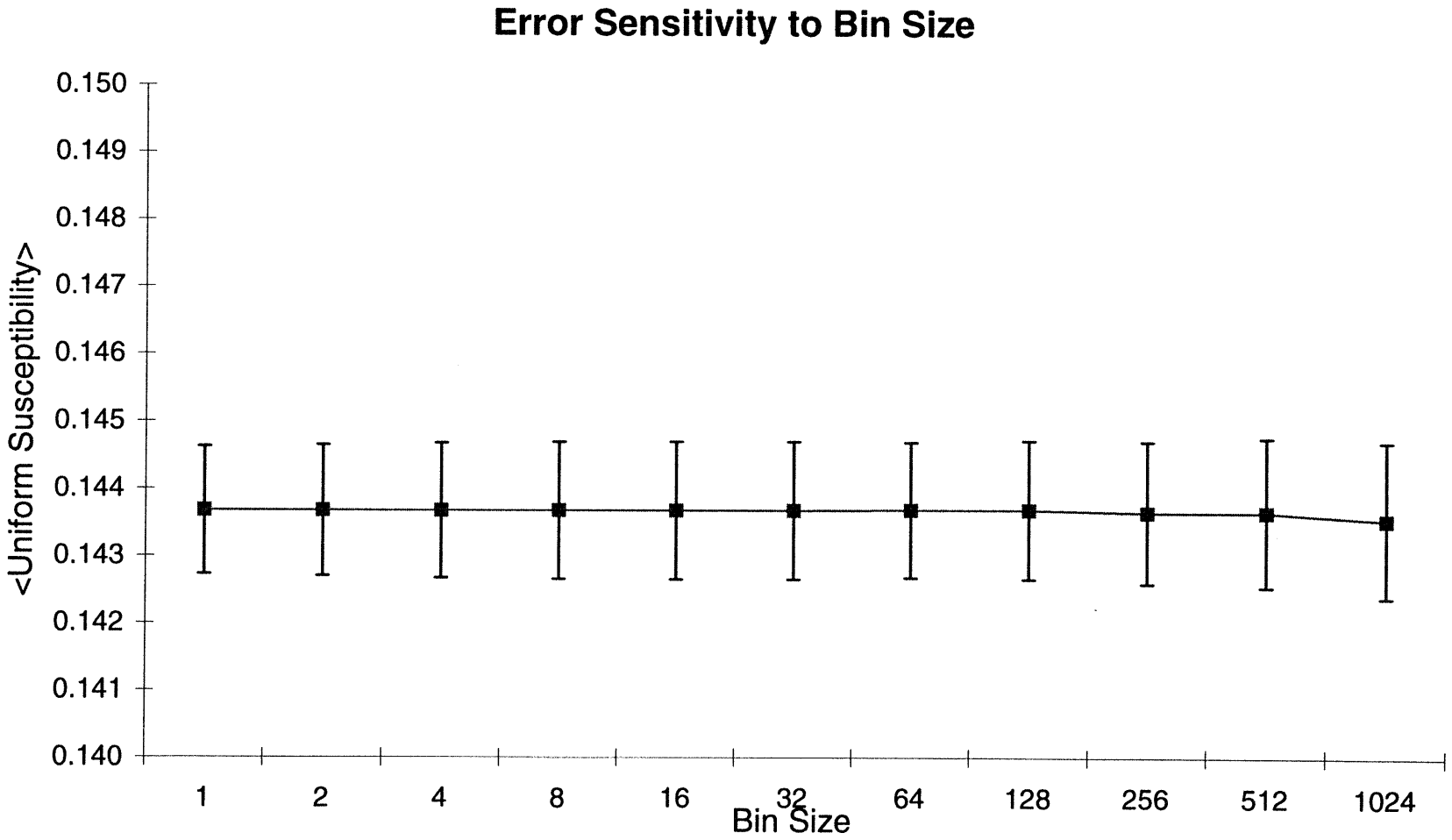
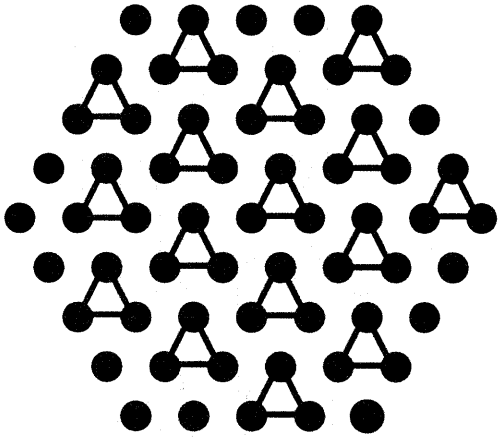


Figure D-5. Typical error dependencies on bin size. Here the error estimate for uniform susceptibility is examined for Beta=10, Ns=256. The error estimate is apparently insensitive to binning, indicative of the short autocorrelation time characteristic of the loop cluster algorithm.

"Where the statue stood
Of Newton with his prism and silent face
The marble index of a mind for ever
Voyaging through strange seas of thought, alone"
– William Wordsworth

Appendix E. Prismatic approach to the AFHM on the hexagonal lattice

Because of the shortcomings of the plaquette-based cluster algorithm, we explore a more promising concept – the grouping of the spin interaction into triads. The triads are then stacked in layers. Only three layers are needed to cover the hexagonal space, each layer resembling



Among other virtues this eliminates questions about the order of the layers, since the possible orderings are all related by symmetries of the lattice.

Because the triads represent a coupling in the time direction, the fundamental unit of the system is a prism. The three spin sites in the triad give rise to a basis space with 8 elements, and the transfer matrix for the system is an 8 by 8 matrix. We can write it in the block-diagonal form

$$M = \left(\begin{array}{c|ccc|ccc|c} x & & & & & & & \\ \hline & y & -z & -z & & & & \\ & -z & y & -z & & & & \\ & -z & -z & y & & & & \\ \hline & & & & y & -z & -z & \\ & & & & -z & y & -z & \\ & & & & -z & -z & y & \\ \hline & & & & & & & x \end{array}\right) \leftrightarrow \begin{pmatrix} 111 \\ 110 \\ 101 \\ 011 \\ 100 \\ 010 \\ 001 \\ 000 \end{pmatrix}$$

where as usual a 1 denotes a spin up site and a 0 denotes a spin down site. Note that the states $|100\rangle$ and $|011\rangle$ are out of order from the usual "binary-$|n\rangle$" basis, in order to show the block-diagonal form of the transfer matrix. The blocks themselves are associated with the conserved $m_s = \frac{3}{2}, \frac{1}{2}, -\frac{1}{2}, -\frac{3}{2}$. The quantities x, y, and z are all positive numbers for the antiferromagnetic ($J > 0$) coupling. We plan to absorb the minus signs in the off-diagonal elements into the estimator for sign. We compute

$$x = \exp(-\tfrac{3}{4}\varepsilon\beta J)$$

$$y = \frac{1}{3}\left(\exp(-\tfrac{3}{4}\varepsilon\beta J) + 2\exp(\tfrac{3}{4}\varepsilon\beta J)\right)$$

$$z = \frac{1}{3}\left(\exp(\tfrac{3}{4}\varepsilon\beta J) - \exp(-\tfrac{3}{4}\varepsilon\beta J)\right)$$

Note that as $\varepsilon \to 0$ we will have $x, y \to 1$ and $z \to 0$, and that for $J > 0$ we have

$$y - x = \frac{1}{3}\left(\exp(-\tfrac{3}{4}\varepsilon\beta J) + 2\exp(\tfrac{3}{4}\varepsilon\beta J)\right) - \exp(-\tfrac{3}{4}\varepsilon\beta J) = \frac{4}{3}\sinh(\tfrac{3}{4}\varepsilon\beta J) > 0$$

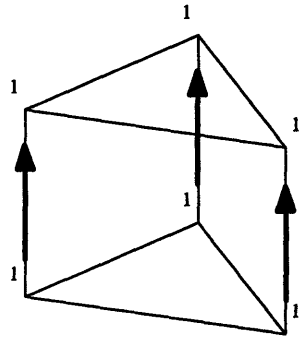
and

$$z = \frac{2}{3}\sinh(\tfrac{3}{4}\varepsilon\beta J) = \frac{y - x}{2} > 0$$

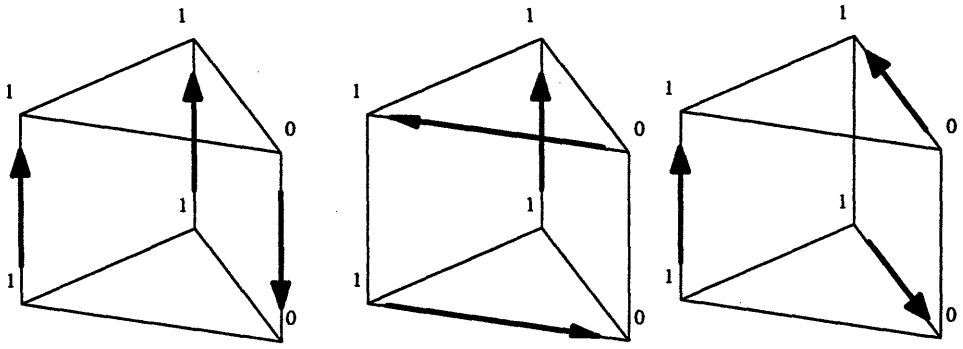
There are thus three categories of plaquettes. It remains to be shown that we can define flows and transition probabilities that are consistent with the Boltzmann factors in the transfer matrix.

We start by examining the plaquettes that, like the plaquettes for the 1-D chain and 2-D square lattices, have no "diagonal" type flows (like those that arise in the ferromagnetic coupling).

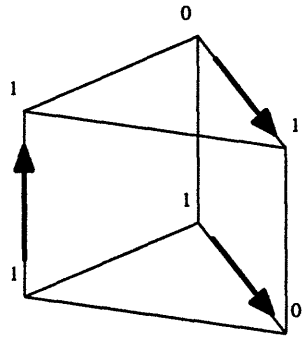
Category "x" represents the 2 matrix elements $\langle 111|M|111\rangle$ and $\langle 000|M|000\rangle$. We propose the flow



Category "y" represents the 6 diagonal elements $\langle 110|M|110\rangle$, etc. We propose the flows:



Category "z" represents the 12 off-diagonal elements $\langle 101|M|110\rangle$, etc. We propose the flows:



and analogously for the other off-diagonal prisms.

The flows in the diagrams above are so far constrained only by the requirements

- ...that flipping of a single flow must give a legal (finite action) configuration. Because the total spin is constant in time, flow between two sites in the same time layer cannot involve "same states", and flow between two sites in different time layers must involve "same states".
- ...that the flow in a "1" state cannot flow backward in time and the flow in a "0" state cannot flow forward in time. This means that the only entry points to the prism are at "1" sites in the lower layer and "0" sites in the upper layer. The reverse applies to exit points.
- ...flows are not diagonal.
- ...flows can't collide.

We can verify that detailed balance can be satisfied with this set of flows.

Consider the transitions from "x" to "y". We wish to have for any states $p(1)w(1\rightarrow 2) = p(2)w(2\rightarrow 1)$ where p is a Boltzmann weight and w is a transition probability. In this case

$$p(1) = p(x) = \exp(-\tfrac{3}{4}\varepsilon\beta J)$$
$$p(2) = p(y) = \frac{1}{3}\left(\exp(-\tfrac{3}{4}\varepsilon\beta J) + 2\exp(\tfrac{3}{4}\varepsilon\beta J)\right)$$
$$w(1 \to 2) = 1$$

If we let the probability of choosing the first flow configuration listed under "y" above (the "continuation" or "strong" configuration) be $w_1 = \exp(-\tfrac{3}{4}\varepsilon\beta J)\Big/\frac{1}{3}\left(\exp(-\tfrac{3}{4}\varepsilon\beta J) + 2\exp(\tfrac{3}{4}\varepsilon\beta J)\right)$, then we will satisfy detailed balance. This works out to

$$w_1 = 3/\left(1 + 2\exp(\tfrac{3}{2}\varepsilon\beta J)\right) \le 1$$

which can indeed be interpreted as a probability. The probabilities of the remaining two "y" flow configurations ("decay" or "weak" configurations) must be equal and must add with w_1 to get unity. Hence they each have probability

$$(1 - w_1)/2 = \left(\exp(\tfrac{3}{2}\varepsilon\beta J) - 1\right)/\left(1 + 2\exp(\tfrac{3}{2}\varepsilon\beta J)\right)$$

We proceed to verify that detailed balance can be satisfied in the "z" sector. Consider the transitions from the weak "y" configurations to the "z" configuration, for example if we flip one of the horizontal path segments in one of the weak flow configurations. We wish to determine the selection probability of the "z" configuration, which we will call w_2. Here we have

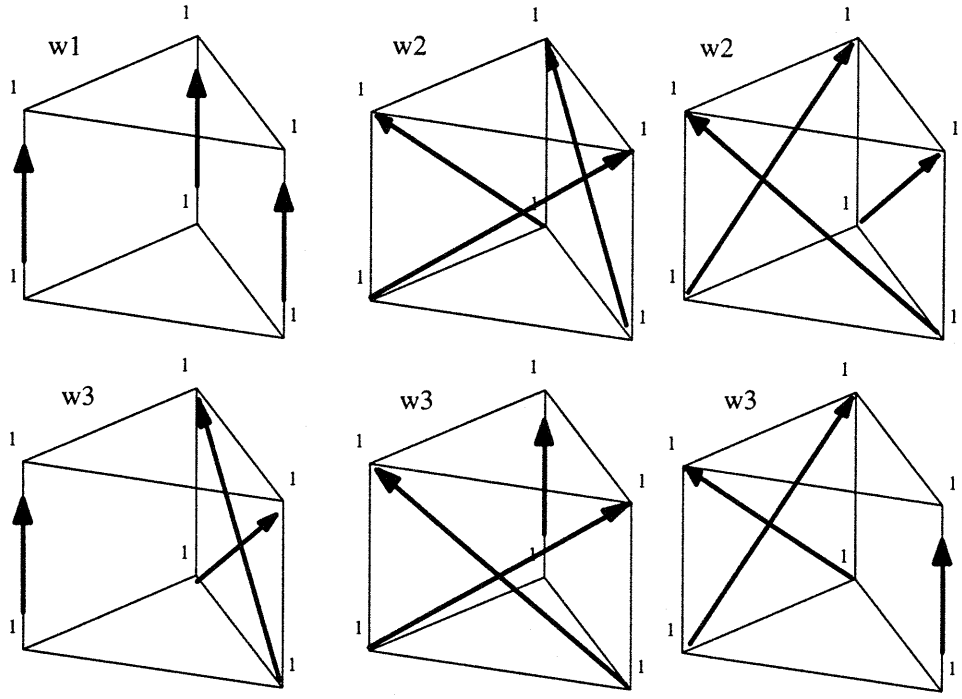
$$p(1) = p(y) = \frac{1}{3}\left(\exp(-\tfrac{3}{4}\varepsilon\beta J) + 2\exp(\tfrac{3}{4}\varepsilon\beta J)\right)$$
$$p(2) = p(z) = \frac{1}{3}\left(\exp(\tfrac{3}{4}\varepsilon\beta J) - \exp(-\tfrac{3}{4}\varepsilon\beta J)\right)$$
$$w(1 \to 2) = (1 - w_1)/2 = \left(\exp(\tfrac{3}{2}\varepsilon\beta J) - 1\right)/\left(1 + 2\exp(\tfrac{3}{2}\varepsilon\beta J)\right)$$
$$w(2 \to 1) = w_2$$

Here we see that detailed balance can be satisfied if and only if we pick $w_2 = 1$.

Unfortunately, the loop-building strategy that is embodied in the above diagrams is not ergodic. Any path built with these rules changes direction every time it moves from one site to another, so by the time it returns to the initial site it will have changed direction an even number of times. Hence the "even" and "odd" sectors are segregated -- the total number bonds in the lattice configuration remains even or odd when such a path is flipped. Unlike the 1-D chain and 2-D square lattices, the hexagonal lattice can support configurations with an odd number of bonds.

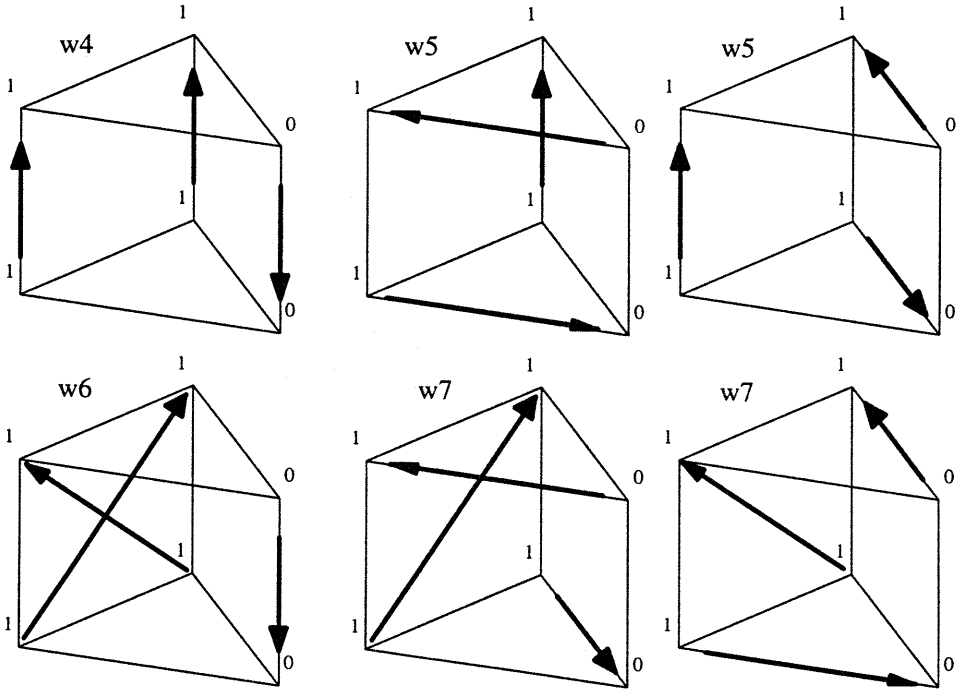
In an effort to determine whether a larger strategy could be ergodic, we allow the plaquettes to have diagonally crossing flows. With three entry points and three exit points and no collisions allowed, each sector will have six flow patterns.

The "x" sector then has the 6 possible flows



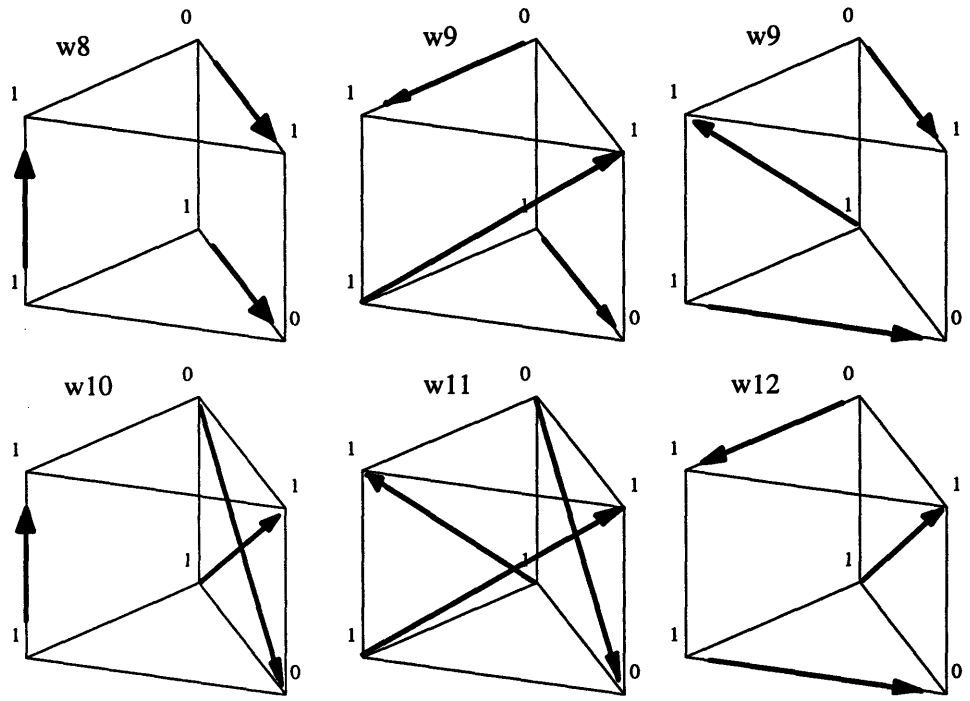
Here we have assigned probabilities based on the symmetry class of the flow pattern. We must enforce the constraint $w_1 + 2w_2 + 3w_3 = 1$.

The "y" sector has six possible flows:



Here we have $w_4 + 2w_5 + w_6 + 2w_7 = 1$.

The six "z" sector flows are:



Here $w_8 + 2w_9 + w_{10} + w_{11} + w_{12} = 1$.

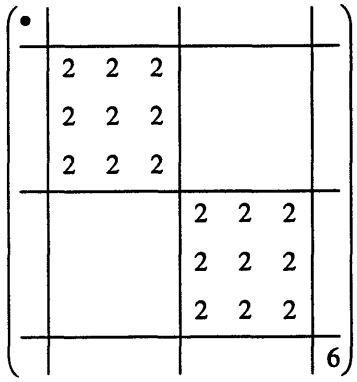
There are several way of approaching the daunting task of formulating the constraints of detailed balance in this more general case. Note to begin with we have 12 unassigned probabilities and 3 sum-to-unity constraints.

There are 20 distinct prisms, corresponding to the 20 non-zero elements of the transfer matrix. Each prism has 6 different flow types, and each prism-flow-type configuration can be visited by singly, doubly, or triply by the cluster path, for a total of 7 different cases (3 single, 3 double, and 1 triple). There is thus a grand total of $20 \times 6 \times 7 = 840$ mappings from one prism to another, if we count all the possible channels from one prism to another. Of course, with 20 possible prisms, we can represent the conditions of detailed balance by a 400-element matrix:

x^+x^+	x^+y^+	x^+z^+	x^+z^-	x^+y^-	x^+x^-
	y^+y^+	y^+z^+	y^+z^-	y^+y^-	y^+x^-
		z^+z^+	z^+z^-	z^+y^-	z^+x^-
			z^-z^-	z^-y^-	z^-x^-
				y^-y^-	y^-x^-
					x^-x^-

Here we have labeled the regions according to the initial and final Boltzmann categories as well as the sign of the z-component of spin, which ranges over $\{+\frac{3}{2},+\frac{1}{2},-\frac{1}{2},-\frac{3}{2}\}$. Into each slot of this array we will insert a Boltzmann factor times a transition probability $p(1)w(1\to 2)$. The condition of detailed balance is that this matrix must be symmetric, so we will call this 20×20 matrix the "balance matrix", to avoid confusion with the transfer matrix, which is 8×8. We see that the 840 mappings get binned into the 400 slots of the balance matrix. Note that none of the mappings wind up in the diagonal entries, so these slots will be empty. There are far from $20\times 19/2 = 190$ independent constraints here, of course, owing to the high degree of symmetry. In particular we have both a triangular symmetry consisting of rotations and reflections, as well as a spin inversion symmetry. We will count the number of independent constraints momentarily.

Another way to visualize the detailed balance condition is to consider the transitions of a given prism. Each prism has 6 flow types and 7 visit types, so there are 42 transitions that it can undergo, many leading to the same final state. Consider, for example, the prism with all spins up. We can refer to the 6-flow diagram above and manually count the number of transitions to each of the 19 other prisms. Arranging the count to overlay the transfer matrix, we find



The total is $18\times 2+6=42$, as we expected. Note that the transitions x^+y^+ and x^+z^+ require a single flow visit, the transitions x^+y^- and x^+z^- require a double flow visit, and x^+x^- requires a triple visit. In this case, because of the symmetry of the initial prism, we note that the weighted rates are identical for all the transitions for a given final class. For example, we find the weighted rate for x^+y^+ to be

$$W(x^+y^+)=p\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}w\left(\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\to\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix}\right)=p\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}w\left(\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\to\begin{pmatrix}1&0&1\\1&0&1\end{pmatrix}\right)=p\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}w\left(\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\to\begin{pmatrix}0&1&1\\0&1&1\end{pmatrix}\right)=x(w_1+w_3)$$

This is to be expected from the triangular symmetry of our problem. We also find

$$W(x^+z^+)=p\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}w\left(\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\to\begin{pmatrix}1&1&0\\1&0&1\end{pmatrix}\text{ etc.}\right)=x(w_2+w_3)$$

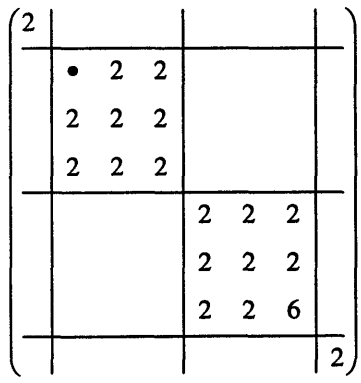
$$W(x^+y^-)=p\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}w\left(\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\to\begin{pmatrix}1&0&0\\1&0&0\end{pmatrix}\text{ etc.}\right)=x(w_1+w_3)$$

$$W(x^+z^-)=p\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}w\left(\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\to\begin{pmatrix}0&1&0\\1&0&0\end{pmatrix}\text{ etc.}\right)=x(w_2+w_3)$$

$$W(x^+x^-)=p\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}w\left(\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\to\begin{pmatrix}0&0&0\\0&0&0\end{pmatrix}\right)=x(w_1+2w_2+3w_3)=x$$

The last equation is a complete inversion, for which detailed balance will automatically be satisfied because of the spin inversion symmetry.

We repeat this exercise starting with a y^+ element. We find the count array is



Note first that transitions within a class can occur but that the condition of detailed balance will automatically be satisfied because triangular symmetry guarantees that the same expression will show up on both sides of the equation. On the other hand, the same weighted rate is not guaranteed to occur for all elements of a final class, because the initial prism is not completely symmetric.

We find

$$W(y^+x^+) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\right) = y(w_4 + w_6) = W(y^+x^-)$$

which we will equate to the weighted rate $W(x^+y^+) = W(x^+y^-)$ later. Also

$$W(y^+y^-) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}1&0&0\\1&0&0\end{pmatrix}\right) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}0&1&0\\0&1&0\end{pmatrix}\right) = y(w_4 + w_5)$$

noting the difference with the complete inversion

$$p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}0&0&1\\0&0&1\end{pmatrix}\right) = y(w_4 + 2w_5 + w_6 + 2w_7) = y$$

for which again detailed balance will be satisfied automatically by spin inversion symmetry.

Trickier to deal with is

$$\begin{aligned} W(y^+z^+;A) &= p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}\right) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}1&1&0\\1&0&1\end{pmatrix}\right) \\ &= p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}1&1&0\\0&1&1\end{pmatrix}\right) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}0&1&1\\1&1&0\end{pmatrix}\right) = y(w_5 + w_7) \\ W(y^+z^+;O) &= p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}0&1&1\\1&0&1\end{pmatrix}\right) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}1&0&1\\0&1&1\end{pmatrix}\right) = y(w_6 + w_7) \end{aligned}$$

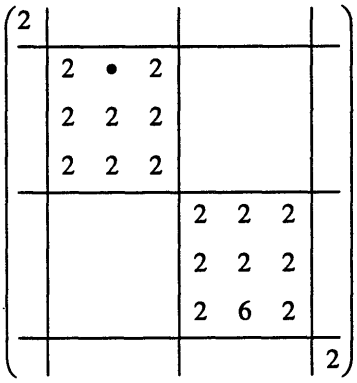
(The labels A and O indicate adjacent and opposite edges, referring to the location of the flipped vertices relative to the unique edge $\begin{pmatrix}0\\0\end{pmatrix}$.)

We also find

$$\begin{aligned} W(y^+z^-;A) &= p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}0&0&1\\0&1&0\end{pmatrix}\right) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}0&1&0\\0&0&1\end{pmatrix}\right) \\ &= p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}1&0&0\\0&0&1\end{pmatrix}\right) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}0&0&1\\1&0&0\end{pmatrix}\right) = y(w_5 + w_7) \\ W(y^+z^-;O) &= p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}0&1&0\\1&0&0\end{pmatrix}\right) = p\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} w\left(\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix} \to \begin{pmatrix}1&0&0\\0&1&0\end{pmatrix}\right) = y(w_6 + w_7) \end{aligned}$$

It remains to analyze the case with initial prism z^+.

The count array is



We find

$$W(z^+x^+) = p\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}w\left(\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}\to\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}\right) = z(w_{10}+w_{11}) = W(z^+x^-)$$

and

$$W(z^+y^+;A) = p\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}w\left(\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}\to\begin{pmatrix}1&1&0\\1&1&0\end{pmatrix}\right) = p\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}w\left(\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}\to\begin{pmatrix}1&0&1\\1&0&1\end{pmatrix}\right) = z(w_8+w_9) = W(z^+y^-;A)$$

$$W(z^+y^+;O) = p\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}w\left(\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}\to\begin{pmatrix}0&1&1\\0&1&1\end{pmatrix}\right) = z(w_{10}+w_{12}) = W(z^+y^-;O)$$

Once again the z^+z^+ and z^-z^- sectors are covered by detailed balance.

The remaining sectors are related to the cases above by spin inversion. We find that we have 4 independent conditions

$$W(x^+y^+) = W(y^+x^+) \Leftrightarrow x(w_1+w_3) = y(w_4+w_6)$$

$$W(x^+z^+) = W(z^+x^+) \Leftrightarrow x(w_2+w_3) = z(w_{10}+w_{11})$$

$$W(y^+z^+;A) = W(z^+y^+;A) \Leftrightarrow y(w_5+w_7) = z(w_8+w_9)$$

$$W(y^+z^+;O) = W(z^+y^+;O) \Leftrightarrow y(w_6+w_7) = z(w_{10}+w_{12})$$

which, together with our 3 sum-to-unity constraints, makes 7 constraint equations on 12 variables. We apparently have the freedom to tune the probabilities to any values in a 5-parameter space.

"The first rule of intelligent tinkering is to save all the parts."
– Paul Ehrlich

Appendix F. Compression of lattice data

Although much of the material in this appendix was made moot by the development of the continuous-time cluster algorithm, there may still be some insights worthy of note. The original intention of this effort was to explore ways that the highly redundant spin state data in a spin-time lattice could be stored and accessed more efficiently.

Bits and Bytes

Mapping a set of spins to the internal organization of a computer is accomplished by identifying up or down spins with individual bits; that is, $|\uparrow\rangle \Leftrightarrow 1, \quad |\downarrow\rangle \Leftrightarrow 0$

The time dependence of the spin associated with any particular lattice site is then represented by a sequence of bits `1110001010111000`.... Arbitrarily long sequences can be represented by arrays of bytes. For a complete lattice we would have a multidimensional array.

The hope we have for compressing the information in this bit stream is precisely that the possible states are quantized (unlike the position variable in a ordinary path integral), so it is possible that, for example, by merely recording the points of transition from one state to the other, we can effect savings in storage.

Now it happens that a "random" sequence of bits is not particularly compressible, at least not with lossless compression algorithms. In our spin language, this is equivalent to observing that if the state changes every few time slices, recording the times of transition will actually take up more array space than the original spin sequence.

However, if the probability of transition is small for every time slice, then we expect long sequences of spins which are the same, with sporadic transitions. We expect such patterns to be highly compressible. This is related to the observation that the largest summands in the partition function are those with many $(1-p) \approx 1$ terms and relatively few $p << 1$ terms.

Comment: It is unclear whether it is best to pursue a strategy of maximal compression, as might be achieved with Huffman or Lempel-Ziv compression [see Nelson92], or whether to pursue a more simple-minded but "practical" algorithm. The former is optimal in the sense that the required information is encoded in the least number of bytes, but there is less than a faint hope that observables can be sampled without fully packing and unpacking the bit stream. On the other hand, if we unpack the compressed stream "site-by-site" we still effect a real savings in RAM requirements. Regardless, there is still some extra computational overhead associated with maximal compression. The potential advantage of vanilla compression is that there is hope that observables can be accessed directly with the compressed byte streams, and also that the compressed streams can be generated and manipulated directly.

"Scheme A"

One straightforward technique for encoding sequences of bits is the following algorithm, which we will call "Scheme A". This scheme is targeted at byte streams with relatively few transitions. Note that the algorithm described is intended for machines that are set up to deal with 8 bit bytes, which is practically a universal standard. It is sometimes convenient to use hexadecimal notation to describe our bit sequences (recall that the decimal conversion is A=10, B=11, C=12, D=13, E=14, F=15 in hex). A single byte can be thought of as a sequence of eight bits or two hexes. Thus 00101100 = 4C hex, also sometimes denoted 0x4C.

For a sequence of bits which we expect to comprise long stretches of identical bits, we compress the stream as follows:

We first imagine that the stream `1111111...1110000...0001111...` is divided up into 8-bit bytes. Most of the bytes will be either all zeroes (`00000000` = `00` hex) or all ones (`11111111` = `FF` hex). Of the remaining bytes, most will have a single "transition" such as `00011111` = 1F. There are 7 patterns that start with a zero and end with a one, and 7 that start with a one and end with a zero:

`{0...1} = {01, 03, 07, 0F, 1F, 3F, 7F}` `{1...0} = {FE, FC, F8, F0, E0, C0, 80}`

All 256 bit patterns can be so categorized by the number of transitions (the frequencies are simply the binomial coefficients ${}_7C_n$). In addition, a transition can occur at a byte boundary, in which case we will credit it to the preceding byte.

The scheme A compression algorithm is this: we bookkeep two arrays of numbers: the first records the length of each sequence of identical bytes, and the second records the transition byte. For example, the sequence of bytes (here 256 bytes long)

```
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF 80 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 1F FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF F8 00 00 00 00 00
```

is encoded

# of bytes (in decimal):	57	107	84	5 + ...
transition byte (in hex):	80	1F	F8	...

so this sequence bytes can be encoded in as few as 7 bytes. Actually, we need at least one or a few mor es to record the length of the compressed stream, but such indexing storage is negligible and will not be counted in the development below.

We choose as a convention that the odd-indexed positions (1,3,5,7...) are devoted to FF strings and the even-indexed positions are devoted to 00. If this proves cumbersome, we can explore another scheme wherein the identity of the running byte (00 or FF) is recorded in yet another array... but first things first.

Note to save storage space we encode the length information in single bytes, which in turn requires that strings of bytes longer than 255 be encoded in multiple positions. If the length of the uncompressed byte stream is large enough, it may pay to switch to multiple-byte array elements for the upper ("length") array. If the length of any sequence of identical bits is enough to require an overflow into the next array element, we call this a "storage fault".

Although scheme A does not provide maximal compression, it has the virtue that information about the order of bits is strictly preserved, and unpacking is potentially unnecessary. Also, all possible bit patterns can be encoded with this scheme (since we will allow entries to be zero), albeit with the disadvantage that patterns with frequent transitions (i.e. with no occurrences of 00 or FF) can actually expand to twice the initial storage space. (Of course it is possible to devote one more bit of storage to "turn off" compression when the transition frequency reaches some threshold. More about this below.)

If further packing is required, the sequence of transition bytes can be easily compressed, e.g. with Huffman coding, since the relative frequencies are known.

Estimating the compression achievable with Scheme A

Let us explore the compression ratio for different bit patterns. Assume that the uncompressed sequence of spin information takes up N bytes of storage space – i.e. that $8N$ spins are included (note that this differs from the definition used in the previous section). We assume that the sequence of bits begins with 1, and that the periodicity condition is satisfied by requiring that the (virtual) $8N+1$ bit is also 1. The configuration space has 2^{8N-1} elements. However, as we noted above, the partition function is dominated by sequences with few transitions.

For the (unique) sequence with all spins up ($m=0$) we have the sequence FF FF ... FF. If $N<256$, this can be encoded in a single byte. If $N \geq 256$, we allow the "transition byte" itself to be FF, so that the sequence is encoded

# of bytes (in decimal):	255	0	255	...
transition byte (in hex):	FF	FF	FF	...

(recall our convention that every other slot is devoted to 00 byte runs). The number of bytes required for the compressed stream is $\cong 4N/256$ for $N > 256$.

If N is sufficiently large, it pays to go to a two-byte encoding scheme for the upper array, in which strings up to $2^{16}-1=65535$ bytes long can be encoded; then the compressed length for $m=0$ is $\cong 6N/65536$ for $N > 65536$.

We see that storage faults can exact a significant penalty if N is large enough. On the other hand, storage faults mainly occur when the average run of untransitioned bytes ($\approx N/(m+1)$) is on the order of 256 or greater. In other words, once $m > (N/256-1)$ the storage fault frequency drops off. We see that the decision to implement double byte storage for the upper array depends on the typical number of transitions (m) to be processed.

For sequences with only two transitions ($m=2$), we typically have

$$\underbrace{\texttt{FF FF ... FF}}_{n_1}\ \underbrace{\texttt{(tb)}}_{1}\ \underbrace{\texttt{00 00 ... 00}}_{n_2}\ \underbrace{\texttt{(tb)}}_{1}\ \underbrace{\texttt{FF FF ... FF}}_{n_3}$$

where (tb) indicates one of the transition bytes. If $N < 256$, this compresses to

# of bytes (in decimal):	n_1	n_2	n_3
transition byte (in hex):	(tb)	(tb)	

or five bytes. (We could eliminate one of these bytes by using the constraint $n_1+n_2+n_3+2=N$ ... but this is equivalent and offsetting to the indexing storage already neglected). If $N \geq 256$, potentially extra bytes would be required to encode the additional length information, or we could be motivated to go to two-byte storage for the upper array. On rare occasion, both transitions will fall in the same transition byte, as for example in the string `11000111` = `0xc7`. The complete sequence would resemble

$$\underbrace{\texttt{FF FF ... FF}}_{n_1}\ \underbrace{\texttt{C7}}_{1}\ \underbrace{\texttt{FF FF ... FF}}_{n_2}$$

It is a matter of convention how to encode these rare cases, but one way is to code zero length for the number of 00 bytes and pull the next byte into the transition byte slot.

# of bytes (in decimal):	n_1	0	n_2-1
transition byte (in hex):	C7	FF	

In general, for "sparse" transitions ($m << N$), we see that we need $2m+1$ bytes of storage for moderate N, and additional storage if any of the n_i exceeds 255. The average length of the unbroken sequences is $N/(m+1)$. If we choose two-byte storage for the upper array, the number of bytes required jumps to $3m+2$ but fewer storage faults are incurred.

As the number of transitions grows denser, the length of the compressed byte stream increases and is only partially offset by the occurrence of multiple transitions in single bytes. A detailed development is presented below.

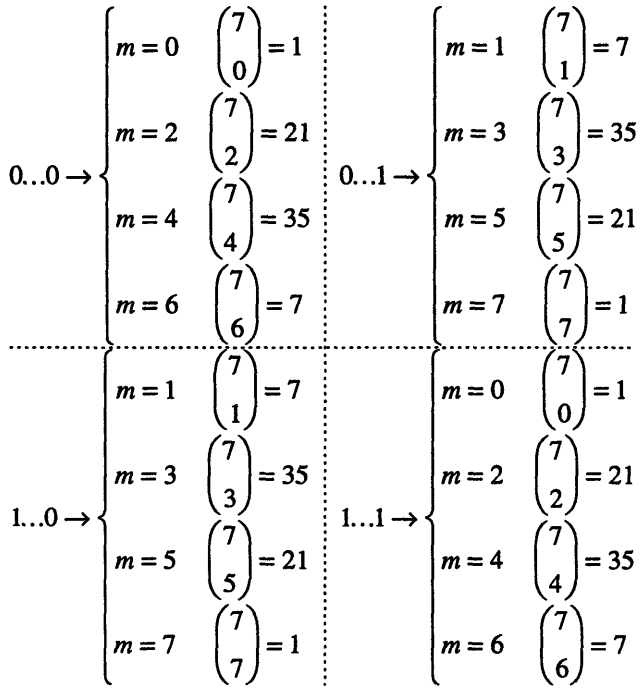
Note that even if we have a high probability of transition from one bit (spin state at a time slice) to the next, it is possible to exploit the correlation of bits that are adjacent in time. A transformation that flips every other spin in the time sequence can be used to transform to the previous case (details not worked out). However, because we will always be interested in the case $\varepsilon \to 0$, this is not an important detail.

Comment: A promising strategy for reaping further savings in compression is to employ "lossy" compression algorithms; i.e. codecs that approximate the bit pattern, rather than preserving all the information in the bit stream. It certainly seems likely, given the statistical nature of the Monte Carlo method, that further loss of information can be more than compensated by the increased sampling made possible by compression. Note that lossy algorithms have been studied in great detail for the purposes of encoding pictorial information; huge compression ratios are possible in practice. I have not yet explored this possibility.

Distribution of Transitions.

We consider the general case of an N-byte sequence (8N bits). We assume that the sequence of bits begins with 1, and that the periodicity condition is satisfied by requiring that the (virtual) $8N+1$ bit is also 1. The configuration space has 2^{8N-1} elements. There is exactly one sequence with $m=0$ transitions, namely `FF FF ... FF`.

First we must clarify a subtlety in the counting of transitions. Within each byte, there are eight bits and thus at most seven transitions. As we mentioned in the main paper, the relative frequencies of m-transition bytes are given by the binomial coefficients:



However, it is also possible for a transition to occur at a byte boundary, for example, in the stream `FF FF 00 ....`

In such cases we will credit the transition to the preceding byte, so that each byte can have up to 8 transitions – seven "intrinsic" and one "extrinsic". Thus in scheme A the sequence

`FF FF FF FF FF 00 00 00 00 00 00 FF FF FF ...`

is encoded

# of bytes (in decimal):	4	5	3+ ...
transition byte (in hex):	FF	00	...

Consider the case $m = 2$. We have two ways of counting the number of sequences with exactly two transitions. On the one hand, we can treat the sequence as a string of bits, after each of which we can place a transition. There are obviously

$$\binom{8N}{2} = \frac{(8N)!}{2!(8N-2)!} = 32N^2 - 4N$$

sequences with two transitions. On the other hand, we can add together the number of sequences in which the two transitions occur in distinct bytes, plus the number of sequences in which both transitions occur in the same byte. The former number is

$$\binom{8}{1}\binom{8}{1}\binom{N}{2} = 32N^2 - 32N$$

and the latter is $\binom{8}{2}\binom{N}{1}=28N$. We see that indeed

$$(32N^2-32N)+(28N)=32N^2-4N$$

as we expect.

For general m, bitwise counting tells us that there are $\binom{8N}{m}$ sequences. For $m \le N$ it is possible to enumerate the sequences in which no more than one transition per byte is encountered. There are $\binom{8}{1}^m\binom{N}{m}$ such sequences. Proceeding to enumerate the sequences with many bytes with a single transition and one or more bytes with exactly two transitions, we see

$$\underbrace{(1,1,1\ldots1)}_{m} \rightarrow \binom{8}{1}^m\binom{N}{m}$$

$$\underbrace{(1,1,1\ldots1}_{m-2}\underbrace{,2)}_{1} \rightarrow \binom{8}{1}^{m-2}\binom{8}{2}\left(\frac{N!}{(m-2)!1!(N-m+1)!}\right)$$

$$\vdots$$

$$\underbrace{(1,1,1\ldots1}_{m-2j}\underbrace{,2,\ldots2)}_{j} \rightarrow \binom{8}{1}^{m-2j}\binom{8}{2}^{j}\left(\frac{N!}{(m-2j)!\,j!(N-m+j)!}\right)$$

At this point it is clear that the distribution of transitions is intimately connected with the partition function of number theory (that is, the number of ways of splitting a positive integer into summands). Since each byte can contain at most 8 transitions, the number of possible byte patterns is given by $p_8(m)$, which is the number of ways of splitting the number m into summands less than or equal to 8. Let us write the general pattern as having j_1 bytes with 1 transition, j_2 with 2 transitions, etc.

$$\underbrace{1,\ldots1}_{j_1}\underbrace{,2,\ldots2}_{j_2},\ldots\underbrace{8,\ldots8}_{j_8}$$

The indices j_i have the restriction $m=\sum_i i\,j_i$ where the sum extends over the range $i=1,2,\ldots8$. (Equivalently we could write $\sum_i j_i = m-\sum_i (i-1)j_i$ to eliminate one of the indices.) This pattern has a frequency given by

$$\binom{8}{1}^{j_1}\binom{8}{2}^{j_2}\cdots\binom{8}{8}^{j_8}\left(\frac{N!}{j_1!\,j_2!\ldots j_8!\left(N-\sum_i j_i\right)!}\right)$$

Without further proof, we write the distribution of transitions for the general case as

$$\binom{8N}{m}=\sum_{\{j_i\}\in P_8(m)}\left(\prod_i\binom{8}{i}^{j_i}\right)\left(\frac{N!}{\left(\prod_i j_i!\right)\left(N-\sum_i j_i\right)!}\right)$$

where the notation $\{j_i\}\in P_8(m)$ indicates that the summation is over all the partitions of the integer m with no summand greater than 8. This result is of general validity and is independent of the compression scheme used – it merely formalizes the counting of transitions.

Now for scheme A, we observe that the actual length of the compressed stream can be estimated in terms of the indices j_i. The basic idea is that getting credit for multiple transitions in a single byte shortens the length of the compressed byte stream for a given Recall our result for sparse transitions that the compressed length is $2m+1$. Now observe that any transition byte with 3 transitions instead of 1 saves 4 storage bytes (i.e. one full 00 FF cycle). Similarly, having 5 transitions in a byte saves 8 bytes, and having 7 transitions saves 12 bytes. One can verify that the even-indexed j_i also save storage, so that our revised estimate of storage is

$$\#\text{ of bytes}=(2m+1)-4j_3-4j_4-8j_5-8j_6-12j_7-12j_8$$

At this point one can in principle compute the average compression ratio of scheme A as a function of m and N.

Before doing so, however, let us explore the limitations of assuming that the number of transitions is sparse. In particular, we can easily calculate the relative frequency of sequences which contain only single transitions – precisely the sequences for which scheme A is optimum. This ratio will be denoted Q_1. From the results above, we find

$$Q_1 = \frac{\binom{8}{1}^m \binom{N}{m}}{\binom{8N}{m}} = \frac{8^m N!}{m!(N-m)!} \frac{m!(8N-m)!}{(8N)!}$$

$$= \frac{8^m N(N-1)...(N-m+1)}{8N(8N-1)...(8N-m+1)} = \prod_{j=1}^{m-1} \frac{N-j}{N-j/8}$$

For $m/N \to 0$ we find $Q_1 \to 1$. Also, for $m > N$ necessarily $Q_1 = 0$, for by the pigeonhole principle at least one byte must contain more than one transition. (Remember that the range of m is 0 to $8N$.) For example, if $N = 256$ we plot Q_1 vs. m:

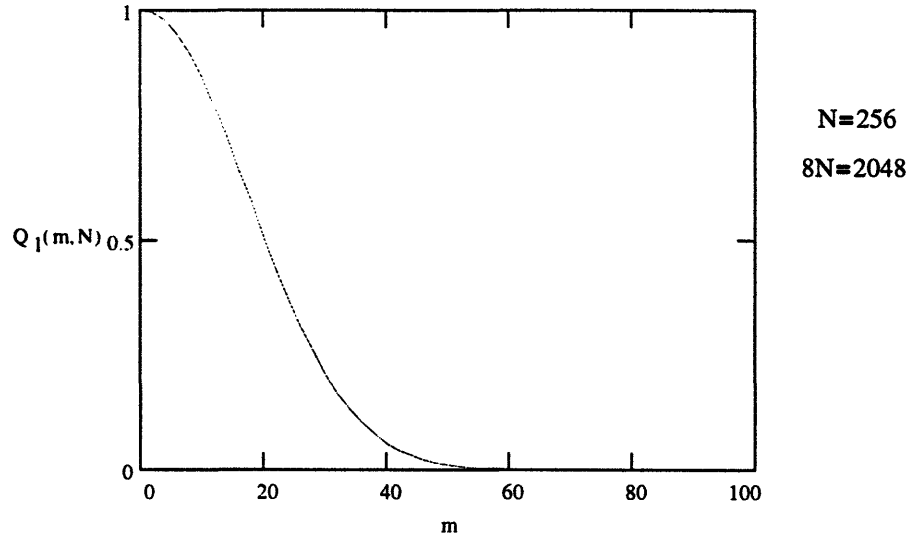


Figure F-1. The function Q_1, which gives the relative frequency of bytes that contain a single transition.

We see that the frequency of single-transition streams drops to zero long before the formal cutoff $m = N$. We expect a corresponding decrease in the effectiveness of scheme A.

Although we can drop scheme A for bit streams in which it doesn't pay, it is not clear whether any compression scheme will be effective for moderately large m – i.e. for frequent random transitions.

For completeness' sake, it should be pointed out that similar distributions Q_2, Q_3,... Q_8 can be defined. Here Q_i is the relative frequency of sequences which contain at least one byte with i transitions and no bytes with more than i transitions, expressed as a function of the total number of transitions m. For example,

$$Q_2(m,N) \equiv \frac{\sum_{j_2=1}^{m/2} \binom{8}{1}^{j_1} \binom{8}{2}^{j_2} \frac{N!}{j_1!\, j_2!\, (N-(j_1+j_2))!}}{\binom{8N}{m}}$$

with $j_1 = m - 2j_2$. A plot of Q_2 for $N = 256$ follows.

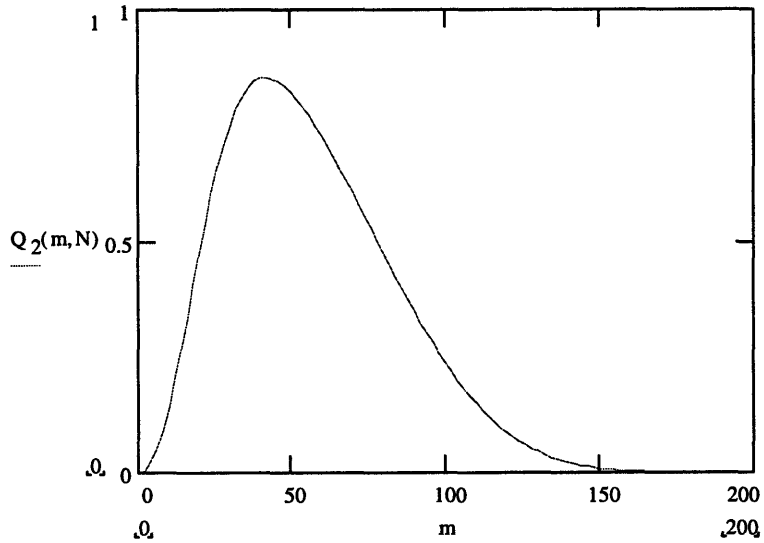


Figure F-2. The function Q_2, which gives the relative frequency of bytes with exactly two transitions.

These two distributions can be combined:

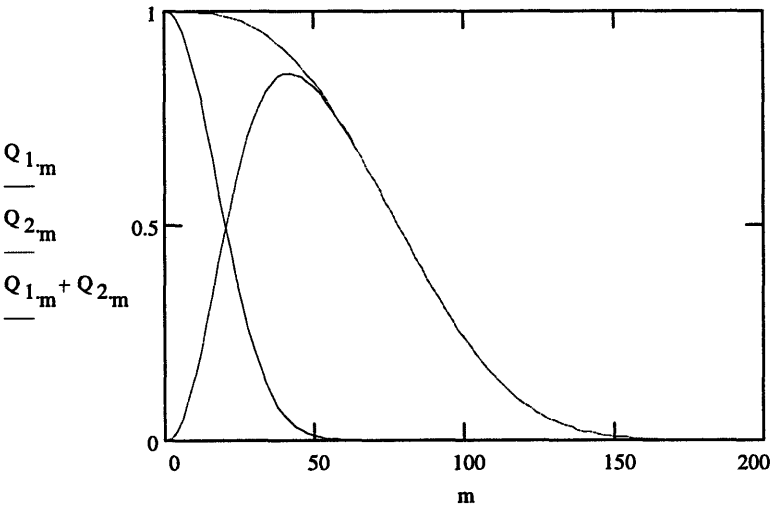


Figure F-3. The sum of Q_1 and Q_2.

Notice that, sensibly enough, Q_2 "takes over" the distribution for awhile after Q_1 peters out. Similarly, the higher order sequences each take their turn in leading the distribution as m goes through its range. The generalization of the pigeonhole principle constraint is that Q_i becomes strictly zero for $m > Ni$.

The significance of this observation is that potentially a different compression scheme can be made effective over a wider range of m if we concentrate attention on bytes with one or two transitions – in other words, there are only 58 bytes with 2 or fewer intrinsic transitions, and hence they can be coded with a few as 6 bits apiece.

" 'Classic.' A book which people praise and don't read."
– Mark Twain

Bibliography

Papers

[Andrei83] N. Andrei, K. Furuya, and J.H. Lowenstein, Rev. Mod. Phys. 55 (1983) 331.
"Solution of the Kondo Problem"

[Bardeen57] J. Bardeen, L.N. Cooper, and J.R. Schrieffer, Phys. Rev. 108 (1957) 1175.
"Theory of Superconductivity"

[Barma77] M. Barma and B.S. Shastry, Phys. Lett. 61A (1977) 15.
"d-Dimensional Hubbard Model as a (d+1)-Dimensional Classical Problem"

[Barma78] M. Barma and B.S. Shastry, Phys. Rev. B18 (1978) 3351.
"Classical equivalents of one-dimensional quantum-mechanical systems"

[Beard96a] B.B. Beard and U.-J. Wiese, MIT Preprint CTP-2509, cond-mat/9602164.
"Simulations of Discrete Quantum Systems in Continuous Euclidean Time"

[Beard96b] B.B. Beard, A. Ferrando, M. Greven, and U.-J. Wiese, MIT Preprint.
"Testing Chiral Perturbation Theory for the 2-d Spin 1 Heisenberg Antiferromagnet"

[Bednorz86] J.G. Bednorz and K.A. Müller, Z. Phys. B64 (1986) 189.
"Possible High T_c Superconductivity in the Ba-La-Cu-O System"

[Bethe31] H.A. Bethe, Z. Phys. 71 (1931) 205.
"Zur Theorie der Metalle. I. Eigenwerte und Eigenfunktionen der linearen Atomkette."

[Bietenholz95a] W. Bietenholz, A. Pochinsky, and U.-J. Wiese, Phys. Rev. Lett. 75 (1995) 4524.
"Meron-Cluster Simulation of the θ-Vacuum in the 2-d O(3)-Model"

[Bietenholz95b] W. Bietenholz, A. Pochinsky, and U.-J. Wiese, Nucl. Phys. B-S47, Lattice '95 Proceedings, (1996) 727.
"Testing Haldane's Conjecture in the O(3)-Model by a Meron Cluster Simulation"

[Bonner64] J.C. Bonner and M.E. Fisher, Phys. Rev. A135 (1964) 640.
"Linear Magnetic Chains with Anisotropic Coupling"

[Bougourzi96] A.H. Bourgourzi, M. Couture, and M. Kacir, ITP Preprint ITP-SB-96-21 (April 1996).
"Exact two-spinons dynamical correlation function of the Heisenberg model"

[Buonanno95] A. Buonanno and G. Cella, Phys. Rev. D 51 (1995) 5865.
"Swendsen-Wang update algorithm for the Symanzik improved σ model"

[Caracciolo93] S. Caracciolo et al., Nucl. Phys. B403 (1993) 475.
"Wolff-type embedding algorithms for general nonlinear σ-models"

[Caracciolo95a] S. Caracciolo et al., Phys. Rev. Lett. 74 (1995) 2969.
"Extrapolating Monte Carlo Simulations to Infinite Volume: Finite-Size Scaling at $\xi/L>>1$"

[Caracciolo95b] S. Caracciolo et al., Phys. Rev. Lett. 75 (1995) 1891.
"Asymptotic Scaling in the Two-Dimensional O(3) σ Model at Correlation Length 10^5"

[Caracciolo95c] S. Caracciolo et al., Nucl. Phys. B-S42, Lattice '94 Proceedings, (1995) 749.
"New method for the extrapolation of finite-size data to infinite volume"

[Caracciolo95d] S. Caracciolo et al., Nucl. Phys. B-S42, Lattice '94 Proceedings, (1995) 752.
"Two-dimensional O(3) σ-model up to correlation length 10^5"

[Chakravarty82] S. Chakravarty and D.B. Stein, Phys. Rev. Lett. 49 (1982) 582.
"Monte Carlo Simulation of Quantum Spin Systems"

[Chakravarty88] S. Chakravarty, B.I. Halperin, and D.R. Nelson, Phys. Rev. Lett. 60 (1988) 1057.
"Low-Temperature Behavior of Two-Dimensional Quantum Antiferromagnets"

[Chakravarty89] S. Chakravarty, B.I. Halperin, and D.R. Nelson, Phys. Rev. B39 (1989) 2344.
"Two-dimensional quantum Heisenberg antiferromagnet at low temperatures"

[Cooper82] F. Cooper, B. Freedman, and D. Preston, Nucl. Phys. B210 (1982) 210.
"Solving $\phi^4_{1,2}$ field theory with Monte Carlo"

[Cullen83] J.J. Cullen and D.P. Landau, Phys. Rev. B27 (1983) 297.
"Monte Carlo studies of one-dimensional quantum Heisenberg and XY models"

[DeDivitiis95] G. de Divitiis et al. (ALPHA Collaboration), Nucl. Phys. B437 (1995) 447.
"Universality and the approach to the continuum limit in lattice gauge theory"

[D'Elia95] M. D'Elia, F. Farchioni, and A. Papa, Nucl. Phys. B456 (1995) 313.
"Scaling and Topology in the 2-d O(3) σ-model on the lattice with the fixed point action"

[DeRaedt85] H. de Raedt and A. Lagendijk, Phys. Rep. 127 (1985) 233.
"Monte Carlo Simulation of Quantum Statistical Lattice Models"

[Evertz93] H.G. Evertz, G. Lana, and M. Marcu, Phys. Rev. Lett. 70 (1993) 875.
"Cluster Algorithms for Vertex Models"

[Farhi92] E. Farhi and S. Gutmann, Ann. Phys. 213 (1992) 182.
"The Functional Integral Constructed Directly from the Hamiltonian"

[Galli96a] A. Galli, hep-lat/9605007.
"A new approach to significantly reduce the negative sign problem in quantum Monte Carlo"

[Galli96b] A. Galli, hep-lat/9605026.
"Suppression of the negative sign problem in quantum Monte Carlo"

[Greven94] M. Greven et al., Phys. Rev. Lett. 72 (1994) 1096.
"Spin Correlations in the 2D Heisenberg Antiferromagnet $Sr_2CuO_2Cl_2$: Neutron Scattering, Monte Carlo Simulation, and Theory"

[Greven95a] M. Greven et al., Z. Phys. B96 (1995) 465.
"Neutron scattering study of the two-dimensional spin s=½ square-lattice Heisenberg antiferromagnet $Sr_2CuO_2Cl_2$"

[Greven95b] M. Greven, Doctoral Thesis, MIT (May 1995).
"Neutron Scattering Study of Magnetism in Insulating and Superconducting Lamellar Copper Oxides"

[Greven96] M. Greven et al., MIT Preprint.
"Testing Asymptotic Scaling of the Correlation Length for the 2-d Spin ½ Heisenberg Antiferromagnet"

[Handscomb62] D.C. Handscomb, Proc. Cambridge Philos. Soc.58 (1962) 594.
"The Monte Carlo Method in Quantum Statistical Mechanics"

[Handscomb64] D.C. Handscomb, Proc. Cambridge Philos. Soc.60 (1964) 115.
"A Monte Carlo method applied to the Heisenberg ferromagnet"

[Hasenbusch95] M. Hasenbusch, Nucl. Phys. B-S42, Lattice '94 Proceedings, (1995) 764.
"An Improved Estimator for the Correlation Function of 2D Nonlinear Sigma Models"

[Hasenfratz90a] P. Hasenfratz and H. Leutwyler, Nucl. Phys. B343 (1990) 241.
"Goldstone Boson related finite size effects in field theory and critical phenomena with $O(N)$ symmetry"

[Hasenfratz90b] P. Hasenfratz, M. Maggiore, and F. Niedermayer, Phys. Lett. B245 (1990) 522.
"The exact mass gap of the $O(3)$ and $O(4)$ non-linear σ-models in d=2"

[Hasenfratz90c] P. Hasenfratz and F. Niedermayer, Phys. Lett. B245 (1990) 529.
"The exact mass gap of the $O(N)$ σ-model for arbitrary $N \geq 3$ in d=2"

[Hasenfratz91] P. Hasenfratz and F. Niedermayer, Phys. Lett. B268 (1991) 231.
"The exact correlation length of the antiferromagnetic d=2+1 Heisenberg model at low temperatures"

[Hasenfratz93] P. Hasenfratz and F. Niedermayer, Z. Phys. B92 (1993) 91.
"Finite size and temperature effects in the AF Heisenberg model"

[Heisenberg28] W. Heisenberg, Z. Phys. 49 (1928) 619.
"Zur Theorie des Ferromagnetismus"

[Hirsch81] J.E. Hirsch, D.J. Scalapino, and R.L. Sugar, Phys. Rev. Lett. 47 (1981) 1628.
"Efficient Monte Carlo Procedure for Systems with Fermions"

[Hubbard59] J. Hubbard, Phys. Rev. Lett. 3 (1959) 77.
"Calculation of Partition Functions"

[Hubbard63a] J. Hubbard, Proc. Royal Soc. London, Ser. A 276 (1963) 238.
"Electron correlations in narrow energy bands"

[Hubbard63b] J. Hubbard, Proc. Royal Soc. London, Ser. A 276 (1963) 238.
"Electron correlations in narrow energy bands: II. The degenerate band case"

[Hubbard64] J. Hubbard, Proc. Royal Soc. London, Ser. A 276 (1963) 238.
"Electron correlations in narrow energy bands: III. An improved solution"

[Kim93] J.-K. Kim, Phys. Rev. Lett. 70 (1993) 1735.
"Application of Finite Size Scaling to Monte Carlo Simulations"

[Kim94a] J.-K. Kim, Phys. Rev. D 50 (1994) 4663.
"Asymptotic scaling of the mass gap in the two-dimensional O(3) nonlinear σ model: A numerical study"

[Kim94b] J.-K. Kim, Nucl. Phys. B-S34, Lattice '93 Proceedings, (1994) 702.
"Estimating bulk values based on finite size scaling"

[Kim95] J.-K. Kim, Phys. Lett. B345 (1995) 469.
"Scaling behavior of the four-point renormalized coupling constant in the two dimensional O(2) and O(3) non-linear sigma models"

[Kim96] J.-K. Kim, A.J.F. de Souza, and D.P. Landau, cond-mat/9602060.
"Numerical Computation of Finite-Size Scaling Functions: An Alternative Approach to Finite Size Scaling"

[Lüscher93] M. Lüscher, DESY Preprint 93-133 (hep-lat/930920).
"A Portable High-Quality Random Number Generator for Lattice Field Theory Simulations"

[Lyklema82] J.W. Lyklema, Phys. Rev. Lett. 49 (1982) 88.
"Quantum-Statistical Monte Carlo Method for Heisenberg Spins"

[Lyklema83] J.W. Lyklema, Phys. Rev. B27 (1983) 3108.
"Monte Carlo study of the one-dimensional quantum Heisenberg ferromagnet near T=0"

[Makivic91] M.S. Makivic´ and H.-Q. Ding, Phys. Rev. B43 (1991) 3562.
"Two-dimensional spin-1/2 Heisenberg antiferromagnet: A quantum Monte Carlo study"

[Metropolis53] N. Metropolis, A. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, J. Chem. Phys. 21 (1953) 1087.
"Equations of state calculations by fast computing machines"

[Sandvik96a] A.W. Sandvik, preprint.
"Using covariance to increase the accuracy of quantum Monte Carlo simulations – An improved estimate for the sublattice magnetisation of the 2D Heisenberg antiferromagnet"

[Sandvik96b] A.W. Sandvik, preprint.
"Error reduction using covariance in quantum Monte Carlo simulations"

[Shirane87] G. Shirane et al., Phys. Rev. Lett. 59 (1987) 1613.
"Two-dimensional Antiferromagnetic Quantum Spin-Fluid State in La_2CuO_4"

[Suzuki76a] M. Suzuki, Prog. Theor. Phys. 56 (1976) 1454.
"Relationship between d-Dimensional Quantal Spin Systems and (d+1)-Dimensional Ising Systems"

[Suzuki76b] M. Suzuki, Commun. Math. Phys. 51 (1976) 183.
"Generalized Trotter's Formula and Systematic Approximants of Exponential Operators and Inner Derivations with Applications to Many-Body Problems"

[Suzuki77] M. Suzuki, Commun. Math. Phys. 57 (1977) 193.
"On the Convergence of Exponential Operators – the Zassenhaus Formula, BCH Formula and Systematic Approximants"

[Suzuki84] M. Suzuki, J. Math. Phys. 26 (1985) 601.
"Decomposition formulas of exponential operators and Lie exponentials with some applications to quantum mechanics and statistical physics"

[Suzuki85] M. Suzuki, Phys. Rev. B31 (1985) 2957.
"Transfer-Matrix method and Monte Carlo simulation in quantum spin systems"

[Swendsen87] R.H. Swendsen and J.-S. Wang, Phys. Rev. Lett. 58 (1087) 86.
"Nonuniversal Critical Dynamics in Monte Carlo Simulations"

[Weinberg66] S. Weinberg, Phys. Rev. Lett. 17 (1966) 616.
"Pion scattering lengths"

[Weinberg68] S. Weinberg, Phys. Rev. 166 (1968) 1586.
"Nonlinear realizations of Chiral Symmetry"

[Wiese92] U.-J. Wiese and H.-P. Ying, Phys. Lett. A168 (1992) 143.
"Blockspin cluster algorithms for quantum spin systems"

[Wiese93] U.-J. Wiese, Phys. Lett. B311 (1993) 235.
"Bosonization and cluster updating of lattice fermions"

[Wiese94] U.-J. Wiese and H.-P. Ying, Z. Phys. B93 (1994) 147.
"A determination of the low energy parameters of the 2-d Heisenberg antiferromagnet"

[Wolff89] U. Wolff, Phys. Rev. Lett. 62 (1989) 361.
"Collective Monte Carlo Updating for Spin Systems"

[Wu87] M.K. Wu, J.R. Ashburn, C.J. Torny, P.H. Hor, R.L. Meng, L. Gao, Z.J. Huang, Y.Q. Wang, and C.W. Chu, Phys. Rev. Lett. 58 (1987) 908.

[Ying93] H.-P. Ying, U.-J. Wiese, and D.-R. Ji, Phys. Lett. A183 (1993) 441.
"Loop-Cluster algorithm: an application for the 2D quantum Heisenberg antiferromagnet"

Books

[Arfken85] G. Arfken, "Mathematical Methods for Physicists" (Academic Press, San Diego) 1985.

[Das93] A. Das, "Field Theory: A Path Integral Approach" (World Scientific, Singapore) 1993.

[Feynman65] R.P. Feynman and A.R. Hibbs, "Quantum Mechanics and Path Integrals" (McGraw-Hill, New York) 1965.

[Feynman72] R.P. Feynman, "Statistical Mechanics" (Addison-Wesley, Reading) 1972.

[Goldenfeld92] N. Goldenfeld, "Lectures on Phase Transitions and the Renormalization Group" (Addison-Wesley, reading) 1992.

[Georgi84] H. Georgi, "Weak Interactions and Modern Particle Theory" (Benjamin Cummings, Menlo Park) 1984.

[Guidry91] M. Guidry, "Gauge Field Theories" (John Wiley, New York) 1991.

[Halley88] J.W. Halley, Ed. "Theories of High-Temperature Superconductivity" (Addison-Wesley, Redwood City) 1988.

[Hammersley64] Hammersley and Handscomb, "Monte Carlo Methods" (Methuen, London) 1964.

[Hill60] T.L. Hill, "An introduction to Statistical Thermodynamics" (Dover, New York) 1986.

[Huang87] K. Huang, "Statistical Mechanics" 2nd Ed. (John Wiley, New York) 1987.

[Ibach95] H. Ibach and H. Lüth, "Solid-State Physics" (Springer, Berlin) 1995.

[Jones73] W. Jones and N.H. March, "Theoretical Solid State Physics" (Dover, New York) 1985.

[Kaku93] M. Kaku, "Quantum Field Theory" (Oxford University Press, New York) 1993.

[Kittel87] C. Kittel, "Quantum Theory of Solids" (John Wiley, NewYork) 1987.

[Landau59] L.D. Landau and E.M. Lifshitz, "Statistical Physics" (Pergamon, Oxford) 1980.

[Ma85] S.-K. Ma, "Statistical Mechanics" (World Scientific, Philadelphia) 1985.

[March67] N.H. March, W.H. Young, and S. Sampanthar, "The Many-Body Problem in Quantum Mechanics" (Dover, New York) 1995.

[Negele88] J.W. Negele and H. Orland, "Quantum Many-Particle Systems" (Addison-Wesley, Reading) 1988.

[Nelson92] M. Nelson, "The Data Compression Book" (M&T, New York) 1992.

[Pathria72] R.K. Pathria, "Statistical Mechanics" (Pergamon, Oxford) 1972.

[Peskin95] Peskin and Schroeder, "An Introduction to Quantum Field Theory" (Addison-Wesley, Reading) 1995.

[Press86] W.H. Press et al., "Numerical Recipes" (Cambridge University Press, Cambridge) 1986.

[Sakurai85] J.J. Sakurai, "Modern Quantum Mechanics" (Benjamin Cummings, Menlo Park) 1985.

[Sternberg94] S. Sternberg, "Group Theory and Physics" (Cambridge University Press, Cambridge) 1994.

[Swanson92] M.S. Swanson, "Path Integrals and Quantum Processes" (Academic Press, Boston) 1992.

Seventeen Haiku

I.
So small is h bar
So vast is the speed of light
Both one for all time

II.
Bare and dressed masses
Renormalization group
Shows how you relate

III.
In-states and out-states
All that is? and Is that all?
Virtual no more

IV.
Fadeev-Popov
Ghosts of unitarity
Haunt the between-time

V.
Lonely graviton
Trek across the universe
And find no there there

VI.
Child's top falls over
Rests on the table tonight
Electron? No rest!

VII.
Singlet and triplet
Two spins joined but separate
Can't tell up or down

VIII.
One sees two sees three...
Until the spin chain closes
Then bosons make waves

IX.
Perfect solid sphere
Moment of inertia is
$2/5\ M\ R^2$

X.
Minkowski space-time!
Stress-energy density
Warps you forever!

XI.
Solar neutrinos
Does your deficit tell us
How the world will end?

XII.
Nucleon structure
Probed with tiny leptons, and
Parallel machines

XIII.
Full of surprises:
Antiferromagnetic
Heisenberg Model

XIV.
No new paradigms –
Physics sees evolution,
Not revolution

XV.
Zen master Dogen
Laughs at our perfect action
But we think it's good!

XVI.
Sliding plank problem?
Remember how to solve it?
Bane of grad students....

XVII.
Was, Will be, Elsewhere
Three invariant domains
All meet at one point